Advanced Workflows

PART 17 — CONTENTS

Chapter 196

Workflow Integrations

This chapter describes third party Workflow Integration and Codec plugins for DaVinci Resolve.

Contents

Workflow Integrations in DaVinci Resolve (Studio Version Only)

DaVinci Resolve allows third parties to create their own custom interface plugins using scripting languages. This makes possible a direct integration between DaVinci Resolve and other software programs, for a variety of uses. More than one Integration plugin can be active at the same time.

After installation, plugins can be enabled in DaVinci Resolve by going to Workspace > Workflow Integrations, and selecting your plugin from the drop-down menu.

Creating Workflow Integration Plugins

Users can write their own Workflow Integration Plugin (an Electron app), using Resolve Javascript's API, and Python or Lua scripts. For more information on how to create a Workflow Integration Plugin go to Help > Documentation > Developer, and open up the Workflow Integrations folder for technical details and sample code.

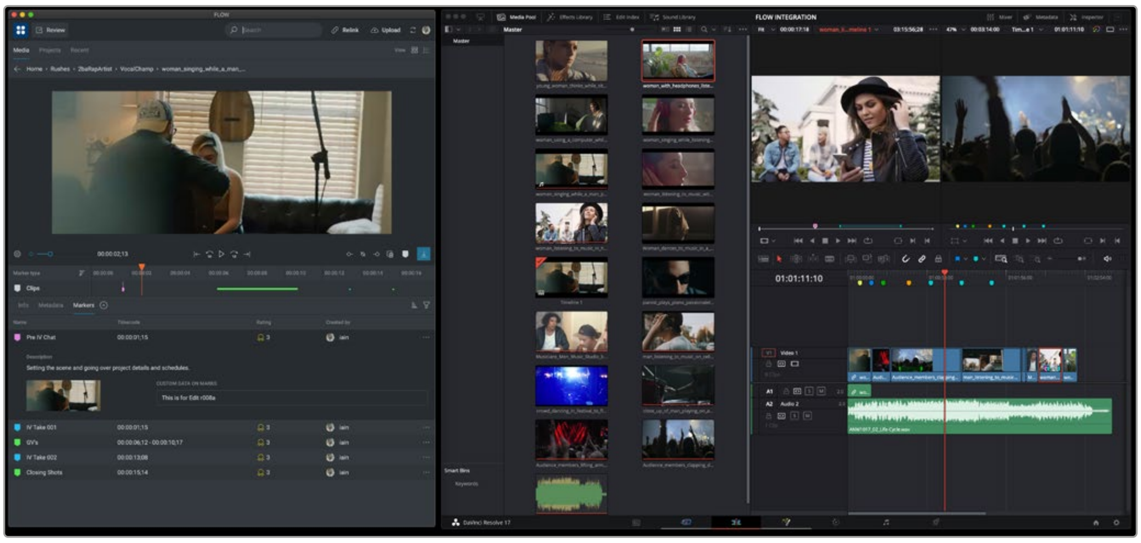
Workflow Integration Plugins

There are several Media Asset Management (MAM) systems that can now directly be accessed through DaVinci Resolve using the Workflow Integration Plugins.

EditShare

EditShare has created a workflow integration plugin that allows DaVinci Resolve to interface directly with their FLOW media management system. This plugin allows you to comment, search, and preview media in FLOW without leaving DaVinci Resolve. You can also upload revisions, manage proxy media, and maintain full metadata support throughout the process.

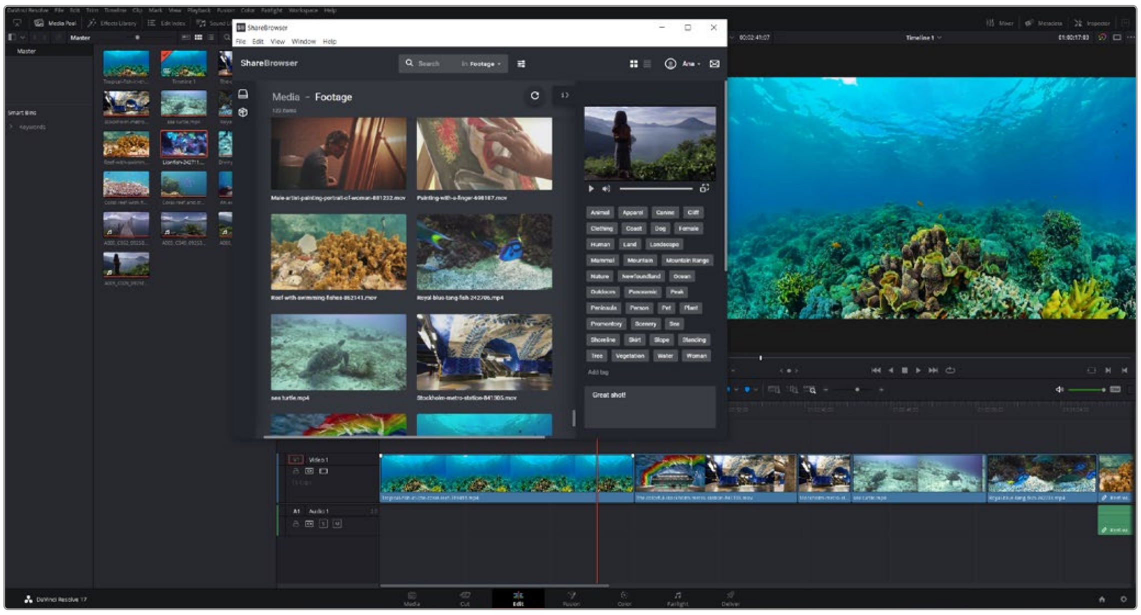
For more information on this plugin and how FLOW works with DaVinci Resolve go to:
https://editshare.com/say-hello-to-flow-and-davinci-resolve-studio/


EditShare's FLOW Integration Plugin

Studio Network Solutions (SNS)

Studio Network Solutions (SNS) created the ShareBrowser Integration Plugin to interface between DaVinci Resolve and their ShareBrowser media asset management software, included with SNS EVO media servers. This plugin allows your team to search, tag, preview, comment, organize, and import media without leaving the DaVinci Resolve interface. Your team can directly import the media into a DaVinci Resolve project and the metadata you entered carries over along with the media.

For more information on this plugin and how SNS's high-speed server or cloud solutions work with DaVinci Resolve, go to: https://www.studionetworksolutions.com/.


SNS ShareBrowser Integration Plugin

Codec Plugins (Studio Version Only)

Codec plugins allow third parties to install new codecs for encoding in the Deliver page that are not currently supported in the main DaVinci Resolve software. This opens the door for extremely specific deliverables that would normally require passes through multiple programs to deliver.

MainConcept

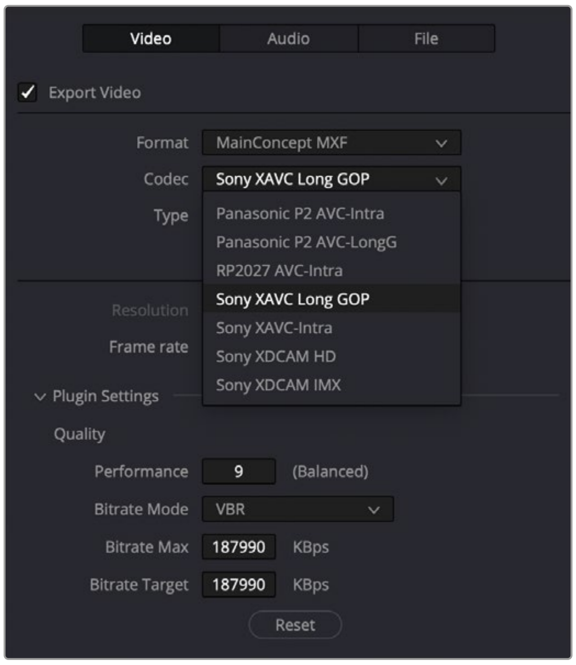
The MainConcept Codec Plugin allows you to render your DaVinci Resolve Studio timelines in a variety of new codecs:

— AS-11 UK SD, AS-11 UK HD along with an included XML metadata file to create AS-11 UK DPP compliant content.

— MainConcept's software HEVC Main and Main 10 profiles, allowing H.265 files in 8-bit/10-bit 4:2:0/4:2:2 at up to 8K resolution.

— MainConcept MXF and MP4, allowing encoding into the native camera formats used by Sony XAVC/XDCAM and Panasonic P2 AVC based cameras.

More information on the MainConcept Codec Plugin for DaVinci Resolve can be found here: https://www.mainconcept.com/blackmagic-plugins



✓ Export Video

Format	MainConcept MXF
Codec	Sony XAVC Long GOP
Performance	9 (Balanced)
Bitrate Mode	VBR
Bitrate Max	187990 KBps
Bitrate Target	187990 KBps

The MainConcept Codec Plugin for DaVinci Resolve options in the Deliver page

Chapter 197

Creating DCTL LUTs

This chapter describes how to create DCTL LUTs to perform your own custom mathematical transformations in DaVinci Resolve.

Contents

About DCTL

DCTL files are actually color transformation scripts that DaVinci Resolve sees and applies just like any other LUT. Unlike other LUTs, which are 1D or 3D lookup tables of values that approximate image transformations using interpolation, DCTL files are actually comprised of computer code that directly transforms images using combinations of math functions that you devise. Additionally, DCTL files run natively on the GPU of your workstation, so they can be fast.

Anyone with the mathematical know-how can make and install a DCTL. Simply enter your transformation code, using a syntax that's similar to C (described in more detail below), into any text editor capable of saving a plain ASCII text file, and make sure its name ends with the ".dctl" (DaVinci Color Transform Language) file extension. Once that's done, move the file to the LUT directory of your workstation. Where that is depends on which OS you're using:

— **On Mac OS X:** Library/Application Support/Blackmagic Design/DaVinci Resolve/LUT/
— **On Windows:** C:\ProgramData\Blackmagic Design\DaVinci Resolve\Support\LUT
— **On Linux:** /home/resolve/LUT

When DaVinci Resolve starts up, assuming the syntax of your .dctl is correct, they appear in the Color page Node contextual menu within the DaVinci CTL submenu.

DCTL Syntax

Users need to put __DEVICE__ in front of each function they write. For example:

```
__DEVICE__ float2 DoSomething()
```

The main entry function (transform) should come after all other functions, with the following format argument:

```
__DEVICE__ float3 transform(float p_R, float p_G, float p_B)
```

The main entry function must also have a float3 return value.

For the following floating point math functions, please use the described syntax:

```
float _fabs(float)                  // Absolute Value
float _powf(float x, float y        // Compute x to the power of y
float _logf(float)                  // Natural logarithm
float _log2f(float)                 // Base 2 logarithm
float _log10f(float)                // Base 10 logarithm
float _exp2f(float)                 // Exponential base 2
float _expf(float)                  // Exponential base E
float _copysignf(float x, float y)  // Return x with sign changed to sign y
float _fmaxf(float x, float y)      // Return y if x < y
float _fminf(float x, float y)       // Return y if x > y

float _saturatef(float x, float      // Return min(max(x, minVal), maxVal)
minVal, float maxVal)

float _sqrtf(float)                  // Square root

int   _ceil(float                    // Round to integer toward + infinity

int   _floor(float)                  // Round to integer toward - infinity

float _fmod(float x, float y)        // Modulus. Returns x – y * trunc(x / y)

float _fremainder(float x, float y)  // Floating point remainder

int   _round(float x)                // Integral value nearest to x rounding

float _hypotf(float x, float y)      // Square root of (x^2 + y^2)

float _atan2f(float x)               // Arc tangent of (y / x)

float _sinf(float x)                 // Sine

float _cosf(float x)                 // Cosine

float _acosf(float x)                // Arc cosine

float _asinf(float x)                // Arc sine

float _fdivide(float x, float y)     // Return (x / y)

float _frecip(float x)               // Return (1 / x)
```

The following functions support integer type:

```
min, max, abs, rotate
```

Other supported C Math functions include:

```
acosh, acospi, asinh, asinpi, atan, atanh, atanpi, atan2pi, cbrt, cosh, cospi,
exp10, expm1, trunc, fdim, fma, lgamma, log1p, logb, rint, round, rsqrt,
sincos, sinh, sinpi, tan, tanh, tanpi, tgamma
```

Vector types float2, float3, and float4 are supported. The data fields are:

```
float x
float y
float z
float w
```

To generate a vector value, use make_floatN() where N = 2, 3, or 4.

Users can define their own structure using "typedef struct." For example:

```
typedef struct
    {
        float c00, c01, c02;
        float c10, c11, c12;
    } Matrix;
```

To declare constant memory, use *__CONSTANT__*. For example:

```
__CONSTANT__ float NORM[] = {1.0f / 3.0f, 1.0f / 3.0f, 1.0f / 3.0f};
```

To pass the constant memory as a function argument, use the *__CONSTANTREF__* qualifier, e.g.:

```
__DEVICE__ float DoSomething(__CONSTANTREF__ float* p_Params)
```

A float value must have the 'f' character at the end (e.g. 1.2f).

A Simple DCT LUT Example

The following code shows an example of how to create a simple color gain transformation using the DCT LUT syntax.

```
// Example to demonstrate simple color gain transformation
__DEVICE__ float3 transform(float p_R, float p_G, float p_B)
{
    const float r = p_R * 1.2f;
    const float g = p_G * 1.1f;
    const float b = p_B * 1.2f;
    return make_float3(r, g, b);
}
```

A Matrix DCT LUT Example

The following code shows an example of creating a matrix transform using the DCT LUT syntax.

// Example to demonstrate the usage of user defined matrix type to transform RGB to YUV in Rec. 709

```
__CONSTANT__ float RGBToYUVMat[9] = { 0.2126f ,  0.7152f , 0.0722f,
                                     -0.09991f, -0.33609f, 0.436f,
                                      0.615f  , -0.55861f, -0.05639f };

__DEVICE__ float3 transform(int p_Width, int p_Height, int p_X, int p_Y,
float p_R, float p_G, float p_B)
{
    float3 result;

    result.x = RGBToYUVMat[0] * p_R + RGBToYUVMat[1] * p_G + RGBToYUVMat[2] *
p_B;
    result.y = RGBToYUVMat[3] * p_R + RGBToYUVMat[4] * p_G + RGBToYUVMat[5] *
p_B;
    result.z = RGBToYUVMat[6] * p_R + RGBToYUVMat[7] * p_G + RGBToYUVMat[8] *
p_B;

    return result;
}
```

A More Complex DCT LUT Example

The following code shows an example of creating a mirror effect, illustrating how you can access pixels spatially.

```
// Example of spatial access for mirror effect

__DEVICE__ float3 transform(int p_Width, int p_Height, int p_X, int p_Y, __
TEXTURE__ p_TexR, __TEXTURE__ p_TexG, __TEXTURE__ p_TexB)
{
    const bool isMirror = (p_X < (p_Width / 2));
    const float r = (isMirror) ? _tex2D(p_TexR, p_X, p_Y) : _tex2D(p_TexR, p_
Width - 1 - p_X, p_Y);
    const float g = (isMirror) ? _tex2D(p_TexG, p_X, p_Y) : _tex2D(p_TexG, p_
Width - 1 - p_X, p_Y);
    const float b = (isMirror) ? _tex2D(p_TexB, p_X, p_Y) : _tex2D(p_TexB, p_
Width - 1 - p_X, p_Y);
    return make_float3(r, g, b);
}
```

Chapter 198

TCP Protocol for DaVinci Resolve Transport Control

This chapter describes how to create third-party utilities that use Transport Control with DaVinci Resolve.

Contents

About the TCP Protocol Version 1.2

This protocol defines the communication standard between third-party applications ("Client") and DaVinci Resolve ("Server") using the TCP protocol.

Port number 9060 will be used by the server. SSL will not be used in this protocol. Communication takes the form of request-response messages, where the Client initiates a command, and the Server responds appropriately.

To use this protocol, you must first type the following string into the Advanced panel of the DaVinci Resolve System Preferences:

```
System.Remote.Control = 1
```

Data Types

The following data types are used in this protocol:

— **float (f):** A 4-byte IEEE 754 single precision float

— **int (i):** A 4-bytes signed int

— **unsigned char (uc):** A 1-byte unsigned char (0–255)

— **string (s):** A UTF-8 encoded string. No terminator is specified. The string is a composite type, transmitted as a single int (i) specifying the number of characters in the string (N), followed by N unsigned chars (uc) containing the letters of the string.

NOTE: The bytes of the float and int types are transmitted in little endian order.

Command Format

Commands are transmitted as a single string (using characters a–z (0x61 – 0x7A) only), followed by any additional payload required by the command in the definition.

Response Format

The response to any command is composed of a status byte (unsigned char), followed by any additional payload required by the response.

Communication Delays

Once the first byte of the command string is sent, the rest of the command string and the payload data must follow without delay. At the end of COMMAND, the server must respond immediately. If there is a delay of more than 5 seconds during this process, the party waiting for data may drop the connection assuming that the peer has become unresponsive.

There is currently no limit on the delay between two consecutive commands.

NOTE: Alternatively, a maximum allowable delay may be defined, in which case, the client may issue periodic 'connect' commands to keep the connection alive.

Status Response Values

The meaning of the status values are as follows:

— **0x00:** Command was executed successfully. Any additional payload is sent as expected.

— **0xFF:** Command could not be executed successfully. No additional payload will follow.

TCP Protocol Stream

The following commands can be sent over the protocol stream.

connect

The client initiates the stream by sending a connect command string. There is no payload. The server responds with a status value of 0x00.

goto

The client sends a goto command string followed by four unsigned chars representing the hour, minute, second, and frame of the timecode.

The server responds with an appropriate status byte based on the execution of the command.

play

The client sends a play command string followed by a floating point value. Play in real-time is 1.0, stop is 0.0, reverse is -1.0, 2x is 2.0, etc.

The server responds with an appropriate status byte based on the execution of the command.

gettc

The client sends a gettc command string.

The server responds with an appropriate status byte (status byte may be 0xFF if no timeline exists, for instance). If the status byte is 0x00, it is followed by four unsigned chars representing the hour, minute, second, and frame of the timecode.

getframerate

The client sends a getframerate command string.

The server responds with an appropriate status byte. If the status byte is 0x00, it is followed by a floating point value for the frame rate.