AME Scripting API Guide

Release 1.0

Adobe Incorporated

Feb 08, 2024

GUIDE

Welcome to the AME Scripting Guide!

This guide shows you how to use ExtendScript to script AME and how to automate recurring tasks.

The first part is about concepts & techniques, the second part is an alphabetical reference with all objects, methods, and properties described in full detail, including parameter types, return values, default values, explanations of parameter values, and full code examples that can be used as a starting point.

CHAPTER

ONE

AME SCRIPTING GUIDE

This is a community-created and -maintained project documenting Extendscript usage for Adobe Media Encoder.

1.1 How do I run scripts in AME?

Similarly to Premiere Pro, you can use Visual Studio Code and the "ExtendScript Debugger" extension to send scripts from VS Code to AME and debug them in the app. Open the script, select the target and run or debug the script.

You can also launch scripts from the command line on Mac and Windows, like this

```
<fullPathToAMEbinary> --console es.processFile <fullPathToScript>
```

Example for executing a test script on Mac:

```
"/Applications/Adobe Media Encoder (Beta)/Adobe Media Encoder (Beta).app/
 Contents/MacOS/Adobe Media Encoder (Beta)"
 --console es.processFile ~/Desktop/test.js
```

1.2 What is a good starting point to understand the scripting objects?

Let's start with a very basic script:

```
// make sure to replace "\\" by "/" on Mac with a valid path
var source = "D:\\full\\path\\to\\camera3.mxf";
var preset = "D:\\full\\path\\to\\AME\\MediaIO\\systempresets\\58444341_4d584658\\
↪XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\full\\path\\to\\Output\\test";

var exporter = app.getExporter();

if (exporter) {
   var encoderWrapper  = exporter.exportItem(source, destination, preset);

   exporter.addEventListener("onEncodeComplete", function(eventObj) {
        // We can get the encoding status from the event or from the exporter
        $.writeln("Encode Complete Status: " + eventObj.encodeCompleteStatus);

        var encodeSuccess = exporter.encodeSuccess;
```

(continues on next page)

(continued from previous page)

```
        $.writeln("Encode Complete Status alt: " + encodeSuccess);
    }, false)

    exporter.addEventListener("onError", function(eventObj) {
        // We can get the encoding status from the event or from the exporter
        $.writeln("Error while encoding");

        var encodeSuccess = exporter.encodeSuccess;
        $.writeln("Encode Complete Status: " + encodeSuccess);
    }, false)

}
```

In order to encode a source file in AME, you need to provide the paths of the source file and destination folder, and the preset to be used.

The event listener for `onEncodeComplete` will be called once the encode has successfully finished.

1.3 Where can I ask more questions and get help?

Got more questions than what is covered here? Head over to the Adobe Media Encoder forum here: https://community.adobe.com/t5/adobe-media-encoder/ct-p/ct-media-encoder

CHAPTER

TWO

ADOBE MEDIA ENCODER (AME) SCRIPTING API REFERENCE (A-Z)

Revision date: 2023-09-04

2.1 AMEBatchItemCreationFailedEvent

The event will be sent after batch item creation failed. Can be used for the following FrontendScriptObject API's: 'addFileToBatch', 'addTeamProjectsItemToBatch' and 'addDLToBatch'.

2.1.1 Properties

- `error: string` : Get the error string
- `onBatchItemCreationFailed: constant string` : Notify when the batch item creation failed.
- `srcFilePath: string` : Get the source file path.

2.1.2 Code Samples

```
var source = "C:\\testdata\\testmedia.mp4";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia.mp4"

var frontend = app.getFrontend();
if (frontend) {
  frontend.addEventListener("onBatchItemCreationFailed", function (eventObj) {
    $.writeln("Sourcefile", eventObj.srcFilePath);
    $.writeln("onBatchItemCreationFailed: error", eventObj.error);
  });

  var batchItemSuccess = frontend.addItemToBatch(source);
  if (batchItemSuccess) {
    $.writeln(source, " has been added successfully");
  }
}
```

2.2 AMEExportEvent

Provides the following event types: onMediaInfoCreated, onBatchItemStatusChanged, onItemEncodingStarted, onAudioPreEncodeProgress, onEncodingItemProgressUpdated, onEncodeComplete, onError, onPostProcessListInitialized

2.2.1 Properties

- `audioInfo: string` : Returns the audio pre-encoding info for the event type onAudioPreEncodeProgress (available since 24.0).
- `audioProgress: float` : Returns the audio pre-encoding progress for the event type onAudioPreEncodeProgress (available since 24.0).
- `encodeCompleteStatus: bool` : Returns true after encoding has been completed for a batch item. Can be called for onEncodeComplete event.
- `encodeCompleteTime: float` : Returns the encoding time in milliseconds. Can be called for onEncodeComplete event.
- `groupIndex: unsigned int` : Returns the batch group index. Can be called for onBatchItemStatusChanged event.
- `itemIndex: unsigned int` : Returns the batch item index. Can be called for onBatchItemStatusChanged event.
- `onAudioPreEncodeProgress: constant string` : Notify when the audio pre-encode progress changes (available since 24.0)
- `onBatchItemStatusChanged: constant string` : Notify when batch item status has been changed. You can call the API's groupIndex, itemIndex and status for more info.
- `onEncodeComplete: constant string` : Notify when the batch item has been encoded. You can call the API's encodeCompleteStatus and encodeCompleteTime for more info.
- `onEncodingItemProgressUpdated: constant string` : Notify the encoding progress.
- `onError: constant string` : Notify when there's an error while encoding the batch item.
- `onItemEncodingStarted: constant string` : Notify when the encoding of a batch item has started.
- `onMediaInfoCreated: constant string` : Notify when media info has been created.
- `onPostProcessListInitialized: constant string` : Notify when the post process list is initialized.
- `progress: float` : Returns the batch item encoding progress value which is between 0 and 1. Can be called for onEncodingItemProgressUpdated event
- `status: unsigned int` : Returns the batch item status. 0 : Waiting, 1 : Done, 2 : Failed, 3 : Skipped, 4 : Encoding, 5 : Paused, 6 : Stopped, 7 : Any, 8 : AutoStart, 9 : Done Warning, 10 : Watch Folder Waiting. Can be called for onBatchItemStatusChanged event.

2.2.2 Code Samples

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();
if (exporter) {
  exporter.addEventListener(
    "onEncodeComplete",
    function (eventObj) {
      $.writeln("Encode Complete Status: " + eventObj.encodeCompleteStatus);
    },
    false
  );

  // Alternatively you can access the correct name of that event via the following const␣
↪property:
  var encodeCompleteEvent = AMEExportEvent.onEncodeComplete;
  exporter.addEventListener(
    encodeCompleteEvent,
    function (eventObj) {
      $.writeln(
        "Encode Complete Status (alt): " + eventObj.encodeCompleteStatus
      );
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();
if (exporter) {
  exporter.addEventListener(
    "onEncodeComplete",
    function (eventObj) {
      $.writeln(
        "Encode Complete Time in milli seconds: " + eventObj.encodeCompleteTime
```

(continues on next page)

(continued from previous page)

```
      );
    },
    false
  );

  // Alternatively you can access the correct name of that event via the following const␣
↪property:
  var encodeCompleteEvent = AMEExportEvent.onEncodeComplete;
  exporter.addEventListener(
    encodeCompleteEvent,
    function (eventObj) {
      $.writeln(
        "Encode Complete Time in milli seconds: (alt): " +
          eventObj.encodeCompleteTime
      );
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var batchItemStatusChangedEvent = AMEExportEvent.onBatchItemStatusChanged;
$.writeln(
  "Event name is identical with the const property API name ('onBatchItemStatusChanged
↪'): " +
    batchItemStatusChangedEvent
);
var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    batchItemStatusChangedEvent,
    function (eventObj) {
      $.writeln("Batch group index: " + eventObj.groupIndex);
      $.writeln("Batch item index: " + eventObj.itemIndex);
      //Possible status values:
      //kBatchItemStatus_Waiting = 0,
      //kBatchItemStatus_Done,
      //kBatchItemStatus_Failed,
      //kBatchItemStatus_Skipped,
      //kBatchItemStatus_Encoding,
      //kBatchItemStatus_Paused,
```

(continues on next page)

(continued from previous page)

```
      //kBatchItemStatus_Stopped,
      //kBatchItemStatus_Any,
      //kBatchItemStatus_AutoStart,
      //kBatchItemStatus_Done_Warning,
      //kBatchItemStatus_WatchFolderWaiting
      $.writeln("Batch item status: " + eventObj.status);
    },
    false
  );

  // Alternatively you can listen to "onBatchItemStatusChanged"
  exporter.addEventListener(
    "onBatchItemStatusChanged",
    function (eventObj) {
      $.writeln("Batch group index (alt): " + eventObj.groupIndex);
      $.writeln("Batch item index (alt): " + eventObj.itemIndex);
      //Possible status values:
      //kBatchItemStatus_Waiting = 0,
      //kBatchItemStatus_Done,
      //kBatchItemStatus_Failed,
      //kBatchItemStatus_Skipped,
      //kBatchItemStatus_Encoding,
      //kBatchItemStatus_Paused,
      //kBatchItemStatus_Stopped,
      //kBatchItemStatus_Any,
      //kBatchItemStatus_AutoStart,
      //kBatchItemStatus_Done_Warning,
      //kBatchItemStatus_WatchFolderWaiting
      $.writeln("Batch item status (alt): " + eventObj.status);
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var encodeCompleteEvent = AMEExportEvent.onEncodeComplete;

$.writeln(
  "Event name is identical with the const property API name ('onEncodeComplete'): " +
    encodeCompleteEvent
);
var exporter = app.getExporter();
```

(continues on next page)

(continued from previous page)

```
if (exporter) {
  exporter.addEventListener(
    encodeCompleteEvent,
    function (eventObj) {
      $.writeln("Encode Complete Status: " + eventObj.encodeCompleteStatus);
      $.writeln(
        "Encode Complete Time (in milli seconds): " +
          eventObj.encodeCompleteTime
      );
    },
    false
  );

  // Alternatively you can listen to "onEncodeComplete"
  exporter.addEventListener(
    "onEncodeComplete",
    function (eventObj) {
      $.writeln(
        "Encode Complete Status (alt): " + eventObj.encodeCompleteStatus
      );
      $.writeln(
        "Encode Complete Time in milli seconds (alt): " +
          eventObj.encodeCompleteTime
      );
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var encodingItemProgressUpdatedEvent =
  AMEExportEvent.onEncodingItemProgressUpdated;
$.writeln(
  "Event name is identical with the const property API name (
↪'onEncodingItemProgressUpdated'): " +
    encodingItemProgressUpdatedEvent
);
var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
```

(continues on next page)

(continued from previous page)

```
    encodingItemProgressUpdatedEvent,
    function (eventObj) {
      $.writeln("Encoding progress for batch item: " + eventObj.progress);
    },
    false
  );

  // Alternatively you can listen to "onEncodingItemProgressUpdated"
  exporter.addEventListener(
    "onEncodingItemProgressUpdated",
    function (eventObj) {
      $.writeln("Encoding progress for batch item (alt): " + eventObj.progress);
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var errorEvent = AMEExportEvent.onError;
$.writeln(
  "Event name is identical with the const property API name ('onError'): " +
    errorEvent
);

var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    errorEvent,
    function (eventObj) {
      $.writeln("Error while encoding");
    },
    false
  );

  // Alternatively you can listen to "onError"
  exporter.addEventListener(
    "onError",
    function (eventObj) {
      $.writeln("Error while encoding (alt)");
    },
    false
```

(continues on next page)

(continued from previous page)

```
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var itemEncodingStartedEvent = AMEExportEvent.onItemEncodingStarted;
$.writeln(
  "Event name is identical with the const property API name ('onItemEncodingStarted'): "␣
↪+
    itemEncodingStartedEvent
);
var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    itemEncodingStartedEvent,
    function (eventObj) {
      $.writeln("Encoding started for batch item.");
    },
    false
  );

  // Alternatively you can listen to "onItemEncodingStarted"
  exporter.addEventListener(
    "onItemEncodingStarted",
    function (eventObj) {
      $.writeln("Encoding started for batch item (alt).");
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";
```

(continues on next page)

(continued from previous page)

```
var mediaInfoCreatedEvent = AMEExportEvent.onMediaInfoCreated;
$.writeln(
  "Event name is identical with the const property API name ('onMediaInfoCreated'): " +
    mediaInfoCreatedEvent
);
var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    mediaInfoCreatedEvent,
    function (eventObj) {
      $.writeln("Media info created");
    },
    false
  );

  // Alternatively you can listen to "onMediaInfoCreated"
  exporter.addEventListener(
    "onMediaInfoCreated",
    function (eventObj) {
      $.writeln("Media info created (alt)");
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var postProcessListInitializedEvent =
  AMEExportEvent.onPostProcessListInitialized;
$.writeln(
  "Event name is identical with the const property API name (
↪'onPostProcessListInitialized'): " +
    postProcessListInitializedEvent
);
var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    postProcessListInitializedEvent,
    function (eventObj) {
      $.writeln("Post process list has been initialized.");
```

(continues on next page)

(continued from previous page)

```
    },
    false
  );

  // Alternatively you can listen to "onPostProcessListInitialized"
  exporter.addEventListener(
    "onPostProcessListInitialized",
    function (eventObj) {
      $.writeln("Post process list has been initialized (alt).");
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    "onEncodingItemProgressUpdated",
    function (eventObj) {
      $.writeln("Encoding progress for batch item: " + eventObj.progress);
    },
    false
  );

  // Alternatively you can access the correct name of that event via the following const␣
↪property:
  var encodingItemProgressUpdatedEvent =
    AMEExportEvent.onEncodingItemProgressUpdated;
  exporter.addEventListener(
    encodingItemProgressUpdatedEvent,
    function (eventObj) {
      $.writeln("Encoding progress for batch item (alt): " + eventObj.progress);
    },
    false
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();

if (exporter) {
  exporter.addEventListener(
    "onBatchItemStatusChanged",
    function (eventObj) {
      //Possible status values:
      // 0 : Waiting
      // 1 : Done
      // 2 : Failed
      // 3 : Skipped
      // 4 : Encoding
      // 5 : Paused
      // 6 : Stopped
      // 7 : Any
      // 8 : AutoStart
      // 9 : Done Warning
      // 10 : Watch Folder Waiting.
      $.writeln("Batch item status: " + eventObj.status);
    },
    false
  );

  // Alternatively you can access the correct name of that event via the following const␣
↪property:
  var batchItemStatusChangedEvent = AMEExportEvent.onBatchItemStatusChanged;
  exporter.addEventListener(
    batchItemStatusChangedEvent,
    function (eventObj) {
      //Possible status values:
      // 0 : Waiting
      // 1 : Done
      // 2 : Failed
      // 3 : Skipped
      // 4 : Encoding
      // 5 : Paused
      // 6 : Stopped
      // 7 : Any
      // 8 : AutoStart
      // 9 : Done Warning
      // 10 : Watch Folder Waiting.
      $.writeln("Batch item status (alt): " + eventObj.status);
    },
    false
```

(continues on next page)

(continued from previous page)

```
  );

  var encoderWrapper = exporter.exportItem(source, destination, preset);
}
```

2.3 AMEFrontendEvent

The event will be sent after a batch item has been created successfully.

2.3.1 Properties

- `onItemAddedToBatch: constant string` : Notify when a batch item has been created successfully. Can be used for all FrontendScriptObject API's which creates a batch item.

2.3.2 Code Samples

```
var source = "C:\\testdata\\testmedia.mp4";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia.mp4"

var frontend = app.getFrontend();
if (frontend) {
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("Item added to Batch");
  });

  var batchItemSuccess = frontend.addItemToBatch(source);
  if (batchItemSuccess) {
    $.writeln(source, " has been added successfully");
  }
}
```

2.4 Application

Top level app object

2.4.1 Properties

- `buildNumber: string` : Get application build number

2.4.2 Methods

- `assertToConsole(): bool` : Redirect assert output to stdout.
- `bringToFront(): bool` : Bring application to front
- `cancelTask(taskID: int): bool` : Cancel the task that matches the task ID
- `getEncoderHost(): scripting object` : Get the encoder host object. See EncoderHostScriptObject
- `getExporter(): scripting object` : Get the exporter object. See ExporterScriptObject
- `getFrontend(): scripting object` : Get the front end object. See FrontendScriptObject
- `getWatchFolder(): scripting object` : Get the watch folder object. See WatchFolderScriptObject
- `isBlackVideo(sourcePath: string): bool` : True if all frames are black
- `isSilentAudio(sourcePath: string): bool` : True if audio is silent
- `quit(): bool` : Quit the AME app
- `renderFrameSequence(sourcePath: string, outputPath: string, renderAll: bool, startFrame: int): bool` : Render still frames for given source
- `scheduleTask(scriptToExecute: string, delayInMilliseconds: int, repeat: bool): int` : Schedule a script to run after delay, returns task ID
  - `scriptToExecute`: Put your script as text, e.g. 'app.getEncoderHost().runBatch()'.
- `wait(milliseconds: unsigned int): bool` : Non UI blocking wait in milliseconds
- `write(text: string): bool` : Write text to std out

2.4.3 Code Samples

```
var exporter = app.getExporter();
// check ExporterScriptObject to see which methods/properties you can apply
```

```
var source = "C:\\testdata\\testmedia.mp4";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia.mp4"

var blackVideo = app.isBlackVideo(source);
if (blackVideo) {
  $.writeln("The input file has only black frames");
}
```

```
var source = "C:\\testdata\\testmedia.mp4";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia.mp4"
```

(continues on next page)

(continued from previous page)

```
var silent = app.isSilentAudio(source);
if (silent) {
  $.writeln("The input file has no audio");
}
```

```
var source = "C:\\testdata\\testmedia4.mp4";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var destination = "/Users/Shared/testdata/outputFolder/output.mp4;

var renderall = true;
var startTime = 0;
var success = app.renderFrameSequence(
  source,
  destination,
  renderall,
  startTime
);
if (success) {
  $.writeln("renderFrameSequence() successfully done");
}
```

```
var format = "";
var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HighQuality720HD.epr";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";

var frontend = app.getFrontend();
if (frontend) {
  // Either format or preset can be empty, output is optional
  var encoderWrapper = frontend.addFileToBatch(source, format, preset);

  if (encoderWrapper) {
    var taskID = app.scheduleTask(
      "var e = app.getEncoderHost(); e.runBatch()",
      5000,
      false
    );
  } else {
    $.writeln("Encoder wrapper object is not valid.");
  }
} else {
  $.writeln("Frontend object is not valid.");
}
```

2.5 EncoderHostScriptObject

Provides several utility methods including batch commands to run, pause or stop the batch.

2.5.1 Methods

- `createEncoderForFormat(inFormatName: string): scripting object` : Returns an 'Encoder-Wrapper' script object for the requested format.
- `getBatchEncoderStatus(): string` : Returns the current status of the batch encoder. The values are: invalid, paused, running, stopped, stopping (available since 23.3).
- `getCurrentBatchPreview(inOutputPath: string): bool` : Writes out the current batch preview image (tiff format) to the given path.
  - `inOutputPath`: Path to store a 'tiff' file.
- `getFormatList(): array of strings` : Returns a list of all available formats.
- `getSourceInfo(sourcePath: string): scripting object` : Returns a 'SourceMediaInfo' script object which can give detailed info about the provided source.
  - `sourcePath`: Media path
- `getSupportedImportFileTypes(): array of strings` : Returns list of all available formats.
- `isBatchRunning(): bool` : Returns true if a batch job is running.
- `pauseBatch(): bool` : Pauses the batch (always returns true).
- `runBatch(): bool` : Runs the batch (always returns true).
- `stopBatch(): bool` : Stops the batch (always returns true).

2.5.2 Code Samples

```
var format = "H.264"; // e.g. H.264
var source = "C:\\testdata\\testmedia1.mxf";
var outputFile = "C:\\testdata\\outputFolder\\output.tiff";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia1.mxf"
// var outputFile = "/Users/Shared/testdata/outputFolder/output.tiff";

var encoderHost = app.getEncoderHost();

if (encoderHost) {
  encoderHost.addEventListener(
    "onBatchEncoderStatusChanged",
    function (eventObj) {
      $.writeln(
        "onBatchEncoderStatusChanged to status: " + eventObj.batchEncoderStatus
      );
    }
  );
```

(continues on next page)

(continued from previous page)

```
  // API "getSourceInfo"
  var sourceMediaInfo = encoderHost.getSourceInfo(source);
  if (sourceMediaInfo) {
    // For 'sourceMediaInfo' you can now call properties of the 'SourceMediaInfo' script␣
↪object, e.g.:
    // (See detailed info in the documentation of 'SourceMediaInfo')
    $.writeln(
      "Embedded description of the source: " + sourceMediaInfo.description
    );
  }

  // API "getFormatList"
  var formatList = encoderHost.getFormatList();
  $.writeln("formatList: " + formatList);

  // API "createEncoderForFormat"
  var encoderWrapper = encoderHost.createEncoderForFormat(format);
  if (encoderWrapper) {
    // For 'encoder' you can now call properties/methods of the 'EncoderWrapper" script␣
↪object, e.g.:
    // (See detailed info in the documentation of 'EncoderWrapper')
    var frameRate = "25";
    encoderWrapper.setFrameRate(frameRate);
  }

  // API "isBatchRunning"
  var isBatchRunning = encoderHost.isBatchRunning();
  // With the current script the return value should be 'false' since no batch (job) is␣
↪running.
  // After adding batch items (see FrontendScriptObject) and calling encoderHost.
↪runBatch() this method returns 'true' as long as a job is running.
  $.writeln("isBatchRunning: " + isBatchRunning);

  // API "getBatchEncoderStatus"
  var batchStatus = encoderHost.getBatchEncoderStatus();
  // expected value is "stopped", because the batch had not been started.
  // The values are: invalid, paused, running, stopped, stopping
  $.writeln("batch status is: " + batchStatus);

  // API "runBatch" (always returns true and therefore it's not necessary to store the␣
↪result)
  encoderHost.runBatch();

  // API "pauseBatch" (always returns true and therefore it's not necessary to store the␣
↪result)
  encoderHost.pauseBatch();

  // API "stopBatch" (always returns true and therefore it's not necessary to store the␣
↪result)
  encoderHost.stopBatch();

  // API "getCurrentBatchPreview"
```

(continues on next page)

(continued from previous page)

```
  var result = encoderHost.getCurrentBatchPreview(outputFile);
  $.writeln("result: " + result);

  // API "getSupportedImportFileTypes"
  var supportedFileTypes = encoderHost.getSupportedImportFileTypes();
  $.writeln("supportedFileTypes: " + supportedFileTypes);
} else {
  $.writeln("encoderHost script object not defined");
}
```

2.6 EncoderHostWrapperEvent

Provides the following event types for items in the batch queue: onItemEncodingStarted, onAudioPreEncodeProgress, onEncodingItemProgressUpdate, onItemEncodeComplete. For multiple batch items in the queue we recommend to use this event to ensure that the event types will be received for all batch items. It provides the following event type for the whole batch queue: onBatchEncoderStatusChanged.

2.6.1 Properties

- `audioInfo: string` : Returns the audio pre-encoding info for the event type onAudioPreEncodeProgress (available since 24.0).
- `audioProgress: float` : Returns the audio pre-encoding progress for the event type onAudioPreEncodeProgress (available since 24.0).
- `batchEncoderStatus: string` : Returns the status of the batch encoder, when the event was sent. Can be called for onBatchEncoderStatusChanged event, otherwise the status will be invalid. The values are: invalid, paused, running, stopped, stopping (available since 23.3).
- `onAudioPreEncodeProgress: constant string` : Notify when the audio pre-encode progress changes (available since 24.0).
- `onBatchEncoderStatusChanged: constant string` : Notify when the batch encoder status has changed. Get the new status from the batchEncoderStatus property. (available since 23.3)
- `onEncodingItemProgressUpdate: constant string` : Notify of the batch item encoding progress (available since 23.1).
- `onItemEncodeCompleted: constant string` : Notify when the batch item has been encoded.
- `onItemEncodingStarted: constant string` : Notify when the batch item encoding started (available since 23.1).
- `outputFilePath: string` : Returns the path of the output file. Can be called for onItemEncodingStarted and onItemEncodeComplete events.
- `progress: float` : Returns the encoding progress between 0 and 1. Can be called for onEncodingItemProgressUpdate event.
- `result: string` : Returns the encoding result 'True' or 'False'. Can be called for onItemEncodeComplete event.
- `sourceFilePath: string` : Returns the path of the source file. Can be called for onItemEncodingStarted and onItemEncodeComplete events.

2.6.2 Code Samples

```
// Please use this event when you have multiple batch items in the queue (added manually␣
↪or via a script as below)
// to ensure you receive all event types
var source_1 = "C:\\testdata\\testmedia1.mxf";
var source_2 = "C:\\testdata\\testmedia2.mxf";
var source_3 = "C:\\testdata\\testmedia3.mxf";

// //sources for mac
// var source_1 = "/Users/Shared/testdata/testmedia1.mxf"
// var source_2 = "/Users/Shared/testdata/testmedia2.mxf";
// var source_3 = "/Users/Shared/testdata/testmedia3.mxf";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("frontend.onItemAddedToBatch: success");
  });

  var batchItemSuccess_1 = frontend.addItemToBatch(source_1);
  var batchItemSuccess_2 = frontend.addItemToBatch(source_2);
  var batchItemSuccess_3 = frontend.addItemToBatch(source_3);
  if (batchItemSuccess_1 && batchItemSuccess_2 && batchItemSuccess_3) {
    $.writeln(
      "Batch item added successfully for the source files ",
      source_1 + " , " + source_2 + " , " + source_3
    );

    encoderHost = app.getEncoderHost();
    if (encoderHost) {
      // listen to the item encoding started event (available since 23.1.)
      encoderHost.addEventListener(
        "onItemEncodingStarted",
        function (eventObj) {
          $.writeln(
            "onItemEncodingStarted: Source File Path: " +
              eventObj.sourceFilePath
          );
          $.writeln(
            "onItemEncodingStarted: Output File Path: " +
              eventObj.outputFilePath
          );
        }
      );

      /* for earlier versions (23.0. or older) there's an additional step necessary to␣
↪listen to the onItemEncodingStarted event
        var exporter = app.getExporter();
        if (exporter) {
            exporter.addEventListener(
                "onItemEncodingStarted",
```

(continues on next page)

(continued from previous page)

```
                function (eventObj) {
                $.writeln("onItemEncodingStarted");
                }
            );
        }
    */

    // listen to the item encoding progress event (available since 23.1.)
    encoderHost.addEventListener(
      "onEncodingItemProgressUpdate",
      function (eventObj) {
        $.writeln(
          "onEncodingItemProgessUpdate: Encoding Progress: " +
            eventObj.progress
        );
      }
    );

    // listen to the audio pre-encoding progress event (available since 24.0.)
    encoderHost.addEventListener(
      "onAudioPreEncodeProgress",
      function (eventObj) {
        $.writeln("Audio pre-encoding info: " + eventObj.audioInfo);
        $.writeln("Audio pre-encoding progress: " + eventObj.audioProgress);
      },
      false
    );

    /* for earlier versions (23.0. or older) there's an additional step necessary to␣
↪listen to the onEncodingItemProgressUpdated event
       var exporter = app.getExporter();
       if (exporter) {
           exporter.addEventListener(
               "onEncodingItemProgressUpdated",
               function (eventObj) {
               $.writeln("onEncodingItemProgessUpdated: Encoding Progress: " + eventObj.
↪progress);
               }
           );
       }
    */

    // listen to the item encoding complete event
    encoderHost.addEventListener("onItemEncodeComplete", function (eventObj) {
      $.writeln("onItemEncodeComplete: Result: " + eventObj.result);
      $.writeln(
        "onItemEncodeComplete: Source File Path: " + eventObj.sourceFilePath
      );
      $.writeln(
        "onItemEncodeComplete: Output File Path: " + eventObj.outputFilePath
      );
    });
```

(continues on next page)

(continued from previous page)

```
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
  $.writeln("frontend not valid");
}
```

2.7 EncoderWrapper

Queue item object to set encode properties

2.7.1 Properties

- `outputFiles: array of strings` : Gets the list of files the encode generated
- `outputHeight: float` : Gets the height of the encoded output file
- `outputWidth: float` : Gets the width of the encoded output file

2.7.2 Methods

- `SetIncludeSourceXMP(includeSourceXMP: bool): bool` : Toggle the inclusion of source XMP [boolean] input value required
- `getEncodeProgress(): int` : Returns the encode progress as percentage
- `getEncodeTime(): float` : Return the encode time in milliseconds
- `getLogOutput(): string` : Returns the log output including possible warnings and errors (available since 23.2.).
- `getMissingAssets(includeSource: bool, includeOutput: bool): array of strings` : Returns a list of missing assets
  - `includeSource`: Get missing asset list from the source group if requested
- `getPresetList(): array of strings` : Returns the presets available for the assigned format
- `loadFormat(format: string): bool` : Changes the format for the batch item
  - `format`: E.g. 'H.264' Loads all presets available for the assigned format
- `loadPreset(presetPath: string): bool` : Loads and assigns the preset to the batch item
- `setCropOffsets(left: unsigned int, top: unsigned int, right: unsigned int, bottom: unsigned int): bool` : Sets the crop offsets
- `setCropState(cropState: bool): bool` : Sets the crop state [boolean] input value required
- `setCropType(cropType: unsigned int): bool` : Sets the scale type

  - `cropType`: 0 ScaleToFit, 1 ScaleToFitBeforeCrop, 2 SetAsOutputSize, 3 ScaleToFill, 4 ScaleToFillBeforeCrop, 5 StretchToFill, 6 StretchToFillBeforeCrop

- `setCuePointData(inCuePointsFilePath: string): bool` : Sets the cue point data
- `setFrameRate(framerate: string): bool` : Sets the frame rate for the batch item
  - `framerate`: E.g. '24' as string
- `setIncludeSourceCuePoints(includeSourceCuePoints: bool): bool` : Toggle the inclusion of cue points [boolean] input value required
- `setOutputFrameSize(width: unsigned int, height: unsigned int): bool` : Sets the output frame size
- `setRotation(rotationValue: float): bool` : Sets the rotation (in a 360 degree system)
  - `rotationValue`: E.g. 0.0 - 360.0
- `setScaleType(scaleType: unsigned int): bool` : Sets the scale type
  - `scaleType`: 0 ScaleToFit, 1 ScaleToFitBeforeCrop, 2 SetAsOutputSize, 3 ScaleToFill, 4 ScaleToFillBeforeCrop, 5 StretchToFill, 6 StretchToFillBeforeCrop
- `setTimeInterpolationType(interpolationType: unsigned int): bool` : Set the time interpolation type
  - `interpolationType`: 0 FrameSampling, 1 FrameBlending, 2 OpticalFlow
- `setUseFrameBlending(useFrameBlending: bool): bool` : Toggle the use of frame blending [boolean] input value required
- `setUseMaximumRenderQuality(useMaximumRenderQuality: bool): bool` : Toggle the use of maximum render quality [boolean] input value required
- `setUsePreviewFiles(usePreviewFiles: bool): bool` : Toggle the use of previews files. [boolean] input value required
- `setWorkArea(workAreaType: unsigned int, startTime: float, endTime: float): bool` : Sets the work area type, start and end time for the batch item
  - `workAreaType`: 0 Entire, 1 InToOut, 2 WorkArea, 3 Custom, 4 UseDefault
- `setWorkAreaInTicks(workAreaType: unsigned int, startTime: string, endTime: string): bool` : Sets the work area type, start and end time in ticks for the batch item (available since 23.3)
  - `workAreaType`: 0 Entire, 1 InToOut, 2 WorkArea, 3 Custom, 4 UseDefault
- `setXMPData(templateXMPFilePath: string): bool` : Sets XMP data to given template

2.7.3 Code Samples

```
var format = "";
var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HighQuality720HD.epr";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
```

(continues on next page)

(continued from previous page)

```
var frontend = app.getFrontend();
if (frontend) {
  // Either format or preset can be empty, output is optional
  var encoderWrapper = frontend.addFileToBatch(source, format, preset);

  if (encoderWrapper) {
    $.writeln(
      "Frontend script engine added the source file using addFileToBatch-",
      source,
      " successfully"
    );

    $.writeln("width :", encoderWrapper.outputWidth);
    $.writeln("height:", encoderWrapper.outputHeight);
    $.writeln("outputFiles:", encoderWrapper.outputFiles);

    //input value is string please use e.g. "25"
    encoderWrapper.setFrameRate("25");

    //int, 0-Entire, 1-InToOut, 2-WorkArea, 3-Custom, 4:UseDefault
    encoderWrapper.setWorkArea(2, 0.0, 1.0);

    var usePreviewFiles = true;
    encoderWrapper.setUsePreviewFiles(usePreviewFiles);

    var useMaximumRenderQuality = true;
    encoderWrapper.setUseMaximumRenderQuality(useMaximumRenderQuality);

    var useFrameBlending = true;
    encoderWrapper.setUseFrameBlending(useFrameBlending);

    // int-0-FrameSampling, 1-FrameBlending, 2-OpticalFlow
    encoderWrapper.setTimeInterpolationType(1);

    // be aware that this method first letter is upper case
    var includeSourceXMP = true;
    encoderWrapper.SetIncludeSourceXMP(includeSourceXMP);

    var includeSourceCuePoints = false;
    encoderWrapper.setIncludeSourceCuePoints(includeSourceCuePoints);

    var cropState = true;
    encoderWrapper.setCropState(cropState);

    //int, 0-ScaleToFit, 1-ScaleToFitBeforeCrop, 2-SetAsOutputSize, 3-ScaleToFill, 4-
↪ScaleToFillBeforeCrop, 5-StretchToFill, 6-StretchToFillBeforeCrop",
    encoderWrapper.setCropType(4);

    //int, 0-ScaleToFit, 1-ScaleToFitBeforeCrop, 2-SetAsOutputSize, 3-ScaleToFill, 4-
↪ScaleToFillBeforeCrop, 5-StretchToFill, 6-StretchToFillBeforeCrop",
    encoderWrapper.setScaleType(4);
```

(continues on next page)

(continued from previous page)

```
    // rotate clockwise, input values will be transformed into [0 - 360], so -90 is␣
↪equal to 270
    encoderWrapper.setRotation(180);

    //left, top, right, bottom
    encoderWrapper.setCropOffsets(10, 20, 10, 20);

    //width and height
    encoderWrapper.setOutputFrameSize(1200, 800);

    // default is off - deprecated
    //encoderWrapper.setCuePointData();

    var encoderHostWrapper = app.getEncoderHost();
    if (encoderHostWrapper) {
      encoderHostWrapper.runBatch();
    }
  } else {
    $.writeln("encoderWrapper is not valid");
  }
} else {
  $.writeln("frontend obj is not valid");
}
```

```
var format = "H.264";
var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HighQuality1080_HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality1080_HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var frontend = app.getFrontend();
if (frontend) {
  /**
   * getLogOutPut() returns a string in JSON format containing the possible errors and␣
↪warnings as well as the summary of the batch item
   * that is added to the queue.
   *
   * The getLogOutput() method is implemented in the EncoderWrapperScriptObject.
   * You can use getLogOutput() method when you have used one of these following methods:
   *
   * FrontEndScriptObject:
   * - addFileToBatch()
   * - addDLToBatch()
   * - addTeamProjectsToBatch()
   * - stitchFiles()
   * In Addition it is possible to get the batch item status with
   * encoderWrapper.addListener("onStatusChanged"){...} Here you will get "Done!",
↪"Failed!", "Stopped!"
```

(continues on next page)

(continued from previous page)

```
   *
   * ExportScriptObject:
   * - export()
   * - getSourceMediaInfo()
   * In Addition it is possible to get the batch item status with
   * exporter.addListener("OnBatchItemStatusChanged"){...} Here you will get integer␣
↪values see ExportScriptObject for the details
   *
   * EncoderHostWrapper:
   * - createEncoderFormat()
   *
   * Output format is
   *    {
   *        "time": "2023-01-16T12:18:36.617946",
   *        "error": "",
   *        "summary": []
   *    }
   */

  var encoderWrapper = frontend.addFileToBatch(
    source,
    format,
    preset,
    destination
  );
  if (encoderWrapper) {
    $.writeln("Batch item is successfully added to the queue: ", source);

    encoderWrapper.addEventListener("onEncodeFinished", function (eventObj) {
      // return the log output in JSON Format
      $.writeln(encoderWrapper.getLogOutput());
    });

    // get encoder host to run batch
    var encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.runBatch();
    } else {
      $.writeln("EncoderHostScriptObject is not valid");
    }
  } else {
    $.writeln(
      "EncoderWrapperScriptObject is not valid - batch item wasn't added successfully"
    );
  }
} else {
  $.writeln("FrontendScriptObject is not valid");
}
```

```
var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HighQuality720HD.epr";
```

(continues on next page)

(continued from previous page)

```
// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";

var format = "";
var frontend = app.getFrontend();
if (frontend) {
  var encoderWrapper = frontend.addFileToBatch(source, format, preset);

  if (encoderWrapper) {
    $.writeln(source, " has been added successfully");

    /**if you set the format parameter but no presetfilepath then you will
     * get all related presets to this specific format.
     *
     * If you set the presetfilepath but no format, then the
     * format will be set automatically that matches the current preset */

    var presetList = encoderWrapper.getPresetList();
    for (var index = 0; index < presetList.length; index++) {
      $.writeln(presetList[index]);
    }
  } else {
    $.writeln("encoderWrapper object is not valid");
  }
} else {
  $.writeln("Frontend object is not valid");
}
```

```
var format = "";
var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HighQuality720HD.epr";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";

var frontend = app.getFrontend();
if (frontend) {
  var encoderWrapper = frontend.addFileToBatch(source, format, preset);
  if (encoderWrapper) {
    encoderWrapper.loadFormat("MP3");
  } else {
    $.writeln("EncoderWrapper object is not valid");
  }
} else {
  $.writeln("Frontend object is not valid");
}
```

```
var format = "";
var source = "C:\\testdata\\testmedia4.mp4";
```

(continues on next page)

(continued from previous page)

```
var preset = "C:\\testdata\\HighQuality720HD.epr";

var differentPreset = "C:\\testdata\\High Quality 1080 HD.epr";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var differentPreset = "/Users/Shared/testdata/High Quality 1080 HD.epr";

var frontend = app.getFrontend();
if (frontend) {
  // Either format name or presetPath can be empty, output filepath is optional
  var encoderWrapper = frontend.addFileToBatch(source, format, preset);
  if (encoderWrapper) {
    encoderWrapper.loadPreset(differentPreset);
  } else {
    $.writeln("EncoderWrapper object is not valid");
  }
} else {
  $.writeln("Frontend object is not valid");
}
```

```
var format = "H.264";

var source = "C:\\testdata\\testmedia4.mp4";
var preset = "C:\\testdata\\HD 720p.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia4.mp4"
// var preset = "/Users/Shared/testdata/HD 720p.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

// The value of ticksPerSecond is predefined in premiere pro and ame.
// For more information please have a look into https://ppro-scripting.docsforadobe.dev/
↪other/time.html
var ticksPerSecond = 254016000000;
var startTimeInTicks = 20 * ticksPerSecond;
var timeToAddInTicks = 30 * ticksPerSecond;

var startTimeinTicksStr = String(startTimeInTicks);
var endTimeInTicksStr = String(timeToAddInTicks);

var frontend = app.getFrontend();
if (frontend) {
  var encoderWrapper = frontend.addFileToBatch(
    source,
    format,
    preset,
    destination
  );
  if (encoderWrapper) {
```

(continues on next page)

(continued from previous page)

```
    $.writeln("workarea start time: ", startTimeinTicksStr);
    $.writeln("workarea end time: ", endTimeInTicksStr);
    encoderWrapper.setWorkAreaInTicks(
      2,
      startTimeinTicksStr,
      endTimeInTicksStr
    );
  } else {
    $.writeln("encoderWrapper is not valid");
  }
  var encoderHost = app.getEncoderHost();
  if (encoderHost) {
    encoderHost.runBatch();
  } else {
    $.writeln("encoderHost is not valid");
  }
} else {
  $.writeln("frontend is not valid");
}
```

2.8 EncoderWrapperEvent

An event to inform of encode progress and completion.

2.8.1 Properties

- `audioInfo: string` : Returns the audio pre-encoding info for the event type onAudioPreEncodeProgress (available since 24.0).
- `audioProgress: string` : Returns the audio pre-encoding progress for the event type onAudioPreEncodeProgress (available since 24.0).
- `onAudioPreEncodeProgress: constant string` : Notify when the audio pre-encode progress changes (available since 24.0).
- `onEncodeFinished: constant string` : Notify when the batch item has been encoded.
- `onEncodeProgress: constant string` : Notify when the batch item encode progress changes.
- `result: string` : Returns the encoding result 'Done!', 'Failed!' or 'Stopped!' for the event type onEncodeFinished resp. the encoding progress for the event type onEncodeProgress which is between 0 and 100.

2.8.2 Code Samples

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();
if (exporter) {
  var encoderWrapper = exporter.exportItem(source, destination, preset);
  if (encoderWrapper) {
    encoderWrapper.addEventListener(
      "onEncodeFinished",
      function (eventObj) {
        $.writeln("Encoding result: " + eventObj.result);
      },
      false
    );
    encoderWrapper.addEventListener(
      "onEncodeProgress",
      function (eventObj) {
        $.writeln("Encoding progress: " + eventObj.result);
      },
      false
    );

    // listen to the audio pre-encoding progress event (available since 24.0.)
    encoderWrapper.addEventListener(
      "onAudioPreEncodeProgress",
      function (eventObj) {
        $.writeln("Audio pre-encoding info: " + eventObj.audioInfo);
        $.writeln("Audio pre-encoding progress: " + eventObj.audioProgress);
      },
      false
    );
  }
}
```

2.9 ExporterScriptObject

Contains several encoding methods. You can listen to different types of the AMEExportEvent: onEncodeComplete, onError, onMediaInfoCreated, onBatchItemStatusChanged, onItemEncodingStarted, onEncodingItemProgressUpdated, onAudioPreEncodeProgress, onPostProcessListInitialized

2.9.1 Properties

- `elapsedMilliseconds: float` : Returns the encode time in milliseconds.
- `encodeID: string` : Returns the current encode item ID as string.

2.9.2 Methods

- `exportGroup(sourcePath: string, outputPath: string, presetsPath: string, matchSource: bool = false): bool` : Export the source with the provided list of presets. Returns true in case of success.
  - `sourcePath`: Media path (Premiere Pro projects aren't supported).
  - `outputPath`: If outputPath is empty, then the output file location will be generated based on the source location.
  - `presetsPath`: Multiple preset paths can be provided separated via a | (e.g. 'path1|path2|path3'
  - `matchSource`: Optional. Default value is false
- `exportItem(sourcePath: string, outputPath: string, presetPath: string, matchSource: bool = false, writeFramesToDisk: bool = false): scripting object` : Export the source with the provided preset. Returns an EncoderWrapper object.
  - `sourcePath`: Media path or Premiere Pro project path (In case of a Premiere Pro project the last sequence will be used).
  - `outputPath`: If outputPath is empty, then the output file location will be generated based on the source location.
  - `matchSource`: Optional. Default value is false
  - `writeFramesToDisk`: Optional. Default value is false. True writes five frames at 0%, 25%, 50%, 75% and 100% of the full duration. Known issue: Currently it only works with parallel encoding disabled.
- `exportSequence(projectPath: string, outputPath: string, presetPath: string, matchSource: bool = false, writeFramesToDisk: bool = false, leadingFramesToTrim: int = 0, trailingFramesToTrim: int = 0, sequenceName: string = ""): bool` : Export the Premiere Pro sequence with the provided preset. Returns true in case of success.
  - `projectPath`: Premiere Pro project path.
  - `outputPath`: If outputPath is empty, then the output file location will be generated based on the source location.
  - `matchSource`: Optional. Default value is false.
  - `writeFramesToDisk`: Optional. Default value is false. True writes five frames at 0%, 25%, 50%, 75% and 100% of the full duration. Known issue: Currently it only works with parallel encoding disabled.
  - `leadingFramesToTrim`: Optional. Default value is 0.
  - `trailingFramesToTrim`: Optional. Default value is 0.

  - `sequenceName`: Optional. If sequence name is empty then we use the last sequence of the project.

- `getSourceMediaInfo(sourcePath: string): scripting object` : Returns a SourceMediaInfo object.
- `removeAllBatchItems(): bool` : Remove all batch items from the queue. Returns true in case of success.
- `trimExportForSR(sourcePath: string, outputPath: string, presetPath: string, matchSource: bool = false, writeFramesToDisk: bool = false, leadingFramesToTrim: int = 0, trailingFramesToTrim: int = 0): bool` : Smart render the source with the provided preset. Returns true in case of success.
  - `sourcePath`: Media path or Premiere Pro project path (In case of a Premiere Pro project the last sequence will be used).
  - `outputPath`: If outputPath is empty, then the output file location will be generated based on the source location.
  - `matchSource`: Optional. Default value is false.
  - `writeFramesToDisk`: Optional. Default value is false. True writes five frames at 0%, 25%, 50%, 75% and 100% of the full duration. Known issue: Currently it only works with parallel encoding disabled.
  - `leadingFramesToTrim`: Optional. Default value is 0.
  - `trailingFramesToTrim`: Optional. Default value is 0.

2.9.3 Code Samples

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();
if (exporter) {
  exporter.exportItem(source, destination, preset);
  exporter.addEventListener("onEncodeComplete", function (eventObj) {
    // We can get the encoding time from the event or from the exporter
    $.writeln(
      "Encode Complete Time (in milli seconds): " + eventObj.encodeCompleteTime
    );

    var encodeCompleteTimeMilliseconds = exporter.elapsedMilliseconds;
    $.writeln(
      "Encode Complete Time alt (in milli seconds): " +
        encodeCompleteTimeMilliseconds
    );
  });
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();
if (exporter) {
  var encoderWrapper = exporter.exportItem(source, destination, preset);
  var encodeID = exporter.encodeID;
  $.writeln("Encode ID: " + encodeID);
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset_1 = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var preset_2 = "C:\\testdata\\XDCAMHD 50 PAL 25p.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset_1 = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var preset_2 = "/Users/Shared/testdata/XDCAMHD 50 PAL 25p.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var matchSourceSettings = false; // optional
var presets = preset_1 + "|" + preset_2;

var exporter = app.getExporter();
if (exporter) {
  exporter.addEventListener(
    "onEncodeComplete",
    function (eventObj) {
      // We should arrive here two times (for every preset we have one batch item)
      $.writeln(
        "Encode Complete Status (always true): " + eventObj.encodeCompleteStatus
      );
      // We encode both batch items in parallel and so we don't really get the exact time␣
↪for each batch item
      // When we arrive here the second time we get the total encoding time for both␣
↪batch items (the first
      // could be ignored)
      $.writeln(
        "Encode Complete Time (in milliseconds): " + eventObj.encodeCompleteTime
      );
    },
    false
  );

  exporter.addEventListener(
```

(continues on next page)

(continued from previous page)

```
  "onError",
  function (eventObj) {
    $.writeln("Error while encoding");
  },
  false
);

exporter.addEventListener(
  "onBatchItemStatusChanged",
  function (eventObj) {
    $.writeln("Batch group index: " + eventObj.groupIndex);
    $.writeln("Batch item index: " + eventObj.itemIndex);
    /*
      Possible status values:
      kBatchItemStatus_Waiting = 0,
      kBatchItemStatus_Done,
      kBatchItemStatus_Failed,
      kBatchItemStatus_Skipped,
      kBatchItemStatus_Encoding,
      kBatchItemStatus_Paused,
      kBatchItemStatus_Stopped,
      kBatchItemStatus_Any,
      kBatchItemStatus_AutoStart,
      kBatchItemStatus_Done_Warning,
      kBatchItemStatus_WatchFolderWaiting
      */
    $.writeln("Batch item status: " + eventObj.status);
  },
  false
);

exporter.addEventListener(
  "onItemEncodingStarted",
  function (eventObj) {
    $.writeln("Encoding started for batch item.");
  },
  false
);

exporter.addEventListener(
  "onMediaInfoCreated",
  function (eventObj) {
    $.writeln("Media info created");
  },
  false
);

exporter.addEventListener(
  "onPostProcessListInitialized",
  function (eventObj) {
    $.writeln("Post process list has been initialized.");
  },
```

(continues on next page)

(continued from previous page)

```
    false
  );

  var encodingPreperationSuccess = exporter.exportGroup(
    source,
    destination,
    presets,
    matchSourceSettings
  );
  // Without all optional arguments:
  // var encodingPreperationSuccess = exporter.exportGroup(source, destination, presets);

  $.writeln(
    "Encoding preparations were successful: " + encodingPreperationSuccess
  );
}
```

```
// Supported: PR projects (last sequence will be used)
// var source = "C:\\testdata\\prProjectTest.prproj;
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\XDCAMHD 50 PAL 50i.epr";
var destination = "C:\\testdata\\outputFolder";
var matchSourceSettings = false; // optional
var writeFramesToDisk = false; // optional

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/XDCAMHD 50 PAL 50i.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var exporter = app.getExporter();

if (exporter) {
  // listen to events dispatched by the exporter:
  exporter.addEventListener(
    "onEncodeComplete",
    function (eventObj) {
      $.writeln(
        "Encode Complete Status (always true): " + eventObj.encodeCompleteStatus
      ); // Complete status always true
      $.writeln(
        "Encode Complete Time (in milliseconds): " + eventObj.encodeCompleteTime
      );
    },
    false
  );

  exporter.addEventListener(
    "onError",
    function (eventObj) {
      $.writeln("Error while encoding");
    },
```

(continues on next page)

(continued from previous page)

```
  false
);

exporter.addEventListener(
  "onBatchItemStatusChanged",
  function (eventObj) {
    $.writeln("Batch group index: " + eventObj.groupIndex);
    $.writeln("Batch item index: " + eventObj.itemIndex);
    //Possible status values:
    //kBatchItemStatus_Waiting = 0,
    //kBatchItemStatus_Done,
    //kBatchItemStatus_Failed,
    //kBatchItemStatus_Skipped,
    //kBatchItemStatus_Encoding,
    //kBatchItemStatus_Paused,
    //kBatchItemStatus_Stopped,
    //kBatchItemStatus_Any,
    //kBatchItemStatus_AutoStart,
    //kBatchItemStatus_Done_Warning,
    //kBatchItemStatus_WatchFolderWaiting
    $.writeln("Batch item status: " + eventObj.status);
  },
  false
);

exporter.addEventListener(
  "onEncodingItemProgressUpdated",
  function (eventObj) {
    $.writeln("Encoding progress for batch item: " + eventObj.progress);
  },
  false
);

// listen to the audio pre-encoding progress event (available since 24.0.)
exporter.addEventListener(
  "onAudioPreEncodeProgress",
  function (eventObj) {
    $.writeln("Audio pre-encoding info: " + eventObj.audioInfo);
    $.writeln("Audio pre-encoding progress: " + eventObj.audioProgress);
  },
  false
);

exporter.addEventListener(
  "onItemEncodingStarted",
  function (eventObj) {
    $.writeln("Encoding started for batch item.");
  },
  false
);

exporter.addEventListener(
```

(continues on next page)

(continued from previous page)

```
    "onMediaInfoCreated",
    function (eventObj) {
      $.writeln("Media info created");
    },
    false
  );

  exporter.addEventListener(
    "onPostProcessListInitialized",
    function (eventObj) {
      $.writeln("Post process list has been initialized.");
    },
    false
  );

  var encoderWrapper = exporter.exportItem(
    source,
    destination,
    preset,
    matchSourceSettings,
    writeFramesToDisk
  );
  // Without all optional arguments:
  // var encoderWrapper  = exporter.exportItem(source, destination, preset);

  if (encoderWrapper) {
    encoderWrapper.addEventListener(
      "onEncodeFinished",
      function (eventObj) {
        $.writeln("Encoding result: " + eventObj.result);
      },
      false
    );

    encoderWrapper.addEventListener(
      "onEncodeProgress",
      function (eventObj) {
        $.writeln("Encoding progress: " + eventObj.result);
      },
      false
    );
  }
}
```

```
var preset = "C:\\testdata\\XDCAMHD50PAL25p.epr";
var destination = "C:\\testdata\\Output";
var projectPath = "C:\\testdata\\prProjectTest.prproj";

// //sources for mac
// var preset = "/Users/Shared/testdata/XDCAMHD50PAL25p.epr";
// var destination = "/Users/Shared/testdata/Output";
// var projectPath = "/Users/Shared/testdata/prProjectTest.prproj";
```

(continues on next page)

(continued from previous page)

```
var matchSource = false;
var writeFramesToDisk = false;
var leadingFramesToTrim = 0;
var trailingFramesToTrim = 0;
var sequenceName = "AME-Test-Sequence";

var exporter = app.getExporter();

if (exporter) {
  var encodingPreperationSuccess = exporter.exportSequence(
    projectPath,
    destination,
    preset,
    matchSource,
    writeFramesToDisk,
    leadingFramesToTrim,
    trailingFramesToTrim,
    sequenceName
  );

  $.writeln(
    "Encoding preparations were successful: " + encodingPreperationSuccess
  );

  // please see 'exportGroup' how to register events
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"

var exporter = app.getExporter();
if (exporter) {
  var sourceMediaInfo = exporter.getSourceMediaInfo(source);
  if (sourceMediaInfo) {
    $.writeln("Success");
  }
}
```

```
// Preparation: Be sure there are some batch items in the queue. Otherwise create them␣
↪via scripting API's or directly in the UI
// since we need some batch item in the queue to verify the API removeAllBatchItems
var exporter = app.getExporter();
if (exporter) {
  var success = exporter.removeAllBatchItems();
  $.writeln("Remove all batch items was successful: " + success);
}
```

```
var source = "C:\\testdata\\testmedia.mp4";
```

(continues on next page)

(continued from previous page)

```
var preset = "C:\\testdata\\XDCAMHD50PAL25p.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia.mp4"
// var preset = "/Users/Shared/testdata/XDCAMHD50PAL25p.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var matchSource = false;
var writeFramesToDisk = false;
var leadingFramesToTrim = 10;
var trailingFramesToTrim = 700;

var exporter = app.getExporter();
if (exporter) {
  var encodingPreperationSuccess = exporter.trimExportForSR(
    source,
    destination,
    preset,
    matchSource,
    writeFramesToDisk,
    leadingFramesToTrim,
    trailingFramesToTrim
  );

  $.writeln(
    "Encoding preparations were successful: " + encodingPreperationSuccess
  );

  // please see 'exportGroup' how to register events
}
```

2.10 FrontendScriptObject

Scripting methods to the frontend

2.10.1 Methods

- `addCompToBatch(compPath: string, presetPath: string = "", outputPath: string = ""): bool` : Adds the first comp of an After Effects project resp. the first sequence of a Premiere Pro project to the batch.
  - `compPath`: Path to e.g. an After Effects project or Premiere Pro project. The first comp resp. sequence will be used.
  - `presetPath`: Optional. If presetPath is empty, then the default preset will be applied.
  - `outputPath`: Optional. If outputPath is empty, then the output file name will be generated based on the comp path.

- `addDLToBatch(projectPath: string, format: string, presetPath: string, guid: string, outputPath: string = ""): scripting object` : Adds e.g. an After Effects comp or Premiere Pro sequence to the batch and returns an EncoderWrapper object.
  - `projectPath`: E.g. Premiere Pro or After Effects project path.
  - `format`: E.g. 'H.264'
  - `presetPath`: Either a preset or a format input must be present. If no preset is used then the default preset of the specified format will be applied.
  - `guid`: The unique id of e.g. a Premiere Pro sequence or After Effects composition.
  - `outputPath`: Optional. If outputPath is empty, then the output file name will be generated based on the project path.
- `addFileSequenceToBatch(containingFolder: string, imagePath: string, presetPath: string, outputPath: string = ""): bool` : Adds an image sequence to the batch. The images will be sorted in alphabetical order.
  - `containingFolder`: The folder containing image files.
  - `imagePath`: All images from the containing folder with the same extension will be added to the output file.
  - `outputPath`: Optional. If outputPath is empty, then the output file name will be generated based on the containingFolder name
- `addFileToBatch(filePath: string, format: string, presetPath: string, outputPath: string = ""): scripting object` : Adds a file to the batch and returns an EncoderWrapper object.
  - `filePath`: File path of a media source.
  - `format`: E.g. 'H.264'
  - `presetPath`: Either a preset or a format input must be present. If no preset is used then the default preset of the specified format will be applied.
  - `outputPath`: Optional. If outputPath is empty, then the output file name will be generated based on the file path.
- `addItemToBatch(sourcePath: string): bool` : Adds a media source to the batch.
  - `sourcePath`: Path of the media source.
- `addTeamProjectsItemToBatch(projectsURL: string, format: string, presetPath: string, outputPath: string): scripting object` : Adds a team project item to the batch and returns an EncoderWrapper object.
  - `projectsURL`: Team Projects URL or Team Projects Snap. You can create a tp2snap file in PPro for a ProjectItem via the scripting API saveProjectSnapshot.
  - `format`: E.g. 'H.264'
  - `presetPath`: Either a preset or a format input must be present. If no preset is used then the default preset of the specified format will be applied.
- `addXMLToBatch(xmlPath: string, presetPath: string, outputFolderPath: string = ""): bool` : Adds Final Cut Pro xml to the batch.
  - `xmlPath`: Path to a Final Cut Pro xml file.
  - `outputFolderPath`: Optional. If outputFolderPath is empty, then the output file name will be generated based on the XML file path.
- `getDLItemsAtRoot(projectPath: string): array of strings` : Returns the list of GUIDs for objects (sequences/comps) at the top/root level.

  - projectPath: E.g. Premiere Pro or After Effects project path.

- stitchFiles(mediaPaths: string, format: string, presetPath: string, outputPath: string): scripting object : Adds a batch item for the given media and returns an EncoderWrapper object.
  - mediaPaths: Semicolon delimited list of media paths.
  - format: E.g. 'H.264'
  - presetPath: Either a preset or a format input must be present. If no preset is used then the default preset of the specified format will be applied.
- stopBatch(): bool : Stops the batch.

2.10.2 Code Samples

```
var projectPath = "C:\\testdata\\aeCompTest.aep";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/aeCompTest.aep"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("frontend.onItemAddedToBatch: success");
  });

  var batchItemSuccess = frontend.addCompToBatch(
    projectPath,
    preset,
    destination
  );
  if (batchItemSuccess) {
    $.writeln(
      "Frontend script engine added the source file ",
      projectPath,
      " successfully "
    );

    // get encoderHost to be able to listen for the item complete event
    encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.addEventListener("onItemEncodeComplete", function (eventObj) {
        $.writeln("Result: " + eventObj.result);
        $.writeln("Source File Path: " + eventObj.sourceFilePath);
        $.writeln("Output File Path: " + eventObj.outputFilePath);
      });
```

(continues on next page)

(continued from previous page)

```
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
  $.writeln("frontend not valid");
}
```

```
// The projectPath can be a path to an AfterEffects, Premiere Pro or Character Animator
↪project
var format = "H.264";
var projectPath = "C:\\testdata\\aeCompTest.aep";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var projectPath = "/Users/Shared/testdata/aeCompTest.aep"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var frontend = app.getFrontend();
if (frontend) {
  // first we need the guid of the e.g. ae comps or ppro sequences
  var result = frontend.getDLItemsAtRoot(projectPath);
  $.writeln(result.length + " comps / sequences found.");

  // import e.g. the first comp / sequence
  if (result.length > 0) {
    // listen for batch item added / creation failed event
    frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
      $.writeln("frontend.onItemAddedToBatch: success");
    });

    frontend.addEventListener("onBatchItemCreationFailed", function (eventObj) {
      $.writeln("frontend.onBatchItemCreationFailed: failed");
      $.writeln("srcFilePath: " + eventObj.srcFilePath);
      $.writeln("error: " + eventObj.error);
    });

    var encoderWrapper = frontend.addDLToBatch(
      projectPath,
      format,
      preset,
      result[0],
      destination
    );

    if (encoderWrapper) {
      $.writeln(
```

(continues on next page)

(continued from previous page)

```
        "Batch item added successfully for comp / sequence guid: ",
        result[0]
      );

      // listen for encode progress and encode finish events
      encoderWrapper.addEventListener("onEncodeProgress", function (eventObj) {
        $.writeln("Encoding progress for batch item: " + eventObj.result);
      });

      encoderWrapper.addEventListener("onEncodeFinished", function (eventObj) {
        $.writeln("Encoding result for batch item: " + eventObj.result);
      });

      // get encoder host to run batch
      var encoderHost = app.getEncoderHost();
      if (encoderHost) {
        encoderHost.runBatch();
      } else {
        $.writeln("encoderHost not valid");
      }
    } else {
      $.writeln("encoderWrapper not valid");
    }
  } else {
    $.writeln("the project doesn't have any comps / sequences");
  }
} else {
  $.writeln("frontend not valid");
}
```

```
var firstFile = "C:\\testdata\\Images\\AB-1.jpg";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";
var inContainingFolder = "C:\\testdata\\Images";

// //sources for mac
// var firstFile = "/Users/Shared/testdata/Images/AB-1.jpg"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";
// var inContainingFolder = "/Users/Shared/testdata/Images";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("onAddItemToBatch success");
  });

  var batchItemSuccess = frontend.addFileSequenceToBatch(
    inContainingFolder,
    firstFile,
    preset,
```

(continues on next page)

(continued from previous page)

```
    destination
  );

  if (batchItemSuccess) {
    $.writeln("Batch item added successfully");

    // get encoderHost to be able to listen for the item complete event
    var encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.addEventListener("onItemEncodeComplete", function (eventObj) {
        $.writeln("Result: " + eventObj.result);
        $.writeln("Source File Path: " + eventObj.sourceFilePath);
        $.writeln("Output File Path: " + eventObj.outputFilePath);
      });

      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
    $.writeln("frontend not valid");
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added / creation failed event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("frontend.onItemAddedToBatch: success");
  });

  frontend.addEventListener("onBatchItemCreationFailed", function (eventObj) {
    $.writeln("frontend.onBatchItemCreationFailed: failed");
    $.writeln("srcFilePath: " + eventObj.srcFilePath);
    $.writeln("error: " + eventObj.error);
  });

  var encoderWrapper = frontend.addFileToBatch(
    source,
    "H.264",
    preset,
```

(continues on next page)

(continued from previous page)

```
    destination
  );
  if (encoderWrapper) {
    $.writeln("Batch item added successfully for source file  ", source);

    // listen for encode progress and encode finish event
    encoderWrapper.addEventListener("onEncodeProgress", function (eventObj) {
      $.writeln("Encoding progress for batch item: " + eventObj.result);
    });

    encoderWrapper.addEventListener("onEncodeFinished", function (eventObj) {
      $.writeln("encoderWrapper.onEncodeFinished Success: " + eventObj.result);
    });

    // get encoder host to run batch
    var encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln(
      "encoderWrapper not valid - batch item wasn't added successfully"
    );
  }
} else {
  $.writeln("frontend not valid");
}
```

```
var source = "C:\\testdata\\testmedia3.mxf";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("frontend.onItemAddedToBatch: success");
  });

  var batchItemSuccess = frontend.addItemToBatch(source);
  if (batchItemSuccess) {
    $.writeln("Batch item added successfully for the source file ", source);

    // get encoderHost to be able to listen for the item complete event
    encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.addEventListener("onItemEncodeComplete", function (eventObj) {
        $.writeln("Result: " + eventObj.result);
        $.writeln("Source File Path: " + eventObj.sourceFilePath);
```

(continues on next page)

(continued from previous page)

```
        $.writeln("Output File Path: " + eventObj.outputFilePath);
      });

      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
  $.writeln("frontend not valid");
}
```

```
// use for the source (projectsURL) a valid Team Projects URL or a Team Projects Snap
// you can create a tp2snap file in PPro for a ProjectItem via the scripting API␣
↪saveProjectSnapshot
// e.g. projectItem.saveProjectSnapshot("C:\\testdata\\test.tp2snap");
var format = "H.264";
var teamsProjectPath = "C:\\testdata\\test.tp2snap";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var teamsProjectPath = "/Users/Shared/testdata/test.tp2snap"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added / creation failed event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("frontend.onItemAddedToBatch: success");
  });

  frontend.addEventListener("onBatchItemCreationFailed", function (eventObj) {
    $.writeln("frontend.onBatchItemCreationFailed: failed");
    $.writeln("srcFilePath: " + eventObj.srcFilePath);
    $.writeln("error: " + eventObj.error);
  });

  var encoderWrapper = frontend.addTeamProjectsItemToBatch(
    teamsProjectPath,
    format,
    preset,
    destination
  );

  if (encoderWrapper) {
    $.writeln(
      "Batch item added successfully for Team Projects url: ",
      teamsProjectPath
```

(continues on next page)

(continued from previous page)

```
    );

    // listen for encode progress and encode finish events
    encoderWrapper.addEventListener("onEncodeProgress", function (eventObj) {
      $.writeln("Encoding progress for batch item: " + eventObj.result);
    });

    encoderWrapper.addEventListener("onEncodeFinished", function (eventObj) {
      $.writeln("Encoding result for batch item: " + eventObj.result);
    });

    // get encoder host to run batch
    var encoderHost = app.getEncoderHost();

    if (encoderHost) {
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
  $.writeln("frontend not valid");
}
```

```
var source = "C:\\testdata\\FCP-3.xml"; // Final Cut Pro xml file
var preset = "C:\\testdata\\HighQuality720HD.epr";

// //sources for mac
// var source = "/Users/Shared/testdata/FCP-3.xml"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("onAddItemToBatch success");
  });

  var batchItemsuccess = frontend.addXMLToBatch(source, preset);

  if (batchItemsuccess) {
    $.writeln("Added xml file to batch successfully.");

    // get encoder host to listen for onItemEncodeComplete event and run batch
    encoderHost = app.getEncoderHost();
    if (encoderHost) {
      encoderHost.addEventListener("onItemEncodeComplete", function (eventObj) {
        $.writeln("Result: " + eventObj.result);
        $.writeln("Source File Path: " + eventObj.sourceFilePath);
        $.writeln("Output File Path: " + eventObj.outputFilePath);
```

(continues on next page)

(continued from previous page)

```
      });
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("batch item wasn't added successfully");
  }
} else {
  $.writeln("frontend not valid");
}
```

```
var projectPath = "C:\\testdata\\aeCompTest.aep"; // project path

// //sources for mac
// var projectPath = "/Users/Shared/testdata/aeCompTest.aep"

var frontend = app.getFrontend();
if (frontend) {
  var result = frontend.getDLItemsAtRoot(projectPath);

  $.writeln(result.length + " ae comps found.");
  for (var idx = 0; idx < result.length; ++idx) {
    $.writeln("GUID for item " + idx + " is " + result[idx] + ".");

    // These guids will be needed for e.g. the API frontend.addDLToBatch
  }
} else {
  $.writeln("frontend not valid");
}
```

```
var format = "H.264";
var media_1 = "C:\\testdata\\testmedia.mp4";
var media_2 = "C:\\testdata\\testmedia2.mp4.";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var media_1 = "/Users/Shared/testdata/testmedia.mp4"
// var media_2 = "/Users/Shared/testdata/testmedia2.avi"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var mediaPaths = media_1 + ";" + media_2;

var frontend = app.getFrontend();
if (frontend) {
  // listen for batch item added / creation failed event
  frontend.addEventListener("onItemAddedToBatch", function (eventObj) {
    $.writeln("onAddItemToBatch success");
  });
```

(continues on next page)

(continued from previous page)

```
  frontend.addEventListener("onBatchItemCreationFailed", function (eventObj) {
    $.writeln("onBatchItemCreationFailed");
  });

  var encoderWrapper = frontend.stitchFiles(
    mediaPaths,
    format,
    preset,
    destination
  );

  if (encoderWrapper) {
    $.writeln("Batch item added successfully");

    // listen for encode progress and encode finish events
    encoderWrapper.addEventListener("onEncodeProgress", function (eventObj) {
      $.writeln("Encoding progress for batch item: " + eventObj.result);
    });

    encoderWrapper.addEventListener("onEncodeFinished", function (eventObj) {
      $.writeln("Encoding result for batch item: " + eventObj.result);
    });

    // get encoder host to run batch
    var encoderHost = app.getEncoderHost();

    if (encoderHost) {
      encoderHost.runBatch();
    } else {
      $.writeln("encoderHost not valid");
    }
  } else {
    $.writeln("encoderWrapper not valid");
  }
} else {
  $.writeln("frontend not valid");
}
```

2.11 SourceMediaInfo

Get the width, height, PAR, duration, etc about the imported source

2.11.1 Properties

- `audioDuration:` `string` : Returns the audio duration of the source
- `description:` `string` : Returns embedded description of the source
- `dropFrameTimeCode:` `bool` : Returns true if the timecode is a drop frame timecode
- `duration:` `string` : Returns duration of the source
- `durationInTicks:` `None` : Returns duration of the source in ticks (available since 23.3)
- `fieldType:` `string` : Returns field type of the source
- `frameRate:` `string` : Returns frame rate of the source
- `height:` `string` : Returns height of the source
- `importer:` `string` : Returns the importer used to decode the source
- `numChannels:` `string` : Returns the number of audio channels of the source
- `parX:` `string` : Returns the X PAR of the source
- `parY:` `string` : Returns the Y PAR of the source
- `sampleRate:` `string` : Returns sample rate of the source
- `width:` `string` : Returns width of the source
- `xmp:` `string` : Returns xmp xml of the source

2.11.2 Code Samples

```
var source = "C:\\testdata\\testmedia3.mxf";

// //sources for mac
// var source = "/Users/Shared/testdata/testmedia3.mxf"

var exporter = app.getExporter();
if (exporter) {
  var sourceMediaInfo = exporter.getSourceMediaInfo(source);
  if (sourceMediaInfo) {
    var audioDuration = sourceMediaInfo.audioDuration;
    $.writeln("audio duration of the source: " + audioDuration);

    var description = sourceMediaInfo.description;
    $.writeln("description of the source: " + description);

    var isDropFrame = sourceMediaInfo.dropFrameTimeCode;
    $.writeln("is drop frame: " + dropFrameTimeCode);

    var duration = sourceMediaInfo.duration;
    $.writeln("duration of the source: " + duration);

    var fieldType = sourceMediaInfo.fieldType;
    $.writeln("field type of the source: " + fieldType);

    var frameRate = sourceMediaInfo.frameRate;
```

(continues on next page)

(continued from previous page)

```
    $.writeln("frame rate of the source: " + frameRate);

    var height = sourceMediaInfo.height;
    $.writeln("height of the source: " + height);

    var importer = sourceMediaInfo.importer;
    $.writeln("importer of the source: " + importer);

    var numChannels = sourceMediaInfo.numChannels;
    $.writeln("num channels of the source: " + numChannels);

    var parX = sourceMediaInfo.parX;
    $.writeln("par X of the source: " + parX);

    var parY = sourceMediaInfo.parY;
    $.writeln("par Y of the source: " + parY);

    var sampleRate = sourceMediaInfo.sampleRate;
    $.writeln("sample rate of the source: " + sampleRate);

    var width = sourceMediaInfo.width;
    $.writeln("width of the source: " + width);

    var xmp = sourceMediaInfo.xmp;
    $.writeln("xmp of the source: " + xmp);
  }
}
```

2.12 WatchFolderScriptEvent

An event to inform of batch item import completion

2.12.1 Properties

- `elapsedTime: float` : Returns the encoding elapsed time in milliseconds.
- `onEncodeComplete: constant string` : Notify when the watchfolder job item is complete
- `onEncodeError: constant string` : Notify when the watchfolder job encode fails

2.13 WatchFolderScriptObject

Scripting methods to watch folders

2.13.1 Methods

- `createWatchFolder(folderPath: string, outputPath: string, presetPath: string): bool` : Create a watch folder at destination for the preset and add the source
  - `folderPath`: The path to the folder which should be added as watch folder
- `removeAllWatchFolders(): bool` : Remove all watch folders

2.13.2 Code Samples

```
var folder = "C:\\testdata\\watchFolder";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputFolder";

// //sources for mac
// var folder = "/Users/Shared/testdata/watchFolder"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputFolder";

var watchFolder = app.getWatchFolder();
if (watchFolder) {
  var watchFolderSuccess = watchFolder.createWatchFolder(
    folder,
    destination,
    preset
  );

  if (watchFolderSuccess) {
    $.writeln(folder, " added as a watch folder");
    encoderHostWrapper = app.getEncoderHost();
    if (encoderHostWrapper) {
      watchFolder.addEventListener("onEncodeComplete", function (eventObj) {
        $.writeln("Elapsed Time: " + eventObj.elapsedTime);
        $.writeln("watchFolder.onEncodeComplete");
      });

      watchFolder.addEventListener("onEncodeError", function (eventObj) {
        $.writeln("watchFolder.onEncodeError");
      });

      encoderHostWrapper.runBatch();
    } else {
      $.writeln("EncoderHostWrapper object is not valid");
    }
  } else {
    $.writeln("Watch folder was not created");
  }
}
```

```
var folder = "C:\\testdata\\watchFolder";
var preset = "C:\\testdata\\HighQuality720HD.epr";
var destination = "C:\\testdata\\outputWatchfolder1";
```

(continues on next page)

(continued from previous page)

```
var folder2 = "C:\\testdata\\watchFolder2";
var destination2 = "C:\\testdata\\outputWatchfolder2";

// //sources for mac
// var folder = "/Users/Shared/testdata/watchFolder"
// var preset = "/Users/Shared/testdata/HighQuality720HD.epr";
// var destination = "/Users/Shared/testdata/outputWatchfolder1";
// var folder2 = "/Users/Shared/testdata/watchFolder2"
// var destination2 = "/Users/Shared/testdata/outputWatchfolder2";

var watchFolderObj = app.getWatchFolder();
if (watchFolderObj) {
  watchFolder.createWatchFolder(folder, destination, preset);
  watchFolder.createWatchFolder(folder2, destination2, preset);
  watchFolderObj.removeAllWatchFolders();
} else {
  $.writeln("Watch folder object is not valid");
}
```