FFmpeg Utilities Documentation

Table of Contents

1 Description# TOC

This document describes some generic features and utilities provided by the libavutil library.

2 Syntax# TOC

This section documents the syntax and formats employed by the FFmpeg libraries and tools.

2.1 Quoting and escaping# TOC

FFmpeg adopts the following quoting and escaping mechanism, unless explicitly specified. The following rules are applied:

- ''' and '\' are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.
- A special character is escaped by prefixing it with a '\'.
- All characters enclosed between '''' are included literally in the parsed string. The quote character ''' itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

Note that you may need to add a second level of escaping when using the command line or a script, which depends on the syntax of the adopted shell language.

The function `av_get_token` defined in `libavutil/avstring.h` can be used to parse a token quoted or escaped according to the rules defined above.

The tool `tools/ffescape` in the FFmpeg source tree can be used to automatically quote or escape a string in a script.

2.1.1 Examples# TOC

- Escape the string `Crime d'Amour` containing the `'` special character:

  ```
  Crime d\'Amour
  ```

- The string above contains a quote, so the `'` needs to be escaped when quoting it:

  ```
  'Crime d'\''Amour'
  ```

- Include leading or trailing whitespaces using quoting:

  ```
  '  this string starts and ends with whitespaces  '
  ```

- Escaping and quoting can be mixed together:

  ```
  ' The string '\'string\'' is a string '
  ```

- To include a literal '\' you can use either escaping or quoting:

  ```
  'c:\foo' can be written as c:\\foo
  ```

2.2 Date# TOC

The accepted syntax is:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...]]])|(HHMMSS[.m...]]]))[Z]
now
```

If the value is "now" it takes the current time.

Time is local time unless Z is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

2.3 Time duration# TOC

There are two accepted syntaxes for expressing time duration.

```
[-][HH:]MM:SS[.m...]
```

HH expresses the number of hours, *MM* the number of minutes for a maximum of 2 digits, and *SS* the number of seconds for a maximum of 2 digits. The *m* at the end expresses decimal value for *SS*.

or

```
[-]S+[.m...]
```

S expresses the number of seconds, with the optional decimal part *m*.

In both expressions, the optional '-' indicates negative duration.

2.3.1 Examples# TOC

The following examples are all valid time duration:

'55'

   55 seconds

'12:03:45'

   12 hours, 03 minutes and 45 seconds

'23.189'

   23.189 seconds

2.4 Video size# TOC

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation.

The following abbreviations are recognized:

'ntsc'

   720x480

'pal'

   720x576

'qntsc'

   352x240

'qpal'

352x288

`sntsc`

640x480

`spal`

768x576

`film`

352x240

`ntsc-film`

352x240

`sqcif`

128x96

`qcif`

176x144

`cif`

352x288

`4cif`

704x576

`16cif`

1408x1152

`qqvga`

160x120

`qvga`

320x240

`vga`

640x480

'svga'

800x600

'xga'

1024x768

'uxga'

1600x1200

'qxga'

2048x1536

'sxga'

1280x1024

'qsxga'

2560x2048

'hsxga'

5120x4096

'wvga'

852x480

'wxga'

1366x768

'wsxga'

1600x1024

'wuxga'

1920x1200

'woxga'

2560x1600

'wqsxga'

3200x2048

'wquxga'

3840x2400

'whsxga'

6400x4096

'whuxga'

7680x4800

'cga'

320x200

'ega'

640x350

'hd480'

852x480

'hd720'

1280x720

'hd1080'

1920x1080

'2k'

2048x1080

'2kflat'

1998x1080

'2kscope'

2048x858

'4k'

4096x2160

'4kflat'

3996x2160

'4kscope'

4096x1716

'nhd'

640x360

'hqvga'

240x160

'wqvga'

400x240

'fwqvga'

432x240

'hvga'

480x320

'qhd'

960x540

2.5 Video rate# TOC

Specify the frame rate of a video, expressed as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation.

The following abbreviations are recognized:

'ntsc'

30000/1001

'pal'

25/1

'qntsc'

30000/1001

'qpal'

25/1

'sntsc'

30000/1001

'spal'

25/1

'film'

24/1

'ntsc-film'

24000/1001

2.6 Ratio# TOC

A ratio can be expressed as an expression, or in the form *numerator*:*denominator*.

Note that a ratio with infinite (1/0) or negative value is considered valid, so you should check on the returned value if you want to exclude those values.

The undefined value can be expressed using the "0:0" string.

2.7 Color# TOC

It can be the name of a color as defined below (case insensitive match) or a [0x|#]RRGGBB[AA] sequence, possibly followed by @ and a string representing the alpha component.

The alpha component may be a string composed by "0x" followed by an hexadecimal number or a decimal number between 0.0 and 1.0, which represents the opacity value ('0x00' or '0.0' means completely transparent, '0xff' or '1.0' completely opaque). If the alpha component is not specified then '0xff' is assumed.

The string 'random' will result in a random color.

The following names of colors are recognized:

'AliceBlue'

    0xF0F8FF

'AntiqueWhite'

    0xFAEBD7

'Aqua'

    0x00FFFF

'Aquamarine'

    0x7FFFD4

'Azure'

    0xF0FFFF

'Beige'

    0xF5F5DC

'Bisque'

    0xFFE4C4

'Black'

    0x000000

'BlanchedAlmond'

    0xFFEBCD

'Blue'

    0x0000FF

'BlueViolet'

    0x8A2BE2

'Brown'

0xA52A2A

'BurlyWood'

0xDEB887

'CadetBlue'

0x5F9EA0

'Chartreuse'

0x7FFF00

'Chocolate'

0xD2691E

'Coral'

0xFF7F50

'CornflowerBlue'

0x6495ED

'Cornsilk'

0xFFF8DC

'Crimson'

0xDC143C

'Cyan'

0x00FFFF

'DarkBlue'

0x00008B

'DarkCyan'

0x008B8B

'DarkGoldenRod'

0xB8860B

'DarkGray'

0xA9A9A9

'DarkGreen'

0x006400

'DarkKhaki'

0xBDB76B

'DarkMagenta'

0x8B008B

'DarkOliveGreen'

0x556B2F

'Darkorange'

0xFF8C00

'DarkOrchid'

0x9932CC

'DarkRed'

0x8B0000

'DarkSalmon'

0xE9967A

'DarkSeaGreen'

0x8FBC8F

'DarkSlateBlue'

0x483D8B

'DarkSlateGray'

0x2F4F4F

'DarkTurquoise'

0x00CED1

'DarkViolet'

0x9400D3

'DeepPink'

0xFF1493

'DeepSkyBlue'

0x00BFFF

'DimGray'

0x696969

'DodgerBlue'

0x1E90FF

'FireBrick'

0xB22222

'FloralWhite'

0xFFFAF0

'ForestGreen'

0x228B22

'Fuchsia'

0xFF00FF

'Gainsboro'

0xDCDCDC

'GhostWhite'

0xF8F8FF

‘Gold’

0xFFD700

‘GoldenRod’

0xDAA520

‘Gray’

0x808080

‘Green’

0x008000

‘GreenYellow’

0xADFF2F

‘HoneyDew’

0xF0FFF0

‘HotPink’

0xFF69B4

‘IndianRed’

0xCD5C5C

‘Indigo’

0x4B0082

‘Ivory’

0xFFFFF0

‘Khaki’

0xF0E68C

‘Lavender’

0xE6E6FA

'LavenderBlush'

0xFFF0F5

'LawnGreen'

0x7CFC00

'LemonChiffon'

0xFFFACD

'LightBlue'

0xADD8E6

'LightCoral'

0xF08080

'LightCyan'

0xE0FFFF

'LightGoldenRodYellow'

0xFAFAD2

'LightGreen'

0x90EE90

'LightGrey'

0xD3D3D3

'LightPink'

0xFFB6C1

'LightSalmon'

0xFFA07A

'LightSeaGreen'

0x20B2AA

'LightSkyBlue'

0x87CEFA

'LightSlateGray'

0x778899

'LightSteelBlue'

0xB0C4DE

'LightYellow'

0xFFFFE0

'Lime'

0x00FF00

'LimeGreen'

0x32CD32

'Linen'

0xFAF0E6

'Magenta'

0xFF00FF

'Maroon'

0x800000

'MediumAquaMarine'

0x66CDAA

'MediumBlue'

0x0000CD

'MediumOrchid'

0xBA55D3

'MediumPurple'

0x9370D8

'MediumSeaGreen'

0x3CB371

'MediumSlateBlue'

0x7B68EE

'MediumSpringGreen'

0x00FA9A

'MediumTurquoise'

0x48D1CC

'MediumVioletRed'

0xC71585

'MidnightBlue'

0x191970

'MintCream'

0xF5FFFA

'MistyRose'

0xFFE4E1

'Moccasin'

0xFFE4B5

'NavajoWhite'

0xFFDEAD

'Navy'

0x000080

'OldLace'

0xFDF5E6

'Olive'

0x808000

'OliveDrab'

0x6B8E23

'Orange'

0xFFA500

'OrangeRed'

0xFF4500

'Orchid'

0xDA70D6

'PaleGoldenRod'

0xEEE8AA

'PaleGreen'

0x98FB98

'PaleTurquoise'

0xAFEEEE

'PaleVioletRed'

0xD87093

'PapayaWhip'

0xFFEFD5

'PeachPuff'

0xFFDAB9

'Peru'

0xCD853F

'Pink'

0xFFC0CB

'Plum'

0xDDA0DD

'PowderBlue'

0xB0E0E6

'Purple'

0x800080

'Red'

0xFF0000

'RosyBrown'

0xBC8F8F

'RoyalBlue'

0x4169E1

'SaddleBrown'

0x8B4513

'Salmon'

0xFA8072

'SandyBrown'

0xF4A460

'SeaGreen'

0x2E8B57

'SeaShell'

0xFFF5EE

'Sienna'

0xA0522D

'Silver'

0xC0C0C0

'SkyBlue'

0x87CEEB

'SlateBlue'

0x6A5ACD

'SlateGray'

0x708090

'Snow'

0xFFFAFA

'SpringGreen'

0x00FF7F

'SteelBlue'

0x4682B4

'Tan'

0xD2B48C

'Teal'

0x008080

'Thistle'

0xD8BFD8

'Tomato'

0xFF6347

'Turquoise'

0x40E0D0

'Violet'

0xEE82EE

'Wheat'

0xF5DEB3

'White'

0xFFFFFF

'WhiteSmoke'

0xF5F5F5

'Yellow'

0xFFFF00

'YellowGreen'

0x9ACD32

2.8 Channel Layout# TOC

A channel layout specifies the spatial disposition of the channels in a multi-channel audio stream. To specify a channel layout, FFmpeg makes use of a special syntax.

Individual channels are identified by an id, as given by the table below:

'FL'

front left

'FR'

front right

'FC'

    front center

'LFE'

    low frequency

'BL'

    back left

'BR'

    back right

'FLC'

    front left-of-center

'FRC'

    front right-of-center

'BC'

    back center

'SL'

    side left

'SR'

    side right

'TC'

    top center

'TFL'

    top front left

'TFC'

    top front center

'TFR'

top front right

'TBL'

top back left

'TBC'

top back center

'TBR'

top back right

'DL'

downmix left

'DR'

downmix right

'WL'

wide left

'WR'

wide right

'SDL'

surround direct left

'SDR'

surround direct right

'LFE2'

low frequency 2

Standard channel layout compositions can be specified by using the following identifiers:

'mono'

FC

'stereo'

FL+FR

`2.1`

FL+FR+LFE

`3.0`

FL+FR+FC

`3.0(back)`

FL+FR+BC

`4.0`

FL+FR+FC+BC

`quad`

FL+FR+BL+BR

`quad(side)`

FL+FR+SL+SR

`3.1`

FL+FR+FC+LFE

`5.0`

FL+FR+FC+BL+BR

`5.0(side)`

FL+FR+FC+SL+SR

`4.1`

FL+FR+FC+LFE+BC

`5.1`

FL+FR+FC+LFE+BL+BR

`5.1(side)`

FL+FR+FC+LFE+SL+SR

'6.0'

FL+FR+FC+BC+SL+SR

'6.0(front)'

FL+FR+FLC+FRC+SL+SR

'hexagonal'

FL+FR+FC+BL+BR+BC

'6.1'

FL+FR+FC+LFE+BC+SL+SR

'6.1'

FL+FR+FC+LFE+BL+BR+BC

'6.1(front)'

FL+FR+LFE+FLC+FRC+SL+SR

'7.0'

FL+FR+FC+BL+BR+SL+SR

'7.0(front)'

FL+FR+FC+FLC+FRC+SL+SR

'7.1'

FL+FR+FC+LFE+BL+BR+SL+SR

'7.1(wide)'

FL+FR+FC+LFE+BL+BR+FLC+FRC

'7.1(wide-side)'

FL+FR+FC+LFE+FLC+FRC+SL+SR

'octagonal'

FL+FR+FC+BL+BR+BC+SL+SR

'`downmix`'

DL+DR

A custom channel layout can be specified as a sequence of terms, separated by '+' or '|'. Each term can be:

- the name of a standard channel layout (e.g. '`mono`', '`stereo`', '`4.0`', '`quad`', '`5.0`', etc.)
- the name of a single channel (e.g. '`FL`', '`FR`', '`FC`', '`LFE`', etc.)
- a number of channels, in decimal, optionally followed by '`c`', yielding the default channel layout for that number of channels (see the function `av_get_default_channel_layout`)
- a channel layout mask, in hexadecimal starting with "0x" (see the `AV_CH_*` macros in `libavutil/channel_layout.h`.

Starting from libavutil version 53 the trailing character "c" to specify a number of channels will be required, while a channel layout mask could also be specified as a decimal number (if and only if not followed by "c").

See also the function `av_get_channel_layout` defined in `libavutil/channel_layout.h`.

3 Expression Evaluation# TOC

When evaluating an arithmetic expression, FFmpeg uses an internal formula evaluator, implemented through the `libavutil/eval.h` interface.

An expression may contain unary, binary operators, constants, and functions.

Two expressions *expr1* and *expr2* can be combined to form another expression "*expr1*;*expr2*". *expr1* and *expr2* are evaluated in turn, and the new expression evaluates to the value of *expr2*.

The following binary operators are available: +, -, *, /, ^.

The following unary operators are available: +, -.

The following functions are available:

`abs(x)`

Compute absolute value of *x*.

`acos(x)`

Compute arccosine of *x*.

`asin(x)`

Compute arcsine of *x*.

```
atan(x)
```

Compute arctangent of *x*.

```
between(x, min, max)
```

Return 1 if *x* is greater than or equal to *min* and lesser than or equal to *max*, 0 otherwise.

```
bitand(x, y)
bitor(x, y)
```

Compute bitwise and/or operation on *x* and *y*.

The results of the evaluation of *x* and *y* are converted to integers before executing the bitwise operation.

Note that both the conversion to integer and the conversion back to floating point can lose precision. Beware of unexpected results for large numbers (usually 2^53 and larger).

```
ceil(expr)
```

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

```
clip(x, min, max)
```

Return the value of *x* clipped between *min* and *max*.

```
cos(x)
```

Compute cosine of *x*.

```
cosh(x)
```

Compute hyperbolic cosine of *x*.

```
eq(x, y)
```

Return 1 if *x* and *y* are equivalent, 0 otherwise.

```
exp(x)
```

Compute exponential of *x* (with base `e`, the Euler's number).

```
floor(expr)
```

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

`gauss(x)`

Compute Gauss function of x, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

`gcd(x, y)`

Return the greatest common divisor of x and y. If both x and y are 0 or either or both are less than zero then behavior is undefined.

`gt(x, y)`

Return 1 if x is greater than y, 0 otherwise.

`gte(x, y)`

Return 1 if x is greater than or equal to y, 0 otherwise.

`hypot(x, y)`

This function is similar to the C function with the same name; it returns "sqrt($x*x + y*y$)", the length of the hypotenuse of a right triangle with sides of length x and y, or the distance of the point (x, y) from the origin.

`if(x, y)`

Evaluate x, and if the result is non-zero return the result of the evaluation of y, return 0 otherwise.

`if(x, y, z)`

Evaluate x, and if the result is non-zero return the evaluation result of y, otherwise the evaluation result of z.

`ifnot(x, y)`

Evaluate x, and if the result is zero return the result of the evaluation of y, return 0 otherwise.

`ifnot(x, y, z)`

Evaluate x, and if the result is zero return the evaluation result of y, otherwise the evaluation result of z.

`isinf(x)`

Return 1.0 if x is +/-INFINITY, 0.0 otherwise.

`isnan(x)`

Return 1.0 if x is NAN, 0.0 otherwise.

```
ld(var)
```

Load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

```
log(x)
```

Compute natural logarithm of *x*.

```
lt(x, y)
```

Return 1 if *x* is lesser than *y*, 0 otherwise.

```
lte(x, y)
```

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

```
max(x, y)
```

Return the maximum between *x* and *y*.

```
min(x, y)
```

Return the maximum between *x* and *y*.

```
mod(x, y)
```

Compute the remainder of division of *x* by *y*.

```
not(expr)
```

Return 1.0 if *expr* is zero, 0.0 otherwise.

```
pow(x, y)
```

Compute the power of *x* elevated *y*, it is equivalent to "$(x)^{(y)}$".

```
print(t)
print(t, l)
```

Print the value of expression *t* with loglevel *l*. If *l* is not specified then a default log level is used. Returns the value of the expression printed.

Prints t with loglevel l

```
random(x)
```

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

`root(expr, max)`

Find an input value for which the function represented by *expr* with argument *ld(0)* is 0 in the interval 0..*max*.

The expression in *expr* must denote a continuous function or the result is undefined.

ld(0) is used to represent the function input value, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(0)`. When the expression evaluates to 0 then the corresponding input value will be returned.

`sin(x)`

Compute sine of *x*.

`sinh(x)`

Compute hyperbolic sine of *x*.

`sqrt(expr)`

Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

`squish(x)`

Compute expression `1/(1 + exp(4*x))`.

`st(var, expr)`

Store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in the internal variable. Note, Variables are currently not shared between expressions.

`tan(x)`

Compute tangent of *x*.

`tanh(x)`

Compute hyperbolic tangent of *x*.

`taylor(expr, x)`
`taylor(expr, x, id)`

Evaluate a Taylor series at *x*, given an expression representing the `ld(id)`-th derivative of a function at 0.

When the series does not converge the result is undefined.

ld(id) is used to represent the derivative order in *expr*, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(id)`. If *id* is not specified then 0 is assumed.

Note, when you have the derivatives at y instead of 0, `taylor(expr, x-y)` can be used.

`time(0)`

Return the current (wallclock) time in seconds.

`trunc(expr)`

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

`while(cond, expr)`

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

The following constants are available:

`PI`

area of the unit disc, approximately 3.14

`E`

exp(1) (Euler's number), approximately 2.718

`PHI`

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

* works like AND

+ works like OR

For example the construct:
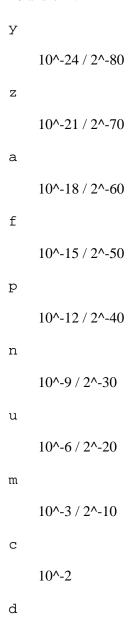
```
if (A AND B) then C
```

is equivalent to:

```
if(A*B, C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System unit prefixes. If 'i' is appended after the prefix, binary prefixes are used, which are based on powers of 1024 instead of powers of 1000. The 'B' postfix multiplies the value by 8, and can be appended after a unit prefix or used alone. This allows using for example 'KB', 'MiB', 'G' and 'B' as number postfix.

The list of available International System prefixes follows, with indication of the corresponding powers of 10 and of 2.

y

  $10^{-24} / 2^{-80}$

z

  $10^{-21} / 2^{-70}$

a

  $10^{-18} / 2^{-60}$

f

  $10^{-15} / 2^{-50}$

p

  $10^{-12} / 2^{-40}$

n

  $10^{-9} / 2^{-30}$

u

  $10^{-6} / 2^{-20}$

m

  $10^{-3} / 2^{-10}$

c

  10^{-2}

d

10^-1

h

10^2

k

10^3 / 2^10

K

10^3 / 2^10

M

10^6 / 2^20

G

10^9 / 2^30

T

10^12 / 2^40

P

10^15 / 2^40

E

10^18 / 2^50

Z

10^21 / 2^60

Y

10^24 / 2^70

4 OpenCL Options# TOC

When FFmpeg is configured with `--enable-opencl`, it is possible to set the options for the global OpenCL context.

The list of supported options follows:

`build_options`

> Set build options used to compile the registered kernels.
>
> See reference "OpenCL Specification Version: 1.2 chapter 5.6.4".

`platform_idx`

> Select the index of the platform to run OpenCL code.
>
> The specified index must be one of the indexes in the device list which can be obtained with `ffmpeg -opencl_bench` or `av_opencl_get_device_list()`.

`device_idx`

> Select the index of the device used to run OpenCL code.
>
> The specified index must be one of the indexes in the device list which can be obtained with `ffmpeg -opencl_bench` or `av_opencl_get_device_list()`.

5 See Also# TOC

ffmpeg, ffplay, ffprobe, ffserver, libavutil

6 Authors# TOC

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated using *makeinfo*.

FFmpeg Utilities Documentation

Table of Contents

1 Description# TOC

This document describes some generic features and utilities provided by the libavutil library.

2 Syntax# TOC

This section documents the syntax and formats employed by the FFmpeg libraries and tools.

2.1 Quoting and escaping# TOC

FFmpeg adopts the following quoting and escaping mechanism, unless explicitly specified. The following rules are applied:

- ''' and '\' are special characters (respectively used for quoting and escaping). In addition to them, there might be other special characters depending on the specific syntax where the escaping and quoting are employed.
- A special character is escaped by prefixing it with a '\'.
- All characters enclosed between ' ' ' are included literally in the parsed string. The quote character ' ' ' itself cannot be quoted, so you may need to close the quote and escape it.
- Leading and trailing whitespaces, unless escaped or quoted, are removed from the parsed string.

Note that you may need to add a second level of escaping when using the command line or a script, which depends on the syntax of the adopted shell language.

The function `av_get_token` defined in `libavutil/avstring.h` can be used to parse a token quoted or escaped according to the rules defined above.

The tool `tools/ffescape` in the FFmpeg source tree can be used to automatically quote or escape a string in a script.

2.1.1 Examples# TOC

- Escape the string `Crime d'Amour` containing the `'` special character:

  ```
  Crime d\'Amour
  ```

- The string above contains a quote, so the `'` needs to be escaped when quoting it:

  ```
  'Crime d'\''Amour'
  ```

- Include leading or trailing whitespaces using quoting:

  ```
  '  this string starts and ends with whitespaces  '
  ```

- Escaping and quoting can be mixed together:

  ```
  ' The string '\'string\'' is a string '
  ```

- To include a literal '\' you can use either escaping or quoting:

  ```
  'c:\foo' can be written as c:\\foo
  ```

2.2 Date# TOC

The accepted syntax is:

```
[(YYYY-MM-DD|YYYYMMDD)[T|t| ]]((HH:MM:SS[.m...]]])|(HHMMSS[.m...]]]))[Z]
now
```

If the value is "now" it takes the current time.

Time is local time unless Z is appended, in which case it is interpreted as UTC. If the year-month-day part is not specified it takes the current year-month-day.

2.3 Time duration# TOC

There are two accepted syntaxes for expressing time duration.

```
[-][HH:]MM:SS[.m...]
```

HH expresses the number of hours, *MM* the number of minutes for a maximum of 2 digits, and *SS* the number of seconds for a maximum of 2 digits. The *m* at the end expresses decimal value for *SS*.

or

```
[-]S+[.m...]
```

S expresses the number of seconds, with the optional decimal part *m*.

In both expressions, the optional '-' indicates negative duration.

2.3.1 Examples# TOC

The following examples are all valid time duration:

'55'

    55 seconds

'12:03:45'

    12 hours, 03 minutes and 45 seconds

'23.189'

    23.189 seconds

2.4 Video size# TOC

Specify the size of the sourced video, it may be a string of the form *width*x*height*, or the name of a size abbreviation.

The following abbreviations are recognized:

'ntsc'

    720x480

'pal'

    720x576

'qntsc'

    352x240

'qpal'

352x288

'sntsc'

    640x480

'spal'

    768x576

'film'

    352x240

'ntsc-film'

    352x240

'sqcif'

    128x96

'qcif'

    176x144

'cif'

    352x288

'4cif'

    704x576

'16cif'

    1408x1152

'qqvga'

    160x120

'qvga'

    320x240

'vga'

640x480

'svga'

800x600

'xga'

1024x768

'uxga'

1600x1200

'qxga'

2048x1536

'sxga'

1280x1024

'qsxga'

2560x2048

'hsxga'

5120x4096

'wvga'

852x480

'wxga'

1366x768

'wsxga'

1600x1024

'wuxga'

1920x1200

'woxga'

2560x1600

'wqsxga'

3200x2048

'wquxga'

3840x2400

'whsxga'

6400x4096

'whuxga'

7680x4800

'cga'

320x200

'ega'

640x350

'hd480'

852x480

'hd720'

1280x720

'hd1080'

1920x1080

'2k'

2048x1080

'2kflat'

1998x1080

'2kscope'

2048x858

'4k'

4096x2160

'4kflat'

3996x2160

'4kscope'

4096x1716

'nhd'

640x360

'hqvga'

240x160

'wqvga'

400x240

'fwqvga'

432x240

'hvga'

480x320

'qhd'

960x540

2.5 Video rate# TOC

Specify the frame rate of a video, expressed as the number of frames generated per second. It has to be a string in the format *frame_rate_num*/*frame_rate_den*, an integer number, a float number or a valid video frame rate abbreviation.

The following abbreviations are recognized:

'ntsc'

30000/1001

'pal'

25/1

'qntsc'

30000/1001

'qpal'

25/1

'sntsc'

30000/1001

'spal'

25/1

'film'

24/1

'ntsc-film'

24000/1001

2.6 Ratio# TOC

A ratio can be expressed as an expression, or in the form *numerator*:*denominator*.

Note that a ratio with infinite (1/0) or negative value is considered valid, so you should check on the returned value if you want to exclude those values.

The undefined value can be expressed using the "0:0" string.

2.7 Color# TOC

It can be the name of a color as defined below (case insensitive match) or a [0x|#]RRGGBB[AA] sequence, possibly followed by @ and a string representing the alpha component.

The alpha component may be a string composed by "0x" followed by an hexadecimal number or a decimal number between 0.0 and 1.0, which represents the opacity value ('0x00' or '0.0' means completely transparent, '0xff' or '1.0' completely opaque). If the alpha component is not specified then '0xff' is assumed.

The string 'random' will result in a random color.

The following names of colors are recognized:

'AliceBlue'

    0xF0F8FF

'AntiqueWhite'

    0xFAEBD7

'Aqua'

    0x00FFFF

'Aquamarine'

    0x7FFFD4

'Azure'

    0xF0FFFF

'Beige'

    0xF5F5DC

'Bisque'

    0xFFE4C4

'Black'

    0x000000

'BlanchedAlmond'

    0xFFEBCD

'Blue'

    0x0000FF

'BlueViolet'

    0x8A2BE2

'Brown'

0xA52A2A

'BurlyWood'

0xDEB887

'CadetBlue'

0x5F9EA0

'Chartreuse'

0x7FFF00

'Chocolate'

0xD2691E

'Coral'

0xFF7F50

'CornflowerBlue'

0x6495ED

'Cornsilk'

0xFFF8DC

'Crimson'

0xDC143C

'Cyan'

0x00FFFF

'DarkBlue'

0x00008B

'DarkCyan'

0x008B8B

'DarkGoldenRod'

0xB8860B

'DarkGray'

0xA9A9A9

'DarkGreen'

0x006400

'DarkKhaki'

0xBDB76B

'DarkMagenta'

0x8B008B

'DarkOliveGreen'

0x556B2F

'Darkorange'

0xFF8C00

'DarkOrchid'

0x9932CC

'DarkRed'

0x8B0000

'DarkSalmon'

0xE9967A

'DarkSeaGreen'

0x8FBC8F

'DarkSlateBlue'

0x483D8B

'DarkSlateGray'

0x2F4F4F

'DarkTurquoise'

0x00CED1

'DarkViolet'

0x9400D3

'DeepPink'

0xFF1493

'DeepSkyBlue'

0x00BFFF

'DimGray'

0x696969

'DodgerBlue'

0x1E90FF

'FireBrick'

0xB22222

'FloralWhite'

0xFFFAF0

'ForestGreen'

0x228B22

'Fuchsia'

0xFF00FF

'Gainsboro'

0xDCDCDC

'GhostWhite'

0xF8F8FF

'Gold'

0xFFD700

'GoldenRod'

0xDAA520

'Gray'

0x808080

'Green'

0x008000

'GreenYellow'

0xADFF2F

'HoneyDew'

0xF0FFF0

'HotPink'

0xFF69B4

'IndianRed'

0xCD5C5C

'Indigo'

0x4B0082

'Ivory'

0xFFFFF0

'Khaki'

0xF0E68C

'Lavender'

0xE6E6FA

'LavenderBlush'

0xFFF0F5

'LawnGreen'

0x7CFC00

'LemonChiffon'

0xFFFACD

'LightBlue'

0xADD8E6

'LightCoral'

0xF08080

'LightCyan'

0xE0FFFF

'LightGoldenRodYellow'

0xFAFAD2

'LightGreen'

0x90EE90

'LightGrey'

0xD3D3D3

'LightPink'

0xFFB6C1

'LightSalmon'

0xFFA07A

'LightSeaGreen'

0x20B2AA

'LightSkyBlue'

0x87CEFA

'LightSlateGray'

0x778899

'LightSteelBlue'

0xB0C4DE

'LightYellow'

0xFFFFE0

'Lime'

0x00FF00

'LimeGreen'

0x32CD32

'Linen'

0xFAF0E6

'Magenta'

0xFF00FF

'Maroon'

0x800000

'MediumAquaMarine'

0x66CDAA

'MediumBlue'

0x0000CD

'MediumOrchid'

0xBA55D3

'MediumPurple'

0x9370D8

'MediumSeaGreen'

0x3CB371

'MediumSlateBlue'

0x7B68EE

'MediumSpringGreen'

0x00FA9A

'MediumTurquoise'

0x48D1CC

'MediumVioletRed'

0xC71585

'MidnightBlue'

0x191970

'MintCream'

0xF5FFFA

'MistyRose'

0xFFE4E1

'Moccasin'

0xFFE4B5

'NavajoWhite'

0xFFDEAD

'Navy'

0x000080

'OldLace'

0xFDF5E6

'Olive'

0x808000

'OliveDrab'

0x6B8E23

'Orange'

0xFFA500

'OrangeRed'

0xFF4500

'Orchid'

0xDA70D6

'PaleGoldenRod'

0xEEE8AA

'PaleGreen'

0x98FB98

'PaleTurquoise'

0xAFEEEE

'PaleVioletRed'

0xD87093

'PapayaWhip'

0xFFEFD5

'PeachPuff'

0xFFDAB9

'Peru'

0xCD853F

'Pink'

0xFFC0CB

'Plum'

0xDDA0DD

'PowderBlue'

0xB0E0E6

'Purple'

0x800080

'Red'

0xFF0000

'RosyBrown'

0xBC8F8F

'RoyalBlue'

0x4169E1

'SaddleBrown'

0x8B4513

'Salmon'

0xFA8072

'SandyBrown'

0xF4A460

'SeaGreen'

0x2E8B57

'SeaShell'

0xFFF5EE

'Sienna'

0xA0522D

'Silver'

0xC0C0C0

'SkyBlue'

0x87CEEB

'SlateBlue'

0x6A5ACD

'SlateGray'

0x708090

'Snow'

0xFFFAFA

'SpringGreen'

0x00FF7F

'SteelBlue'

0x4682B4

'Tan'

0xD2B48C

'Teal'

0x008080

'Thistle'

0xD8BFD8

'Tomato'

0xFF6347

'Turquoise'

0x40E0D0

'Violet'

0xEE82EE

'Wheat'

0xF5DEB3

'White'

0xFFFFFF

'WhiteSmoke'

0xF5F5F5

'Yellow'

0xFFFF00

'YellowGreen'

0x9ACD32

2.8 Channel Layout# TOC

A channel layout specifies the spatial disposition of the channels in a multi-channel audio stream. To specify a channel layout, FFmpeg makes use of a special syntax.

Individual channels are identified by an id, as given by the table below:

'FL'

front left

'FR'

front right

'FC'

    front center

'LFE'

    low frequency

'BL'

    back left

'BR'

    back right

'FLC'

    front left-of-center

'FRC'

    front right-of-center

'BC'

    back center

'SL'

    side left

'SR'

    side right

'TC'

    top center

'TFL'

    top front left

'TFC'

    top front center

'TFR'

top front right

'TBL'

top back left

'TBC'

top back center

'TBR'

top back right

'DL'

downmix left

'DR'

downmix right

'WL'

wide left

'WR'

wide right

'SDL'

surround direct left

'SDR'

surround direct right

'LFE2'

low frequency 2

Standard channel layout compositions can be specified by using the following identifiers:

'mono'

FC

'stereo'

FL+FR

`2.1`

FL+FR+LFE

`3.0`

FL+FR+FC

`3.0(back)`

FL+FR+BC

`4.0`

FL+FR+FC+BC

`quad`

FL+FR+BL+BR

`quad(side)`

FL+FR+SL+SR

`3.1`

FL+FR+FC+LFE

`5.0`

FL+FR+FC+BL+BR

`5.0(side)`

FL+FR+FC+SL+SR

`4.1`

FL+FR+FC+LFE+BC

`5.1`

FL+FR+FC+LFE+BL+BR

`5.1(side)`

FL+FR+FC+LFE+SL+SR

'6.0'

FL+FR+FC+BC+SL+SR

'6.0(front)'

FL+FR+FLC+FRC+SL+SR

'hexagonal'

FL+FR+FC+BL+BR+BC

'6.1'

FL+FR+FC+LFE+BC+SL+SR

'6.1'

FL+FR+FC+LFE+BL+BR+BC

'6.1(front)'

FL+FR+LFE+FLC+FRC+SL+SR

'7.0'

FL+FR+FC+BL+BR+SL+SR

'7.0(front)'

FL+FR+FC+FLC+FRC+SL+SR

'7.1'

FL+FR+FC+LFE+BL+BR+SL+SR

'7.1(wide)'

FL+FR+FC+LFE+BL+BR+FLC+FRC

'7.1(wide-side)'

FL+FR+FC+LFE+FLC+FRC+SL+SR

'octagonal'

FL+FR+FC+BL+BR+BC+SL+SR

'`downmix`'

DL+DR

A custom channel layout can be specified as a sequence of terms, separated by '+' or '|'. Each term can be:

- the name of a standard channel layout (e.g. '`mono`', '`stereo`', '`4.0`', '`quad`', '`5.0`', etc.)
- the name of a single channel (e.g. '`FL`', '`FR`', '`FC`', '`LFE`', etc.)
- a number of channels, in decimal, optionally followed by '`c`', yielding the default channel layout for that number of channels (see the function `av_get_default_channel_layout`)
- a channel layout mask, in hexadecimal starting with "0x" (see the `AV_CH_*` macros in `libavutil/channel_layout.h`.

Starting from libavutil version 53 the trailing character "c" to specify a number of channels will be required, while a channel layout mask could also be specified as a decimal number (if and only if not followed by "c").

See also the function `av_get_channel_layout` defined in `libavutil/channel_layout.h`.

3 Expression Evaluation# TOC

When evaluating an arithmetic expression, FFmpeg uses an internal formula evaluator, implemented through the `libavutil/eval.h` interface.

An expression may contain unary, binary operators, constants, and functions.

Two expressions *expr1* and *expr2* can be combined to form another expression "*expr1*;*expr2*". *expr1* and *expr2* are evaluated in turn, and the new expression evaluates to the value of *expr2*.

The following binary operators are available: +, −, *, /, ^.

The following unary operators are available: +, −.

The following functions are available:

`abs(x)`

Compute absolute value of *x*.

`acos(x)`

Compute arccosine of *x*.

`asin(x)`

Compute arcsine of *x*.

`atan(x)`

Compute arctangent of *x*.

`between(x, min, max)`

Return 1 if *x* is greater than or equal to *min* and lesser than or equal to *max*, 0 otherwise.

`bitand(x, y)`
`bitor(x, y)`

Compute bitwise and/or operation on *x* and *y*.

The results of the evaluation of *x* and *y* are converted to integers before executing the bitwise operation.

Note that both the conversion to integer and the conversion back to floating point can lose precision. Beware of unexpected results for large numbers (usually 2^53 and larger).

`ceil(expr)`

Round the value of expression *expr* upwards to the nearest integer. For example, "ceil(1.5)" is "2.0".

`clip(x, min, max)`

Return the value of *x* clipped between *min* and *max*.

`cos(x)`

Compute cosine of *x*.

`cosh(x)`

Compute hyperbolic cosine of *x*.

`eq(x, y)`

Return 1 if *x* and *y* are equivalent, 0 otherwise.

`exp(x)`

Compute exponential of *x* (with base `e`, the Euler's number).

`floor(expr)`

Round the value of expression *expr* downwards to the nearest integer. For example, "floor(-1.5)" is "-2.0".

`gauss(x)`

   Compute Gauss function of x, corresponding to `exp(-x*x/2) / sqrt(2*PI)`.

`gcd(x, y)`

   Return the greatest common divisor of x and y. If both x and y are 0 or either or both are less than zero then behavior is undefined.

`gt(x, y)`

   Return 1 if x is greater than y, 0 otherwise.

`gte(x, y)`

   Return 1 if x is greater than or equal to y, 0 otherwise.

`hypot(x, y)`

   This function is similar to the C function with the same name; it returns "sqrt($x*x + y*y$)", the length of the hypotenuse of a right triangle with sides of length x and y, or the distance of the point (x, y) from the origin.

`if(x, y)`

   Evaluate x, and if the result is non-zero return the result of the evaluation of y, return 0 otherwise.

`if(x, y, z)`

   Evaluate x, and if the result is non-zero return the evaluation result of y, otherwise the evaluation result of z.

`ifnot(x, y)`

   Evaluate x, and if the result is zero return the result of the evaluation of y, return 0 otherwise.

`ifnot(x, y, z)`

   Evaluate x, and if the result is zero return the evaluation result of y, otherwise the evaluation result of z.

`isinf(x)`

   Return 1.0 if x is +/-INFINITY, 0.0 otherwise.

`isnan(x)`

   Return 1.0 if x is NAN, 0.0 otherwise.

```
ld(var)
```

Load the value of the internal variable with number *var*, which was previously stored with st(*var*, *expr*). The function returns the loaded value.

```
log(x)
```

Compute natural logarithm of *x*.

```
lt(x, y)
```

Return 1 if *x* is lesser than *y*, 0 otherwise.

```
lte(x, y)
```

Return 1 if *x* is lesser than or equal to *y*, 0 otherwise.

```
max(x, y)
```

Return the maximum between *x* and *y*.

```
min(x, y)
```

Return the maximum between *x* and *y*.

```
mod(x, y)
```

Compute the remainder of division of *x* by *y*.

```
not(expr)
```

Return 1.0 if *expr* is zero, 0.0 otherwise.

```
pow(x, y)
```

Compute the power of *x* elevated *y*, it is equivalent to "$(x)\wedge(y)$".

```
print(t)
print(t, l)
```

Print the value of expression *t* with loglevel *l*. If *l* is not specified then a default log level is used. Returns the value of the expression printed.

Prints t with loglevel l

```
random(x)
```

Return a pseudo random value between 0.0 and 1.0. *x* is the index of the internal variable which will be used to save the seed/state.

`root(expr, max)`

>   Find an input value for which the function represented by *expr* with argument *ld(0)* is 0 in the interval
>   0..*max*.

>   The expression in *expr* must denote a continuous function or the result is undefined.

>   *ld(0)* is used to represent the function input value, which means that the given expression will be
>   evaluated multiple times with various input values that the expression can access through `ld(0)`.
>   When the expression evaluates to 0 then the corresponding input value will be returned.

`sin(x)`

>   Compute sine of *x*.

`sinh(x)`

>   Compute hyperbolic sine of *x*.

`sqrt(expr)`

>   Compute the square root of *expr*. This is equivalent to "(*expr*)^.5".

`squish(x)`

>   Compute expression `1/(1 + exp(4*x))`.

`st(var, expr)`

>   Store the value of the expression *expr* in an internal variable. *var* specifies the number of the variable
>   where to store the value, and it is a value ranging from 0 to 9. The function returns the value stored in
>   the internal variable. Note, Variables are currently not shared between expressions.

`tan(x)`

>   Compute tangent of *x*.

`tanh(x)`

>   Compute hyperbolic tangent of *x*.

`taylor(expr, x)`
`taylor(expr, x, id)`

>   Evaluate a Taylor series at *x*, given an expression representing the `ld(id)`-th derivative of a
>   function at 0.

>   When the series does not converge the result is undefined.

ld(id) is used to represent the derivative order in *expr*, which means that the given expression will be evaluated multiple times with various input values that the expression can access through `ld(id)`. If *id* is not specified then 0 is assumed.

Note, when you have the derivatives at y instead of 0, `taylor(expr, x-y)` can be used.

`time(0)`

Return the current (wallclock) time in seconds.

`trunc(expr)`

Round the value of expression *expr* towards zero to the nearest integer. For example, "trunc(-1.5)" is "-1.0".

`while(cond, expr)`

Evaluate expression *expr* while the expression *cond* is non-zero, and returns the value of the last *expr* evaluation, or NAN if *cond* was always false.

The following constants are available:

`PI`

area of the unit disc, approximately 3.14

`E`

exp(1) (Euler's number), approximately 2.718

`PHI`

golden ratio (1+sqrt(5))/2, approximately 1.618

Assuming that an expression is considered "true" if it has a non-zero value, note that:

* works like AND

+ works like OR

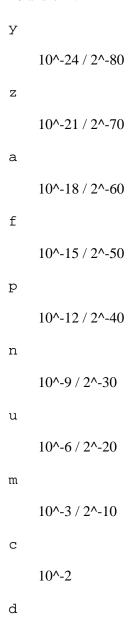For example the construct:

```
if (A AND B) then C
```

is equivalent to:

```
if(A*B, C)
```

In your C code, you can extend the list of unary and binary functions, and define recognized constants, so that they are available for your expressions.

The evaluator also recognizes the International System unit prefixes. If 'i' is appended after the prefix, binary prefixes are used, which are based on powers of 1024 instead of powers of 1000. The 'B' postfix multiplies the value by 8, and can be appended after a unit prefix or used alone. This allows using for example 'KB', 'MiB', 'G' and 'B' as number postfix.

The list of available International System prefixes follows, with indication of the corresponding powers of 10 and of 2.

y

    10^-24 / 2^-80

z

    10^-21 / 2^-70

a

    10^-18 / 2^-60

f

    10^-15 / 2^-50

p

    10^-12 / 2^-40

n

    10^-9 / 2^-30

u

    10^-6 / 2^-20

m

    10^-3 / 2^-10

c

    10^-2

d

10^-1

h

10^2

k

10^3 / 2^10

K

10^3 / 2^10

M

10^6 / 2^20

G

10^9 / 2^30

T

10^12 / 2^40

P

10^15 / 2^40

E

10^18 / 2^50

Z

10^21 / 2^60

Y

10^24 / 2^70

4 OpenCL Options# TOC

When FFmpeg is configured with `--enable-opencl`, it is possible to set the options for the global OpenCL context.

The list of supported options follows:

`build_options`

> Set build options used to compile the registered kernels.
>
> See reference "OpenCL Specification Version: 1.2 chapter 5.6.4".

`platform_idx`

> Select the index of the platform to run OpenCL code.
>
> The specified index must be one of the indexes in the device list which can be obtained with `ffmpeg -opencl_bench` or `av_opencl_get_device_list()`.

`device_idx`

> Select the index of the device used to run OpenCL code.
>
> The specified index must be one of the indexes in the device list which can be obtained with `ffmpeg -opencl_bench` or `av_opencl_get_device_list()`.

5 See Also# TOC

ffmpeg, ffplay, ffprobe, ffserver, libavutil

6 Authors# TOC

The FFmpeg developers.

For details about the authorship, see the Git history of the project (git://source.ffmpeg.org/ffmpeg), e.g. by typing the command `git log` in the FFmpeg source directory, or browsing the online repository at http://source.ffmpeg.org.

Maintainers for the specific components are listed in the file `MAINTAINERS` in the source code tree.

This document was generated using *makeinfo*.