DaVinci Resolve Scripting API - Documentation

Formatted and Maintained by X-Raym - Source - Release Article - Forum Thread - X-Raym Free Resolve Scripts - Donation ❤️

Important: Without any mention in the v19.1 changelog, **Blackmagic just removed UIManager for the free version of Resolve**, breaking all scripts using UI for free users, including the well-known Reactor. **Please join the discussion about this** on BM Forum to help bring it back!

Last Updated: 7 Oct 2025

In this package, you will find a brief introduction to the Scripting API for DaVinci Resolve Studio. Apart from this README.txt file, this package contains folders containing the basic import modules for scripting access (DaVinciResolve.py) and some representative examples.

From v16.2.0 onwards, the nodeIndex parameters accepted by `SetLUT()` and `SetCDL()` are 1-based instead of 0-based, i.e. 1 <= nodeIndex <= total number of nodes.

Overview

As with Blackmagic Fusion scripts, user scripts written in Lua and Python programming languages are supported. By default, scripts can be invoked from the Console window in the Fusion page, or via command line. This permission can be changed in Resolve Preferences, to be only from Console, or to be invoked from the local network. Please be aware of the security implications when allowing scripting access from outside of the Resolve application.

Prerequisites

DaVinci Resolve scripting requires one of the following to be installed (for all users):

```
Lua 5.1
Python >= 3.6 64-bit
Python 2.7 64-bit
```

Using a script

DaVinci Resolve needs to be running for a script to be invoked.

For a Resolve script to be executed from an external folder, the script needs to know of the API location. You may need to set the these environment variables to allow for your Python installation to pick up the appropriate dependencies as shown below:

Mac OS X:

```
RESOLVE_SCRIPT_API="/Library/Application Support/Blackmagic Design/DaVinci
RESOLVE_SCRIPT_LIB="/Applications/DaVinci Resolve/DaVinci Resolve.app/Cont
PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/"
```

Windows:

```
RESOLVE_SCRIPT_API="%PROGRAMDATA%\Blackmagic Design\DaVinci Resolve\Suppor
RESOLVE_SCRIPT_LIB="C:\Program Files\Blackmagic Design\DaVinci Resolve\fus
PYTHONPATH="%PYTHONPATH%;%RESOLVE_SCRIPT_API%\Modules\"
```

Linux:

```
RESOLVE_SCRIPT_API="/opt/resolve/Developer/Scripting"
RESOLVE_SCRIPT_LIB="/opt/resolve/libs/Fusion/fusionscript.so"
PYTHONPATH="$PYTHONPATH:$RESOLVE_SCRIPT_API/Modules/"
(Note: For standard ISO Linux installations, the path above may need to be
```

As with Fusion scripts, Resolve scripts can also be invoked via the menu and the Console.

On startup, DaVinci Resolve scans the subfolders in the directories shown below and enumerates the scripts found in the Workspace application menu under Scripts. Place your script under Utility to be listed in all pages, under Comp or Tool to be available in the Fusion page or under folders for individual pages (Edit, Color or Deliver). Scripts under Deliver are additionally listed under render jobs. Placing your script here and invoking it from the menu is the easiest way to use scripts.

Mac OS X:

- All users: `/Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts`
- Specific user: `/Users/<UserName>/Library/Application Support/Blackmagic Design/DaVinci Resolve/Fusion/Scripts`

Windows:

- All users: `%PROGRAMDATA%\Blackmagic Design\DaVinci Resolve\Fusion\Scripts`
- Specific user: `%APPDATA%\Roaming\Blackmagic Design\DaVinci Resolve\Support\Fusion\Scripts`

Linux:

- All users: `/opt/resolve/Fusion/Scripts (or /home/resolve/Fusion/Scripts/ depending on installation)`
- Specific user: `$HOME/.local/share/DaVinciResolve/Fusion/Scripts`

The interactive Console window allows for an easy way to execute simple scripting commands, to query or modify properties, and to test scripts. The console accepts commands in Python 2.7, Python 3.6 and Lua and evaluates and executes them immediately. For more information on how to use the Console, please refer to the DaVinci Resolve User Manual.

This example Python script creates a simple project:

```
#!/usr/bin/env python
import DaVinciResolveScript as dvr_script
resolve = dvr_script.scriptapp("Resolve")
fusion = resolve.Fusion()
```

```
projectManager = resolve.GetProjectManager()
projectManager.CreateProject("Hello World")
```

The resolve object is the fundamental starting point for scripting via Resolve. As a native object, it can be inspected for further scriptable properties - using table iteration and `getmetatable` in Lua and dir, help etc in Python (among other methods). A notable scriptable object above is fusion - it allows access to all existing Fusion scripting functionality.

Running DaVinci Resolve in headless mode

DaVinci Resolve can be launched in a headless mode without the user interface using the -nogui command line option. When DaVinci Resolve is launched using this option, the user interface is disabled. However, the various scripting APIs will continue to work as expected.

DaVinci Resolve API

Some commonly used API functions are described below (*). As with the resolve object, each object is inspectable for properties and functions.

Resolve

Name	Return	Definition
`Fusion()`	Fusion	Returns the Fusion obj Starting point for Fusio
`GetMediaStorage()`	MediaStorage	Returns the media stor object to query and act media locations.
`GetProjectManager()`	ProjectManager	Returns the project ma object for currently ope database.
`OpenPage(pageName)`	Bool	Switches to indicated p DaVinci Resolve. Input one of (`media` , `cut` , `fusion` , `color` , `fair` `deliver`).
`GetCurrentPage()`	String	Returns the page curre displayed in the main v Returned value can be `media` , `cut` , `edit` , `color` , `fairlight` , None).
`GetProductName()`	string	Returns product name.
`GetVersion()`	[version fields]	Returns list of product fields in [major, minor, build, suffix] format.
`GetVersionString()`	string	Returns product versio "major.minor.patch[suf format.
`LoadLayoutPreset(presetName)`	Bool	Loads UI layout from sa preset named `presetN`

Name	Return	Definition
`UpdateLayoutPreset(presetName)`	Bool	Overwrites preset nam `presetName` with curre layout.
`ExportLayoutPreset(presetName, presetFilePath)`	Bool	Exports preset named `presetName` to path `presetFilePath` .
`DeleteLayoutPreset(presetName)`	Bool	Deletes preset named `presetName` .
`SaveLayoutPreset(presetName)`	Bool	Saves current UI layout preset named `presetN`
`ImportLayoutPreset(presetFilePath, presetName)`	Bool	Imports preset from pa `presetFilePath` . The argument `presetName` how the preset shall be If not specified, the pre named based on the fil
`Quit()`	None	Quits the Resolve App.
`ImportRenderPreset(presetPath)`	Bool	Import a preset from p (string) and set it as cu preset for rendering.
`ExportRenderPreset(presetName, exportPath)`	Bool	Export a preset to a giv (string) if presetName(exists.
`ImportBurnInPreset(presetPath)`	Bool	Import a data burn in p from a given presetPat
`ExportBurnInPreset(presetName, exportPath)`	Bool	Export a data burn in p a given path (string) if presetName (string) ex
`GetKeyframeMode()`	keyframeMode	Returns the currently s keyframe mode (int). R section 'Keyframe Mod information' below for
`SetKeyframeMode(keyframeMode)`	Bool	Returns True when `keyframeMode` (enum) i successfully set. Refer t 'Keyframe Mode inform below for details.
`GetFairlightPresets()`	[presetNames...].	Returns a list of Fairligh by name

ProjectManager

Name	Return	Definition
`ArchiveProject(projectName, filePath, isArchiveSrcMedia=True, isArchiveRenderCache=True, isArchiveProxyMedia=False)`	Bool	Archives project to provided file path with the configuration as provided by the optional arguments
`CreateProject(projectName, mediaLocationPath)`	Project	Creates and returns a project if projectName (string) is unique, and None if it is not. Accepts an optional argument to set the media location path.
`DeleteProject(projectName)`	Bool	Delete project in the current folder if not currently loaded
`LoadProject(projectName)`	Project	Loads and returns the project with name = projectName (string) if there is a match found, and None if there is no matching Project.
`GetCurrentProject()`	Project	Returns the currently loaded Resolve project.
`SaveProject()`	Bool	Saves the currently loaded project with its own name. Returns True if successful.
`CloseProject(project)`	Bool	Closes the specified project without saving.
`CreateFolder(folderName)`	Bool	Creates a folder if folderName (string) is unique.
`DeleteFolder(folderName)`	Bool	Deletes the specified folder if it exists. Returns True in case of success.
`GetProjectListInCurrentFolder()`	[project names...]	Returns a list of project names in current folder.
`GetFolderListInCurrentFolder()`	[folder names...]	Returns a list of folder names in current folder.

Name	Return	Definition
`GotoRootFolder()`	Bool	Opens root folder in database.
`GotoParentFolder()`	Bool	Opens parent folder of current folder in database if current folder has parent.
`GetCurrentFolder()`	string	Returns the current folder name.
`OpenFolder(folderName)`	Bool	Opens folder under given name.
`ImportProject(filePath, projectName=None)`	Bool	Imports a project from the file path provided with given project name, if any. Returns True if successful.
`ExportProject(projectName, filePath, withStillsAndLUTs=True)`	Bool	Exports project to provided file path, including stills and LUTs if withStillsAndLUTs is True (enabled by default). Returns True in case of success.
`RestoreProject(filePath, projectName=None)`	Bool	Restores a project from the file path provided with given project name, if any. Returns True if successful.
`GetCurrentDatabase()`	{dbInfo}	Returns a dictionary (with keys `DbType`, `DbName` and optional `IpAddress`) corresponding to the current database connection
`GetDatabaseList()`	[{dbInfo}]	Returns a list of dictionary items (with keys `DbType`, `DbName` and optional `IpAddress`) corresponding to all the databases added to Resolve

Name	Return	Definition
`SetCurrentDatabase({dbInfo})`	Bool	Switches current database connection to the database specified by the keys below, and closes any open project. `DbType` : `Disk` or `PostgreSQL` (string) `DbName` : database name (string) `IpAddress` : IP address of the PostgreSQL server (string, optional key - defaults to `127.0.0.1`)
`CreateCloudProject({cloudSettings})`	Project	Creates and returns a cloud project. '{cloudSettings}': Check 'Cloud Projects Settings' subsection below for more information.
`LoadCloudProject({cloudSettings})`	Project	Loads and returns a cloud project with the following cloud settings if there is a match found, and None if there is no matching cloud project. '{cloudSettings}': Check 'Cloud Projects Settings' subsection below for more information.
`ImportCloudProject(filePath, {cloudSettings})`	Bool	Returns True if import cloud project is successful; False otherwise `filePath` : String; filePath of file to import '{cloudSettings}': Check 'Cloud Projects Settings' subsection below for more information.

Name	Return	Definition
`RestoreCloudProject(folderPath, {cloudSettings})`	Bool	Returns True if restore cloud project is successful; False otherwise `folderPath`: String; path of folder to restore '{cloudSettings}': Check 'Cloud Projects Settings' subsection below for more information.

Project

Name	Return	Definition
`GetMediaPool()`	MediaPool	Returns the object.
`GetTimelineCount()`	int	Returns the timelines c the project
`GetTimelineByIndex(idx)`	Timeline	Returns tim index, 1 <= project.Get
`GetCurrentTimeline()`	Timeline	Returns the timeline.
`SetCurrentTimeline(timeline)`	Bool	Sets given timeline fo Returns Tru
`GetGallery()`	Gallery	Returns the
`GetName()`	string	Returns pro
`SetName(projectName)`	Bool	Sets projec projectNar unique.
`GetPresetList()`	[presets...]	Returns a li their inforr
`SetPreset(presetName)`	Bool	Sets preset presetNam project.
`AddRenderJob()`	string	Adds a ren current ren render que unique job new rende
`DeleteRenderJob(jobId)`	Bool	Deletes rer job id (strir
`DeleteAllRenderJobs()`	Bool	Deletes all queue.

Name	Return	Definition
`GetRenderJobList()`	[render jobs...]	Returns a li and their ir
`GetRenderPresetList()`	[presets...]	Returns a li presets anc
`StartRendering(jobId1, jobId2, ...)`	Bool	Starts rend indicated b
`StartRendering([jobIds...], isInteractiveMode=False)`	Bool	Starts rend indicated b The option `isInterac` set, enable the UI duri
`StartRendering(isInteractiveMode=False)`	Bool	Starts rend render jobs The option `isInterac` set, enable the UI duri
`StopRendering()`	None	Stops any c processes.
`IsRenderingInProgress()`	Bool	Returns Tru progress.
`LoadRenderPreset(presetName)`	Bool	Sets a pres preset for r presetNam
`SaveAsNewRenderPreset(presetName)`	Bool	Creates nev given name presetNam
`DeleteRenderPreset(presetName)`	Bool	Delete renc provided n
`SetRenderSettings({settings})`	Bool	Sets given rendering. with suppc Refer to "L settings" se informatior settings
`GetRenderJobStatus(jobId)`	{status info}	Returns a c and compl of the job I (string).
`GetQuickExportRenderPresets()`	[preset_name..]	Returns a li render pres
`RenderWithQuickExport(preset_name, {param_dict}`	{status info}	Starts a qu for the curr timeline. pr GetQuickEx list. param_ render sett `TargetDir` `VideoQual`

Name	Return	Definition
		`EnableUpl` `EnableUpl` direct uplo web preset Returns a c and time ta an error str failed or nc Refer to "L Settings" s informatio supported
`GetSetting(settingName)`	string	Returns val setting (inc settingNan the section informatio
`SetSetting(settingName, settingValue)`	Bool	Sets the pr (indicated string) to tl (settingVal the section informatio
`GetRenderFormats()`	{render formats..}	Returns a c extension) formats.
`GetRenderCodecs(renderFormat)`	{render codecs...}	Returns a c descriptior of available render forr
`GetCurrentRenderFormatAndCodec()`	{format, codec}	Returns a c selected fo render cod
`SetCurrentRenderFormatAndCodec(format, codec)`	Bool	Sets given (string) anc (string) as rendering.
`GetCurrentRenderMode()`	int	Returns the Individual clip.
`SetCurrentRenderMode(renderMode)`	Bool	Sets the re Specify ren Individual clip.
`GetRenderResolutions(format, codec)`	[{Resolution}]	Returns list applicable render forr render cod full list of r argument i element in dictionary and `Heigh`

Name	Return	Definition
`RefreshLUTList()`	Bool	Refreshes L
`GetUniqueId()`	string	Returns a u project iter
`InsertAudioToCurrentTrackAtPlayhead(mediaPath, startOffsetInSamples, durationInSamples)`	Bool	Inserts the mediaPath startOffset durationInS playhead o on the Fair True if succ False.
`LoadBurnInPreset(presetName)`	Bool	Loads user in preset fo supplied pr Returns tru
`ExportCurrentFrameAsStill(filePath)`	Bool	Exports cur to supplied must end i format. Ret successful,
`GetColorGroupsList()`	[ColorGroups...]	Returns a li objects in t
`AddColorGroup(groupName)`	ColorGroup	Creates a n groupNam unique stri
`DeleteColorGroup(colorGroup)`	Bool	Deletes the and sets cli
`ApplyFairlightPresetToCurrentTimeline(name)`	Bool	Apply Fairli given name timeline, re successful,

MediaStorage

Name	Return	Definition
`GetMountedVolumeList()`	[paths...]	Returns list of folder paths corresponding to mounted volumes displayed in Resolve's Media Storage.
`GetSubFolderList(folderPath)`	[paths...]	Returns list of folder paths in the given absolute folder path.
`GetFileList(folderPath)`	[paths...]	Returns list of media and file listings in the given absolute

Name	Return	Definition
		folder path. Note that media listings may be logically consolidated entries.
`RevealInStorage(path)`	Bool	Expands and displays given file/folder path in Resolve's Media Storage.
`AddItemListToMediaPool(item1, item2, ...)`	[clips...]	Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is one or more file/folder paths. Returns a list of the MediaPoolItems created.
`AddItemListToMediaPool([items...])`	[clips...]	Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is an array of file/folder paths. Returns a list of the MediaPoolItems created.
`AddItemListToMediaPool([{itemInfo}, ...])`	[clips...]	Adds list of itemInfos specified as dict of `media`, `startFrame` (int), `endFrame` (int) from Media Storage into current Media Pool folder. Returns a list of the MediaPoolItems created.
`AddClipMattesToMediaPool(MediaPoolItem, [paths], stereoEye)`	Bool	Adds specified media files as mattes for the specified MediaPoolItem. StereoEye is an optional argument for specifying which

Name	Return	Definition
		eye to add the matte to for stereo clips (`left` or `right`). Returns True if successful.
`AddTimelineMattesToMediaPool([paths])`	[MediaPoolItems]	Adds specified media files as timeline mattes in current media pool folder. Returns a list of created MediaPoolItems.

MediaPool

Name	Return	Definition
`GetRootFolder()`	Folder	Returns root Folder c
`AddSubFolder(folder, name)`	Folder	Adds new subfolder Folder object with th
`RefreshFolders()`	Bool	Updates the folders i mode
`CreateEmptyTimeline(name)`	Timeline	Adds new timeline w
`AppendToTimeline(clip1, clip2, ...)`	[TimelineItem]	Appends specified M objects in the current the list of appended
`AppendToTimeline([clips])`	[TimelineItem]	Appends specified M objects in the current the list of appended
`AppendToTimeline([{clipInfo}, ...])`	[TimelineItem]	Appends list of clipIn dict of `mediaPoolIte` (float/int), `endFrame` (optional) `mediaType` only, 2 - Audio only), and `recordFrame` (fl the list of appended
`CreateTimelineFromClips(name, clip1, clip2,...)`	Timeline	Creates new timeline name, and appends t MediaPoolItem objec
`CreateTimelineFromClips(name, [clips])`	Timeline	Creates new timeline name, and appends t MediaPoolItem objec
`CreateTimelineFromClips(name, [{clipInfo}])`	Timeline	Creates new timeline name, appending the specified as a dict of `startFrame` (float/ir (float/int), `recordFra`
`ImportTimelineFromFile(filePath, {importOptions})`	Timeline	Creates timeline base within given file

Name	Return	Definition
		(AAF/EDL/XML/FCPX and optional import(support for the keys: `timelineName` : strin name of the timeline valid for DRT import `importSourceClips` : whether source clips imported, True by de DRT import `sourceClipsPath` : st filesystem path to se clips if the media is i original path and if is True `sourceClipsFolders` folder objects to sea if the media is not pr folder and if `importS` False. Not valid for D `interlaceProcessin` whether to enable in on the imported time valid only for AAF im
`DeleteTimelines([timeline])`	Bool	Deletes specified tim pool.
`GetCurrentFolder()`	Folder	Returns currently sele
`SetCurrentFolder(Folder)`	Bool	Sets current folder by
`DeleteClips([clips])`	Bool	Deletes specified clip mattes in the media
`ImportFolderFromFile(filePath, sourceClipsPath=``)`	Bool	Returns true if impor filePath is successful, sourceClipsPath is a s a filesystem path to s clips if the media is in original path, empty
`DeleteFolders([subfolders])`	Bool	Deletes specified sub media pool
`MoveClips([clips], targetFolder)`	Bool	Moves specified clips
`MoveFolders([folders], targetFolder)`	Bool	Moves specified fold
`GetClipMatteList(MediaPoolItem)`	[paths]	Get mattes for specif as a list of paths to th
`GetTimelineMatteList(Folder)`	[MediaPoolItems]	Get mattes in specifie MediaPoolItems.
`DeleteClipMattes(MediaPoolItem, [paths])`	Bool	Delete mattes based for specified MediaPo True on success.
`RelinkClips([MediaPoolItem], folderPath)`	Bool	Update the folder loc media pool clips with folder path.

Name	Return	Definition
`UnlinkClips([MediaPoolItem])`	Bool	Unlink specified med
`ImportMedia([items...])`	[MediaPoolItems]	Imports specified file current Media Pool fo array of file/folder pa of the MediaPoolIten
`ImportMedia([{clipInfo}])`	[MediaPoolItems]	Imports file path(s) ir Pool folder as specifi dict. Returns a list of MediaPoolItems crea Each clipInfo gets im MediaPoolItem unles Frames' is turned on. Example: ImportMed :"file_%03d.dpx", `Sta` `EndIndex` :100}]) wou "file_[001-100].dpx".
`ExportMetadata(fileName, [clips])`	Bool	Exports metadata of `fileName` in CSV for If no clips are specifie media pool will be us
`GetUniqueId()`	string	Returns a unique ID f
`CreateStereoClip(LeftMediaPoolItem, RightMediaPoolItem)`	MediaPoolItem	Takes in two existing and creates a new 3D media pool entry rep media in the media p
`AutoSyncAudio([MediaPoolItems], {audioSyncSettings})`	Bool	Syncs audio for speci [MediaPoolItems] (lis contain a minimum o MediaPoolItems - at one audio clip. Returns True if succes 'Audio Sync Settings'
`GetSelectedClips()`	[MediaPoolItems]	Returns the current s MediaPoolItems
`SetSelectedClip(MediaPoolItem)`	Bool	Sets the selected Me given MediaPoolItem

Folder

Name	Return	Definition
`GetClipList()`	[clips...]	Returns a list of clips (items) within the folder.
`GetName()`	string	Returns the media folder name.
`GetSubFolderList()`	[folders...]	Returns a list of subfolders in the folder.
`GetIsFolderStale()`	bool	Returns true if folder is stale in collaboration mode, false otherwise
`GetUniqueId()`	string	Returns a unique ID for the media pool folder

Name	Return	Definition
`Export(filePath)`	bool	Returns true if export of DRB folder to filePath is successful, false otherwise
`TranscribeAudio()`	Bool	Transcribes audio of the MediaPoolItems within the folder and nested folders. Returns True if successful; False otherwise
`ClearTranscription()`	Bool	Clears audio transcription of the MediaPoolItems within the folder and nested folders. Returns True if successful; False otherwise.

MediaPoolItem

Name	Return	Definition
`GetName()`	string	Returns the clip name.
`SetName(name)`	bool	Sets the clip's name to name(string). Returns True if successful
`GetMetadata(metadataType=None)`	string|dict	Returns the metadata value for the key `metadataType`. If no argument is specified, a dict of all set metadata properties is returned.
`SetMetadata(metadataType, metadataValue)`	Bool	Sets the given metadata to metadataValue (string). Returns True if successful.
`SetMetadata({metadata})`	Bool	Sets the item metadata with specified `metadata` dict. Returns True if successful.
`GetThirdPartyMetadata(metadataType=None)`	string|dict	Returns the third party metadata value for the key `metadataType`. If no argument is specified, a dict of all set third party metadata properties is returned.

Name	Return	Definition
`SetThirdPartyMetadata(metadataType, metadataValue)`	Bool	Sets/Add the given third party metadata to metadataValue (string). Returns True if successful.
`SetThirdPartyMetadata({metadata})`	Bool	Sets/Add the item third party metadata with specified `metadata` dict. Returns True if successful.
`GetMediaId()`	string	Returns the unique ID for the MediaPoolItem.
`AddMarker(frameId, color, name, note, duration, customData)`	Bool	Creates a new marker at given frameId position and with given marker information. `customData` is optional and helps to attach user specific data to the marker.
`GetMarkers()`	{markers...}	Returns a dict (frameId -> {information}) of all markers and dicts with their information. Example of output format: {96.0: { `color`: `Green`, `duration`: 1.0, `note`: `` , `name`: 'Marker 1', `customData`: `` }, ...} In the above example - there is one `Green` marker at offset 96 (position of the marker)
`GetMarkerByCustomData(customData)`	{markers...}	Returns marker {information} for the first matching marker with specifiec customData.
`UpdateMarkerCustomData(frameId, customData)`	Bool	Updates customData (string) for the marker at given frameId position. CustomData is not exposed via UI and is useful for scripting developer to attach

Name	Return	Definition
		any user specific data to markers.
`GetMarkerCustomData(frameId)`	string	Returns customData string for the marker at given frameId position.
`DeleteMarkersByColor(color)`	Bool	Delete all markers of the specified color from the media pool item. `All` as argument deletes all color markers.
`DeleteMarkerAtFrame(frameNum)`	Bool	Delete marker at frame number from the media pool item.
`DeleteMarkerByCustomData(customData)`	Bool	Delete first matching marker with specifiec customData.
`AddFlag(color)`	Bool	Adds a flag with given color (string).
`GetFlagList()`	[colors...]	Returns a list of flag colors assigned to the item.
`ClearFlags(color)`	Bool	Clears the flag of the given color if one exists. An `All` argument is supported and clears all flags.
`GetClipColor()`	string	Returns the item color as a string.
`SetClipColor(colorName)`	Bool	Sets the item color based on the colorName (string).
`ClearClipColor()`	Bool	Clears the item color.
`GetClipProperty(propertyName=None)`	string|dict	Returns the property value for the key `propertyName`. If no argument is specified, a dict of all clip properties is returned. Check the section below for more information.
`SetClipProperty(propertyName, propertyValue)`	Bool	Sets the given property to propertyValue (string). Check the section below for more information.
`LinkProxyMedia(proxyMediaFilePath)`	Bool	Links proxy media located at path

Name	Return	Definition
		specified by arg `proxyMediaFilePath` with the current clip. `proxyMediaFilePath` should be absolute clip path.
`LinkFullResolutionMedia(fullResMediaPath)`	Bool	Links proxy media to full resolution media files specified via its path.
`UnlinkProxyMedia()`	Bool	Unlinks any proxy media associated with clip.
`ReplaceClip(filePath)`	Bool	Replaces the underlying asset and metadata of MediaPoolItem with the specified absolute clip path.
`ReplaceClipPreserveSubClip(filePath)`	Bool	Replaces the underlying asset and metadata of a video or audio clip with the specified absolute clip path, preserving original sub clip extents.
`GetUniqueId()`	string	Returns a unique ID for the media pool item
`TranscribeAudio()`	Bool	Transcribes audio of the MediaPoolItem. Returns True if successful; False otherwise
`ClearTranscription()`	Bool	Clears audio transcription of the MediaPoolItem. Returns True if successful; False otherwise.
`GetAudioMapping()`	json formatted string	Returns a string with MediaPoolItem's audio mapping information. Check 'Audio Mapping' section below for more information.

Name	Return	Definition
`GetMarkInOut()`	{mark}	Returns dict of in/out marks set (keys omitted if not set), example: { `video` : { `in` : 0, `out` : 134}, `audio` : { `in` : 0, `out` : 134}}
`SetMarkInOut(in, out, type=` all `)`	Bool	Sets mark in/out of type `video` , `audio` or `all` (default).
`ClearMarkInOut(type=` all `)`	Bool	Clears mark in/out of type `video` , `audio` or `all` (default).
`MonitorGrowingFile()`	Bool	Monitor a file as long as it keeps growing (stops if the file does not grow for some time).

Timeline

Name	Return	Def
`GetName()`	string	Retu
`SetName(timelineName)`	Bool	Sets is ui
`GetStartFrame()`	int	Retu time
`GetEndFrame()`	int	Retu time
`SetStartTimecode(timecode)`	Bool	Set strir is su
`GetStartTimecode()`	string	Retu
`GetTrackCount(trackType)`	int	Retu type
`AddTrack(trackType, subTrackType)`	Bool	Adc `auc` usec sub lcr, l `5.1` `ada` sub trac
`AddTrack(trackType, newTrackOptions)`	Bool	Adc `auc` `auc` `inc`

Name	Return	Def
		Get `auc` trac `inc` app
`DeleteTrack(trackType, trackIndex)`	Bool	Dele `auc` <=
`GetTrackSubType(trackType, trackIndex)`	string	Retu the lcr, l `5.1` `ada` and `auc` retu
`SetTrackEnable(trackType, trackIndex, Bool)`	Bool	Ena trac trac 1 <
`GetIsTrackEnabled(trackType, trackIndex)`	Bool	Retu trac trac 1 <
`SetTrackLock(trackType, trackIndex, Bool)`	Bool	Locl trac trac 1 <
`GetIsTrackLocked(trackType, trackIndex)`	Bool	Retu trac trac 1 <
`DeleteClips([timelineItems], Bool)`	Bool	Dele time argu (The
`SetClipsLinked([timelineItems], Bool)`	Bool	Link dep
`GetItemListInTrack(trackType, index)`	[items...]	Retu (bas Get
`AddMarker(frameId, color, name, note, duration, customData)`	Bool	Crea and is o to t
`GetMarkers()`	{markers...}	Retu mar Exa `dur` `, `

Name	Return	Def
		indi 96
`GetMarkerByCustomData(customData)`	{markers...}	Retu mat
`UpdateMarkerCustomData(frameId, customData)`	Bool	Upc give expı dev mar
`GetMarkerCustomData(frameId)`	string	Retu give
`DeleteMarkersByColor(color)`	Bool	Dele colc dele
`DeleteMarkerAtFrame(frameNum)`	Bool	Dele num
`DeleteMarkerByCustomData(customData)`	Bool	Dele cust
`GetCurrentTimecode()`	string	Retu curr Colc
`SetCurrentTimecode(timecode)`	Bool	Sets time Deli
`GetCurrentVideoItem()`	item	Retu
`GetCurrentClipThumbnailImage()`	{thumbnailData}	Retu and ima bası Pag An e thur 6_gı Exar
`GetTrackName(trackType, trackIndex)`	string	Retu trac inde Get'
`SetTrackName(trackType, trackIndex, name)`	Bool	Sets by t inde Get'
`DuplicateTimeline(timelineName)`	timeline	Dup time succ
`CreateCompoundClip([timelineItems], {clipInfo})`	timelineItem	Crea with `00:` retu

Name	Return	Def
`CreateFusionClip([timelineItems])`	timelineItem	Cre retu
`ImportIntoTimeline(filePath, {importOptions})`	Bool	Imp opti with `aut` spe mec `igr` spe whe `lir` to s by c `use` info `imp` : Bo sho defa `ins` add defa `ins` offs "00: `ins` `sou` path inac `igr` `sou` fold mec
`Export(fileName, exportType, exportSubtype)`	Bool	Exp exp Refe prop
`GetSetting(settingName)`	string	Retu sett for r
`SetSetting(settingName, settingValue)`	Bool	Sets strir the
`InsertGeneratorIntoTimeline(generatorName)`	TimelineItem	Inse gen
`InsertFusionGeneratorIntoTimeline(generatorName)`	TimelineItem	Inse gen
`InsertFusionCompositionIntoTimeline()`	TimelineItem	Inse
`InsertOFXGeneratorIntoTimeline(generatorName)`	TimelineItem	Inse gen
`InsertTitleIntoTimeline(titleName)`	TimelineItem	Inse into

Name	Return	Def
`InsertFusionTitleIntoTimeline(titleName)`	TimelineItem	Inse strir
`GrabStill()`	galleryStill	Gral Gall
`GrabAllStills(stillFrameSource)`	[galleryStill]	Gral `sti` fran
`GetUniqueId()`	string	Retu
`CreateSubtitlesFromAudio({autoCaptionSettings})`	Bool	Crea Take {aut Sett info Retu
`DetectSceneCuts()`	Bool	Det time othe
`ConvertTimelineToStereo()`	Bool	Con succ
`GetNodeGraph()`	Graph	Retu
`AnalyzeDolbyVision([timelineItems]=[], analysisType=NONE)`	Bool	Ana time succ if [ti on a [tim set a for l
`GetMediaPoolItem()`	MediaPoolItem	Retu the
`GetMarkInOut()`	{mark}	Retu not { `vi` `out`
`SetMarkInOut(in, out, type=` all `)`	Bool	Sets (def
`ClearMarkInOut(type=` all `)`	Bool	Clea `all`
`GetVoiceIsolationState(trackIndex)`	{VoiceIsolationState}	Retu {isEr
`SetVoiceIsolationState(trackIndex, {VoiceIsolationState})`	Bool	Sets give Voic (int) trac True

TimelineItem

Name	Return	Definit
`GetName()`	string	Returns
`SetName(name)`	bool	Sets th Returns
`GetDuration(subframe_precision)`	int/float	Returns fractior subfrar
`GetEnd(subframe_precision)`	int/float	Returns the tim frames
`GetSourceEndFrame()`	int	Returns the me clip.
`GetSourceEndTime()`	float	Returns media
`GetFusionCompCount()`	int	Returns compo timelin
`GetFusionCompByIndex(compIndex)`	fusionComp	Returns object compIr timelin
`GetFusionCompNameList()`	[names...]	Returns names item.
`GetFusionCompByName(compName)`	fusionComp	Returns object
`GetLeftOffset(subframe_precision)`	int/float	Returns frame f fractior subfrar
`GetRightOffset(subframe_precision)`	int/float	Returns frame f Returns subfrar
`GetStart(subframe_precision)`	int/float	Returns the tim frames
`GetSourceStartFrame()`	int	Returns the me clip.
`GetSourceStartTime()`	float	Returns media
`SetProperty(propertyKey, propertyValue)`	Bool	Sets th `proper` `proper` Refer to propert
`GetProperty(propertyKey)`	int/[key:value]	returns if no ke

Name	Return	Definit
		returns table(lu
`AddMarker(frameId, color, name, note, duration, customData)`	Bool	Creates frameId marker optiona specific
`GetMarkers()`	{markers...}	Returns {inform dicts w Exampl `Green` `` , `nam `` `` `custo `` }, ...} inc at clip
`GetMarkerByCustomData(customData)`	{markers...}	Returns first ma custom
`UpdateMarkerCustomData(frameId, customData)`	Bool	Update marker Custom and is u to attac marker
`GetMarkerCustomData(frameId)`	string	Returns marker
`DeleteMarkersByColor(color)`	Bool	Delete color fr argume
`DeleteMarkerAtFrame(frameNum)`	Bool	Delete from th
`DeleteMarkerByCustomData(customData)`	Bool	Delete specifie
`AddFlag(color)`	Bool	Adds a
`GetFlagList()`	[colors...]	Returns to the i
`ClearFlags(color)`	Bool	Clear fl `All` ar all flags
`GetClipColor()`	string	Returns
`SetClipColor(colorName)`	Bool	Sets th colorNa
`ClearClipColor()`	Bool	Clears t
`AddFusionComp()`	fusionComp	Adds a associa
`ImportFusionComp(path)`	fusionComp	Imports given fi

Name	Return	Definit
		adding item.
`ExportFusionComp(path, compIndex)`	Bool	Exports based c provide
`DeleteFusionCompByName(compName)`	Bool	Deletes compo
`LoadFusionCompByName(compName)`	fusionComp	Loads t compo compo
`RenameFusionCompByName(oldName, newName)`	Bool	Renam identifi
`AddVersion(versionName, versionType)`	Bool	Adds a clip bas 1 - rem
`GetCurrentVersion()`	{versionName...}	Returns video c have th version
`DeleteVersionByName(versionName, versionType)`	Bool	Deletes version
`LoadVersionByName(versionName, versionType)`	Bool	Loads a active v 1 - rem
`RenameVersionByName(oldName, newName, versionType`	Bool	Renam by oldN local, 1
`GetVersionNameList(versionType)`	[names...]	Returns the give remote
`GetMediaPoolItem()`	MediaPoolItem	Returns corresp one exi
`GetStereoConvergenceValues()`	{keyframes...}	Returns keyfran conver
`GetStereoLeftFloatingWindowParams()`	{keyframes...}	For the (offset and res params include bottom
`GetStereoRightFloatingWindowParams()`	{keyframes...}	For the (offset and res params include bottom
`SetCDL([CDL map])`	Bool	Keys of `Slope` `Satura`

Name	Return	Definit
		NodeIr nodes. Exampl `NodeIr` 0.2", `0` "0.6 0.7
`AddTake(mediaPoolItem, startFrame, endFrame)`	Bool	Adds m Initializ timeline the full startFra are opt specify
`GetSelectedTakeIndex()`	int	Returns selecte take se
`GetTakesCount()`	int	Returns selecto selecto
`GetTakeByIndex(idx)`	{takeInfo...}	Returns `endFra` with tal
`DeleteTakeByIndex(idx)`	Bool	Deletes numbe
`SelectTakeByIndex(idx)`	Bool	Selects numbe
`FinalizeTake()`	Bool	Finalize
`CopyGrades([tgtTimelineItems])`	Bool	Copies grade t item in True if
`SetClipEnabled(Bool)`	Bool	Sets cli argume
`GetClipEnabled()`	Bool	Gets cli
`UpdateSidecar()`	Bool	Update or RMD
`GetUniqueId()`	string	Returns item
`LoadBurnInPreset(presetName)`	Bool	Loads u preset presetN success
`CreateMagicMask(mode)`	Bool	Returns createc mode c (backw
`RegenerateMagicMask()`	Bool	Returns regene otherw

Name	Return	Definit
`Stabilize()`	Bool	Returns success
`SmartReframe()`	Bool	Perforn True if
`GetNodeGraph(layerIdx)`	Graph	Returns at layer first lay layerId `nodeSt`
`GetColorGroup()`	ColorGroup	Returns exists.
`AssignToColorGroup(ColorGroup)`	Bool	Returns assigne ColorG group i
`RemoveFromColorGroup()`	Bool	Returns success ColorG
`ExportLUT(exportType, path)`	Bool	Exports value p for LUT notes' Saves g `path` the inte If an er provide (.cube/ end of
`GetLinkedItems()`	[TimelineItems]	Returns items.
`GetTrackTypeAndIndex()`	[trackType, trackIndex]	Returns corresp trackTy (int) res trackTy `subtit` 1 <= tr Timelin
`GetSourceAudioChannelMapping()`	json formatted string	Returns audio r 'Audio more ir
`GetIsColorOutputCacheEnabled()`	cache_value	Returns to cach
`GetIsFusionOutputCacheEnabled()`	cache_value	Returns to cach
`SetColorOutputCache(cache_value)`	Bool	Sets ca Equival action '

Name	Return	Definit
`SetFusionOutputCache(cache_value)`	Bool	Sets ca disable menu a Output
`GetVoiceIsolationState()`	{VoiceIsolationState}	Returns dict {isE timelin
`SetVoiceIsolationState({VoiceIsolationState})`	Bool	Sets Vo timelin VoiceIs (bool), range c success
`ResetAllNodeColors()`	Bool	Reset n active v True if

Gallery

Name	Return	Definition
`GetAlbumName(galleryStillAlbum)`	string	Returns the nam of the GalleryStillAlbur object `galleryStillAl` .
`SetAlbumName(galleryStillAlbum, albumName)`	Bool	Sets the name c the GalleryStillAlbur object `galleryStillAl` to `albumName` .
`GetCurrentStillAlbum()`	galleryStillAlbum	Returns current album as a GalleryStillAlbur object.
`SetCurrentStillAlbum(galleryStillAlbum)`	Bool	Sets current alb to GalleryStillAlb object `galleryStillAl` .
`GetGalleryStillAlbums()`	[galleryStillAlbum]	Returns the gall Still albums as a of GalleryStillAlb objects.
`GetGalleryPowerGradeAlbums()`	[galleryStillAlbum]	Returns the gall PowerGrade alb as a list of GalleryStillAlbur objects.
`CreateGalleryStillAlbum()`	galleryStillAlbum	Returns a newly created Still albu (GalleryStillAlbu

Name	Return	Definition
		object), or None not successful.
`CreateGalleryPowerGradeAlbum()`	galleryStillAlbum	Returns a newly created PowerG album (GalleryStillAlbu object), or None not successful.

GalleryStillAlbum

Name	Return	Definition
`GetStills()`	[galleryStill]	Returns the list of GalleryStill objects in the album.
`GetLabel(galleryStill)`	string	Returns the label of the galleryStill.
`SetLabel(galleryStill, label)`	Bool	Sets the new `label` to GalleryStill object `galleryStill`.
`ImportStills([filePaths])`	Bool	Imports GalleryStill from each filePath in [filePaths] list. True if at least one still is imported successfully. False otherwise.
`ExportStills([galleryStill], folderPath, filePrefix, format)`	Bool	Exports list of GalleryStill objects '[galleryStill]' to directory `folderPath`, with filename prefix `filePrefix`, using file format `format` (supported formats: dpx, cin, tif, jpg, png, ppm, bmp, xpm, drx).
`DeleteStills([galleryStill])`	Bool	Deletes specified list of GalleryStill objects '[galleryStill]'.

GalleryStill

This class does not provide any API functions but the object type is used by functions in other classes.

Graph

Name	Return	Definition
`GetNumNodes()`	int	Returns the number of nodes in the graph
`SetLUT(nodeIndex, lutPath)`	Bool	Sets LUT on the node mapping the node index

Name	Return	Definition
		provided, 1 <= nodeIndex <= self.GetNumNodes(). The lutPath can be an absolute path, or a relative path (based off custom LUT paths or the master LUT path). The operation is successful for valid lut paths that Resolve has already discovered (see Project.RefreshLUTList).
`GetLUT(nodeIndex)`	String	Gets relative LUT path based on the node index provided, 1 <= nodeIndex <= total number of nodes.
`SetNodeCacheMode(nodeIndex, cache_value)`	Bool	Sets the cache mode type on the node mapping the node index provided. Refer to "Cache Mode" section below to find the possible values of cache_value.
`GetNodeCacheMode(nodeIndex)`	cache_value	Returns the cache mode type on the node mapping the node index provided.
`GetNodeLabel(nodeIndex)`	string	Returns the label of the node at nodeIndex.
`GetToolsInNode(nodeIndex)`	[toolsList]	Returns toolsList (list of strings) of the tools used in the node indicated by given nodeIndex (int).
`SetNodeEnabled(nodeIndex, isEnabled)`	Bool	Sets the node at the given nodeIndex (int) to isEnabled (bool). 1 <= nodeIndex <= self.GetNumNodes().
`ApplyGradeFromDRX(path, gradeMode)`	Bool	Loads a still from given file path (string) and applies grade to graph with gradeMode (int): 0 - “No keyframes”, 1 - “Source Timecode aligned”, 2 - “Start Frames aligned”.
`ApplyArriCdlLut()`	Bool	Applies ARRI CDL and LUT. Returns True if successful, False otherwise.
`ResetAllGrades()`	Bool	Returns True if all grades were reset successfully, False otherwise.

ColorGroup

Name	Return	Definition
`GetName()`	String	Returns the name (string) of the ColorGroup.
`SetName(groupName)`	Bool	Renames ColorGroup to groupName (string).
`GetClipsInTimeline(Timeline=CurrTimeline)`	[TimelineItem]	Returns a list of TimelineItem that are in colorGroup in the given Timeline. Timeline is Current Timeline by default.
`GetPreClipNodeGraph()`	Graph	Returns the ColorGroup Pre-clip graph.
`GetPostClipNodeGraph()`	Graph	Returns the ColorGroup Post-clip graph.

List and Dict Data Structures

Beside primitive data types, Resolve's Python API mainly uses list and dict data structures. Lists are denoted by [...] and dicts are denoted by { ... } above. As Lua does not support list and dict data structures, the Lua API implements `list` as a table with indices, e.g. { [1] = listValue1, [2] = listValue2, ... }. Similarly the Lua API implements `dict` as a table with the dictionary key as first element, e.g. { [dictKey1] = dictValue1, [dictKey2] = dictValue2, ... }.

Keyframe Mode information

This section covers additional notes for the functions `Resolve.GetKeyframeMode()` and Resolve.SetKeyframeMode(keyframeMode).

`keyframeMode` can be one of the following enums:

```
- `resolve.KEYFRAME_MODE_ALL`     == 0
- `resolve.KEYFRAME_MODE_COLOR`   == 1
- `resolve.KEYFRAME_MODE_SIZING`  == 2
```

Integer values returned by `Resolve.GetKeyframeMode()` will correspond to the enums above.

Cache Mode information

This section covers additional notes for the functions Graph:GetNodeCacheMode(nodeIndex) and Graph:SetNodeCacheMode(nodeIndex, cache_value).

cache_value is an enumerated integer with one of the following values:

```
- `resolve.CACHE_AUTO_ENABLED`  = -1
- `resolve.CACHE_DISABLED`      =  0
- `resolve.CACHE_ENABLED`       =  1
```

Integer values returned by Graph:GetNodeCacheMode(nodeIndex) will correspond to the enums above.

Cloud Projects Settings

This section covers additional notes for the functions `ProjectManager:LoadCloudProject`, `ProjectManager:CreateCloudProject`, `ProjectManager:ImportCloudProject`, and `ProjectManager:RestoreCloudProject`

All four functions `ProjectManager:CreateCloudProject`, `ProjectManager:LoadCloudProject`, `ProjectManager:ImportCloudProject`, and `ProjectManager:RestoreCloudProject` take in a {cloudSettings} dict, that have the following keys:

- `resolve.CLOUD_SETTING_PROJECT_NAME`: String, [`` by default]
- `resolve.CLOUD_SETTING_PROJECT_MEDIA_PATH`: String, [`` by default]
- `resolve.CLOUD_SETTING_IS_COLLAB`: Bool, [False by default]
- `resolve.CLOUD_SETTING_SYNC_MODE`: syncMode (see below), [`resolve.CLOUD_SYNC_PROXY_ONLY` by default]
- `resolve.CLOUD_SETTING_IS_CAMERA_ACCESS`: Bool [False by default]

Note that `ProjectManager:LoadCloudProject` only honour the following keys: `resolve.CLOUD_SETTING_PROJECT_NAME`, `resolve.CLOUD_SETTING_PROJECT_MEDIA_PATH` and `resolve.CLOUD_SETTING_SYNC_MODE`. Only 1st load on a given system will honour all 3 settings. Subsequent loads will honour only `resolve.CLOUD_SETTING_PROJECT_NAME`

Where syncMode is one of the following values:

- `resolve.CLOUD_SYNC_NONE`,
- `resolve.CLOUD_SYNC_PROXY_ONLY`,
- `resolve.CLOUD_SYNC_PROXY_AND_ORIG`

All four functions `ProjectManager:CreateCloudProject`, `ProjectManager:LoadCloudProject`, `ProjectManager:ImportCloudProject`, and `ProjectManager:RestoreCloudProject` require `resolve.PROJECT_MEDIA_PATH` to be defined. `ProjectManager:LoadCloudProject` and `ProjectManager:CreateCloudProject` also requires `resolve.PROJECT_NAME` to be defined.

Audio Sync Settings

This section covers additional notes for the functions `MediaPool:AutoSyncAudio`.

AutoSyncAudio takes in a {audioSyncSettings} dict, that has the following keys:

- `resolve.AUDIO_SYNC_MODE` : audioSyncMode (see below), [`resolve.AUDIO_SYNC_TIMECODE` by default]
- `resolve.AUDIO_SYNC_CHANNEL_NUMBER` : channelNumber (see below) [1 by default]
- `resolve.AUDIO_SYNC_RETAIN_EMBEDDED_AUDIO` : Bool, [False by default]
- `resolve.AUDIO_SYNC_RETAIN_VIDEO_METADATA` : Bool, [False by default]

audioSyncMode can be one of the following:

- `resolve.AUDIO_SYNC_WAVEFORM`
- `resolve.AUDIO_SYNC_TIMECODE`

With AUDIO_SYNC_WAVEFORM mode, channelNumber is used to determine channel offset for comparison. channelNumber can be one of the following:

- `resolve.AUDIO_SYNC_CHANNEL_AUTOMATIC` = -1
- `resolve.AUDIO_SYNC_CHANNEL_MIX` = -2
- an actual channel offset from input media for waveform comparison. 1 <= channel offset <= channelMax, where channelMax is the channel count of the audio clip in [MediaPoolItems] with the fewest channels.

Looking up Project and Clip properties

This section covers additional notes for the functions `Project:GetSetting` , `Project:SetSetting` , `Timeline:GetSetting` , `Timeline:SetSetting` , `MediaPoolItem:GetClipProperty` and `MediaPoolItem:SetClipProperty` . These functions are used to get and set properties otherwise available to the user through the Project Settings and the Clip Attributes dialogs.

The functions follow a key-value pair format, where each property is identified by a key (the settingName or propertyName parameter) and possesses a value (typically a text value). Keys and values are designed to be easily correlated with parameter names and values in the Resolve UI. Explicitly enumerated values for some parameters are listed below.

Some properties may be read only - these include intrinsic clip properties like date created or sample rate, and properties that can be disabled in specific application contexts (e.g. custom colorspaces in an ACES workflow, or output sizing parameters when behavior is set to match timeline)

Getting values:

Invoke `Project:GetSetting` , `Timeline:GetSetting` or `MediaPoolItem:GetClipProperty` with the appropriate property key. To get a snapshot of all queryable properties (keys and values), you can call `Project:GetSetting` , `Timeline:GetSetting` or `MediaPoolItem:GetClipProperty` without parameters (or with a NoneType or a blank property key). Using specific keys to query individual properties will be faster. Note that getting a property using an invalid key will return a trivial result.

Setting values:

Invoke `Project:SetSetting` , `Timeline:SetSetting` or `MediaPoolItem:SetClipProperty` with the appropriate property key and a valid value. When setting a parameter, please check the return value to ensure the success of the operation. You can troubleshoot the validity of keys and values by setting the desired result from the UI and checking property snapshots before and after the change.

The following Project properties have specifically enumerated values:

`superScale` - the property value is an enumerated integer between 0 and 4 with these meanings: 0=Auto, 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. for super scale multiplier '2x Enhanced', exactly 4 arguments must be passed as outlined below. If less than 4 arguments are passed, it will default to 2x. Affects:

• x = Project:GetSetting(`superScale`) and Project:SetSetting(`superScale` , x) • for '2x Enhanced' --> Project:SetSetting(`superScale` , 2, sharpnessValue, noiseReductionValue), where sharpnessValue is a float in the range [0.0, 1.0] and noiseReductionValue is a float in the range [0.0, 1.0]

`timelineFrameRate` - the property value is one of the frame rates available to the user in project settings under "Timeline frame rate" option. Drop Frame can be configured for supported frame rates by appending the frame rate with "DF", e.g. "29.97 DF" will enable drop frame and "29.97" will disable drop frame Affects:

• x = Project:GetSetting(`timelineFrameRate`) and Project:SetSetting(`timelineFrameRate` , x)

The following Clip properties have specifically enumerated values:

"Super Scale" - the property value is an enumerated integer between 1 and 4 with these meanings: 1=no scaling, and 2, 3 and 4 represent the Super Scale multipliers 2x, 3x and 4x. for super scale multiplier '2x Enhanced', exactly 4 arguments must be passed as outlined below. If less than 4 arguments are passed, it will default to 2x. Affects:

• x = MediaPoolItem:GetClipProperty('Super Scale') and MediaPoolItem:SetClipProperty('Super Scale', x) • for '2x Enhanced' --> MediaPoolItem:SetClipProperty('Super Scale', 2, sharpnessValue, noiseReductionValue), where sharpnessValue is a float in the range [0.0, 1.0] and noiseReductionValue is a float in the range [0.0, 1.0]

"Cloud Sync" = the property value is an enumerated integer that will correspond to one of the following enums:

- `resolve.CLOUD_SYNC_DEFAULT` == -1
- `resolve.CLOUD_SYNC_DOWNLOAD_IN_QUEUE` == 0
- `resolve.CLOUD_SYNC_DOWNLOAD_IN_PROGRESS` == 1
- `resolve.CLOUD_SYNC_DOWNLOAD_SUCCESS` == 2
- `resolve.CLOUD_SYNC_DOWNLOAD_FAIL` == 3
- `resolve.CLOUD_SYNC_DOWNLOAD_NOT_FOUND` == 4
- `resolve.CLOUD_SYNC_UPLOAD_IN_QUEUE` == 5
- `resolve.CLOUD_SYNC_UPLOAD_IN_PROGRESS` == 6
- `resolve.CLOUD_SYNC_UPLOAD_SUCCESS` == 7
- `resolve.CLOUD_SYNC_UPLOAD_FAIL` == 8
- `resolve.CLOUD_SYNC_UPLOAD_NOT_FOUND` == 9
- `resolve.CLOUD_SYNC_SUCCESS` == 10

Audio Mapping

This section covers the output for `mpItem.GetAudioMapping()` and `timelineItem.GetSourceAudioChannelMapping()` Mapping format (json result) is similar for mpItem and timelineItem.

This section will follow an example of an mpItem that has audio from its embedded source, and from two other clips that are linked to it. The audio clip attributes of this mpItem will show 3 tracks.

Assume that (A) the embedded track is of format/type `stereo` (2 channels), (B) linked clip 1 track is of format/type '7.1' (8 channels), (C) linked clip 2 track is '5.1'

(6 channels) and assume that the format/type was not changed further.

`mpItem.GetAudioMapping()` returns a string of the form:

```
{
  "embedded_audio_channels": 2,                 # Total number of embedded
  "linked_audio": {                             # A list of only linked au
    "1": {                                      # Same as (B) above
      "channels": 8,
      "offset": -100,                           # Audio at media offset 0
      "path": FILE_PATH
    },
    "2": {                                      # Same as (C) above
      "channels": 6,
      "offset": 200,                            # Audio at media start pla
      "path": FILE_PATH
    }
  },
  "track_mapping": {                            # Listing of all the track
    "1": {
      "channel_idx": [1, 3],                    # In this case, channel in
      "mute": true,                             # Mute 'true' indicates tr
      "type": "Stereo"                          # The length of the 'chann
                                                # In this case, 'Stereo' a
    },
    "2": {
      "channel_idx": [3, 4, 5, 6, 7, 8, 9, 10], # Channel indices here are
      "mute": true,
      "type": "7.1"
    },
    "3": {
      "channel_idx": [1, 1, 1, 1, 15, 16],      # The first four channels
      "mute": false,
      "type": "5.1"
    }
  }
}
```

Auto Caption Settings

This section covers the supported settings for the method Timeline.CreateSubtitlesFromAudio({autoCaptionSettings})

The parameter setting is a dictionary containing the following keys:

- `resolve.SUBTITLE_LANGUAGE` : languageID (see below), [`resolve.AUTO_CAPTION_AUTO` by default]
- `resolve.SUBTITLE_CAPTION_PRESET` : presetType (see below), [`resolve.AUTO_CAPTION_SUBTITLE_DEFAULT` by default]
- `resolve.SUBTITLE_CHARS_PER_LINE` : Number between 1 and 60 inclusive [42 by default]
- `resolve.SUBTITLE_LINE_BREAK` : lineBreakType (see below), [`resolve.AUTO_CAPTION_LINE_SINGLE` by default]
- `resolve.SUBTITLE_GAP` : Number between 0 and 10 inclusive [0 by default]

Note that the default values for some keys may change based on values defined for other keys, as per the UI. For example, if the following dictionary is supplied,

```
{ resolve.SUBTITLE_LANGUAGE = resolve.AUTO_CAPTION_KOREAN,
  resolve.SUBTITLE_CAPTION_PRESET = resolve.AUTO_CAPTION_NETFLIX }
```

the default value for `resolve.SUBTITLE_CHARS_PER_LINE` will be 16 instead of 42

languageIDs:

- `resolve.AUTO_CAPTION_AUTO`
- `resolve.AUTO_CAPTION_DANISH`
- `resolve.AUTO_CAPTION_DUTCH`
- `resolve.AUTO_CAPTION_ENGLISH`
- `resolve.AUTO_CAPTION_FRENCH`
- `resolve.AUTO_CAPTION_GERMAN`
- `resolve.AUTO_CAPTION_ITALIAN`
- `resolve.AUTO_CAPTION_JAPANESE`
- `resolve.AUTO_CAPTION_KOREAN`
- `resolve.AUTO_CAPTION_MANDARIN_SIMPLIFIED`
- `resolve.AUTO_CAPTION_MANDARIN_TRADITIONAL`
- `resolve.AUTO_CAPTION_NORWEGIAN`
- `resolve.AUTO_CAPTION_PORTUGUESE`
- `resolve.AUTO_CAPTION_RUSSIAN`
- `resolve.AUTO_CAPTION_SPANISH`
- `resolve.AUTO_CAPTION_SWEDISH`

presetTypes:

- `resolve.AUTO_CAPTION_SUBTITLE_DEFAULT`
- `resolve.AUTO_CAPTION_TELETEXT`
- `resolve.AUTO_CAPTION_NETFLIX`

lineBreakTypes:

- `resolve.AUTO_CAPTION_LINE_SINGLE`
- `resolve.AUTO_CAPTION_LINE_DOUBLE`

Looking up Render Settings

This section covers the supported settings for the method SetRenderSettings({settings})

The parameter setting is a dictionary containing the following keys:

- `SelectAllFrames` : Bool (when set True, the settings MarkIn and MarkOut are ignored)
- `MarkIn` : int
- `MarkOut` : int
- `TargetDir` : string
- `CustomName` : string
- `UniqueFilenameStyle` : 0 - Prefix, 1 - Suffix.
- `ExportVideo` : Bool
- `ExportAudio` : Bool
- `FormatWidth` : int
- `FormatHeight` : int
- `FrameRate` : float (examples: 23.976, 24)
- `PixelAspectRatio` : string (for SD resolution: `16_9` or `4_3`) (other resolutions: `square` or `cinemascope`)
- `VideoQuality` possible values for current codec (if applicable):

- 0 (int) - will set quality to automatic
- [1 -> MAX] (int) - will set input bit rate

- [`Least` , `Low` , `Medium` , `High` , `Best`] (String) - will set input quality level
- `AudioCodec` : string (example: `aac`)
- `AudioBitDepth` : int
- `AudioSampleRate` : int
- `ColorSpaceTag` : string (example: "Same as Project", `AstroDesign`)
- `GammaTag` : string (example: "Same as Project", `ACEScct`)
- `ExportAlpha` : Bool
- `EncodingProfile` : string (example: `Main10`). Can only be set for H.264 and H.265.
- `MultiPassEncode` : Bool. Can only be set for H.264.
- `AlphaMode` : 0 - Premultiplied, 1 - Straight. Can only be set if `ExportAlpha` is true.
- `NetworkOptimization` : Bool. Only supported by QuickTime and MP4 formats.
- `ClipStartFrame` : int
- `TimelineStartTimecode` : string (example: "01:00:00:00")
- `ReplaceExistingFilesInPlace` : Bool
- `ExportSubtitle` : Bool
- `SubtitleFormat` : string (options: `BurnIn` , `EmbeddedCaptions` , `SeparateFile`)

Looking up timeline export properties

This section covers the parameters for the argument Export(fileName, exportType, exportSubtype).

exportType can be one of the following constants:

- `resolve.EXPORT_AAF`
- `resolve.EXPORT_DRT`
- `resolve.EXPORT_EDL`
- `resolve.EXPORT_FCP_7_XML`
- `resolve.EXPORT_FCPXML_1_8`
- `resolve.EXPORT_FCPXML_1_9`
- `resolve.EXPORT_FCPXML_1_10`
- `resolve.EXPORT_HDR_10_PROFILE_A`
- `resolve.EXPORT_HDR_10_PROFILE_B`
- `resolve.EXPORT_TEXT_CSV`
- `resolve.EXPORT_TEXT_TAB`
- `resolve.EXPORT_DOLBY_VISION_VER_2_9`
- `resolve.EXPORT_DOLBY_VISION_VER_4_0`
- `resolve.EXPORT_DOLBY_VISION_VER_5_1`
- `resolve.EXPORT_OTIO`
- `resolve.EXPORT_ALE`
- `resolve.EXPORT_ALE_CDL` exportSubtype can be one of the following enums:

- `resolve.EXPORT_NONE`

- `resolve.EXPORT_AAF_NEW`
- `resolve.EXPORT_AAF_EXISTING`
- `resolve.EXPORT_CDL`
- `resolve.EXPORT_SDL`
- `resolve.EXPORT_MISSING_CLIPS`

Please note that exportSubType is a required parameter for `resolve.EXPORT_AAF` and `resolve.EXPORT_EDL` . For rest of the exportType, exportSubtype is ignored.

When exportType is `resolve.EXPORT_AAF` , valid exportSubtype values are `resolve.EXPORT_AAF_NEW` and `resolve.EXPORT_AAF_EXISTING` .

When exportType is `resolve.EXPORT_EDL` , valid exportSubtype values are `resolve.EXPORT_CDL` , `resolve.EXPORT_SDL` , `resolve.EXPORT_MISSING_CLIPS` and `resolve.EXPORT_NONE` . Note: Replace `resolve.` when using the constants above, if a different Resolve class instance name is used.

Unsupported exportType types

Starting with DaVinci Resolve 18.1, the following export types are not supported:

```
- `resolve.EXPORT_FCPXML_1_3`
- `resolve.EXPORT_FCPXML_1_4`
- `resolve.EXPORT_FCPXML_1_5`
- `resolve.EXPORT_FCPXML_1_6`
- `resolve.EXPORT_FCPXML_1_7`
```

Looking up Timeline item properties

This section covers additional notes for the function "TimelineItem:SetProperty" and "TimelineItem:GetProperty". These functions are used to get and set properties mentioned.

The supported keys with their accepted values are:

- `Pan` : floating point values from -4.0*width to 4.0*width
- `Tilt` : floating point values from -4.0*height to 4.0*height
- `ZoomX` : floating point values from 0.0 to 100.0
- `ZoomY` : floating point values from 0.0 to 100.0
- `ZoomGang` : a boolean value
- `RotationAngle` : floating point values from -360.0 to 360.0
- `AnchorPointX` : floating point values from -4.0*width to 4.0*width
- `AnchorPointY` : floating point values from -4.0*height to 4.0*height
- `Pitch` : floating point values from -1.5 to 1.5
- `Yaw` : floating point values from -1.5 to 1.5
- `FlipX` : boolean value for flipping horizontally
- `FlipY` : boolean value for flipping vertically
- `CropLeft` : floating point values from 0.0 to width
- `CropRight` : floating point values from 0.0 to width
- `CropTop` : floating point values from 0.0 to height
- `CropBottom` : floating point values from 0.0 to height
- `CropSoftness` : floating point values from -100.0 to 100.0
- `CropRetain` : boolean value for "Retain Image Position" checkbox
- `DynamicZoomEase` : A value from the following constants
    - DYNAMIC_ZOOM_EASE_LINEAR = 0
    - DYNAMIC_ZOOM_EASE_IN
    - DYNAMIC_ZOOM_EASE_OUT
    - DYNAMIC_ZOOM_EASE_IN_AND_OUT
- `CompositeMode` : A value from the following constants
    - COMPOSITE_NORMAL = 0
    - COMPOSITE_ADD

  - COMPOSITE_SUBTRACT
  - COMPOSITE_DIFF
  - COMPOSITE_MULTIPLY
  - COMPOSITE_SCREEN
  - COMPOSITE_OVERLAY
  - COMPOSITE_HARDLIGHT
  - COMPOSITE_SOFTLIGHT
  - COMPOSITE_DARKEN
  - COMPOSITE_LIGHTEN
  - COMPOSITE_COLOR_DODGE
  - COMPOSITE_COLOR_BURN
  - COMPOSITE_EXCLUSION
  - COMPOSITE_HUE
  - COMPOSITE_SATURATE
  - COMPOSITE_COLORIZE
  - COMPOSITE_LUMA_MASK
  - COMPOSITE_DIVIDE
  - COMPOSITE_LINEAR_DODGE
  - COMPOSITE_LINEAR_BURN
  - COMPOSITE_LINEAR_LIGHT
  - COMPOSITE_VIVID_LIGHT
  - COMPOSITE_PIN_LIGHT
  - COMPOSITE_HARD_MIX
  - COMPOSITE_LIGHTER_COLOR
  - COMPOSITE_DARKER_COLOR
  - COMPOSITE_FOREGROUND
  - COMPOSITE_ALPHA
  - COMPOSITE_INVERTED_ALPHA
  - COMPOSITE_LUM
  - COMPOSITE_INVERTED_LUM
- `Opacity` : floating point value from 0.0 to 100.0
- `Distortion` : floating point value from -1.0 to 1.0
- `RetimeProcess` : A value from the following constants
  - RETIME_USE_PROJECT = 0
  - RETIME_NEAREST
  - RETIME_FRAME_BLEND
  - RETIME_OPTICAL_FLOW
- `MotionEstimation` : A value from the following constants
  - MOTION_EST_USE_PROJECT = 0
  - MOTION_EST_STANDARD_FASTER
  - MOTION_EST_STANDARD_BETTER
  - MOTION_EST_ENHANCED_FASTER
  - MOTION_EST_ENHANCED_BETTER
  - MOTION_EST_SPEED_WARP_BETTER
  - MOTION_EST_SPEED_WARP_FASTER
- `Scaling` : A value from the following constants
  - SCALE_USE_PROJECT = 0
  - SCALE_CROP
  - SCALE_FIT
  - SCALE_FILL
  - SCALE_STRETCH
- `ResizeFilter` : A value from the following constants
  - RESIZE_FILTER_USE_PROJECT = 0
  - RESIZE_FILTER_SHARPER

Table of Contents

- RESIZE_FILTER_SMOOTHER
- RESIZE_FILTER_BICUBIC
- RESIZE_FILTER_BILINEAR
- RESIZE_FILTER_BESSEL
- RESIZE_FILTER_BOX
- RESIZE_FILTER_CATMULL_ROM
- RESIZE_FILTER_CUBIC
- RESIZE_FILTER_GAUSSIAN
- RESIZE_FILTER_LANCZOS
- RESIZE_FILTER_MITCHELL
- RESIZE_FILTER_NEAREST_NEIGHBOR
- RESIZE_FILTER_QUADRATIC
- RESIZE_FILTER_SINC
- RESIZE_FILTER_LINEAR Values beyond the range will be clipped width and height are same as the UI max limits

The arguments can be passed as a key and value pair or they can be grouped together into a dictionary (for python) or table (for lua) and passed as a single argument.

Getting the values for the keys that uses constants will return the number which is in the constant

ExportLUT notes

The following section covers additional notes for TimelineItem.ExportLUT(exportType, path).

Supported values for `exportType` (enum) are:

```
- `resolve.EXPORT_LUT_17PTCUBE`
- `resolve.EXPORT_LUT_33PTCUBE`
- `resolve.EXPORT_LUT_65PTCUBE`
- `resolve.EXPORT_LUT_PANASONICVLUT`
```

Deprecated Resolve API Functions

The following API functions are deprecated.

ProjectManager

Name	Return	Definition
`GetProjectsInCurrentFolder()`	{project names...}	Returns a dict of project names in current folder.
`GetFoldersInCurrentFolder()`	{folder names...}	Returns a dict of folder names in current folder.

Project

Name	Return	Definition
`GetPresets()`	{presets...}	Returns a dict of presets and their information.
`GetRenderJobs()`	{render jobs...}	Returns a dict of render jobs and their information.

Name	Return	Definition
`GetRenderPresets()`	{presets…}	Returns a dict of render presets and their information.

MediaStorage

Name	Return	Definition
`GetMountedVolumes()`	{paths…}	Returns a dict of folder paths corresponding to mounted volumes displayed in Resolve's Media Storage.
`GetSubFolders(folderPath)`	{paths…}	Returns a dict of folder paths in the given absolute folder path.
`GetFiles(folderPath)`	{paths…}	Returns a dict of media and file listings in the given absolute folder path. Note that media listings may be logically consolidated entries.
`AddItemsToMediaPool(item1, item2, ...)`	{clips…}	Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is one or more file/folder paths. Returns a dict of the MediaPoolItems created.
`AddItemsToMediaPool([items...])`	{clips…}	Adds specified file/folder paths from Media Storage into current Media Pool folder. Input is an array of file/folder paths. Returns a dict of the MediaPoolItems created.

Folder

Name	Return	Definition
`GetClips()`	{clips…}	Returns a dict of clips (items) within the folder.
`GetSubFolders()`	{folders…}	Returns a dict of subfolders in the folder.

MediaPoolItem

Name	Return	Definition
`GetFlags()`	{colors…}	Returns a dict of flag colors assigned to the item.

Timeline

Name	Return	Definition
`GetItemsInTrack(trackType, index)`	{items...}	Returns a dict of Timeline items on the video or audio track (based on trackType) at specified

TimelineItem

Name	Return	Definition
`GetFusionCompNames()`	{names...}	Returns a dict of Fusion composition names associated with the timeline item.
`GetFlags()`	{colors...}	Returns a dict of flag colors assigned to the item.
`GetVersionNames(versionType)`	{names...}	Returns a dict of version names by provided versionType: 0 - local, 1 - remote.
`GetNumNodes()`	int	Returns the number of nodes in the current graph for the timeline item
`SetLUT(nodeIndex, lutPath)`	Bool	Sets LUT on the node mapping the node index provided, 1 <= nodeIndex <= total number of nodes. The lutPath can be an absolute path, or a relative path (based off custom LUT paths or the master LUT path). The operation is successful for valid lut paths that Resolve has already discovered (see Project.RefreshLUTList).
`GetLUT(nodeIndex)`	String	Gets relative LUT path based on the node index provided, 1 <= nodeIndex <= total number of nodes.
`GetNodeLabel(nodeIndex)`	string	Returns the label of the node at nodeIndex.

Unsupported Resolve API Functions

The following API (functions and parameters) are no longer supported. Use job IDs instead of indices.

Project

Name	Return	Definition
`StartRendering(index1, index2, ...)`	Bool	Please use unique job ids (string) insteac of indices.

Name	Return	Definition
`StartRendering([idxs...])`	Bool	Please use unique job ids (string) insteac of indices.
`DeleteRenderJobByIndex(idx)`	Bool	Please use unique job ids (string) insteac of indices.
`GetRenderJobStatus(idx)` `GetSetting and SetSetting`	{status info}{}	Please use unique job ids (string) insteac of indices. settingName videoMonitorUseRec601For422SDI is no replaced with videoMonitorUseMatrixOverrideFor422S and videoMonitorMatrixOverrideFor422SDI. settingName perfProxyMediaOn is now replaced with perfProxyMediaMode whi takes values 0 - disabled, 1 - when available, 2 - when source not available.