NAME

mluCapabilities: mluFindDeviceByName mluFindJackByName mluFindPathByName mluFindXcodeByName mluFindXcodePipeByName mluFindJackByDirection mluFindFirstInputJack mluFindFirstOutputJack mluFindPathFromJack mluFindPathToJack − convenient functions for accessing the ml capabilites

SYNOPSIS

#include <ML/ml.h>
#include <ML/mlu.h>

MLstatus
mluFindDeviceByName(MLint64 sysId, const char* name,
MLint64* retDevId);

MLstatus
mluFindJackByName(MLint64 devId, const char* name,
MLint64* retDevId);

MLstatus
mluFindPathByName(MLint64 devId, const char* name,
MLint64* retDevId);

MLstatus
mluFindXcodeByName(MLint64 devId, const char* name,
MLint64* retDevId);

MLstatus
mluFindXcodePipeByName(MLint64 xcodeId, const char* name,
MLint64* retPipeId);

MLstatus
mluFindJackByDirection(MLint64 devId, MLint32 direction,
MLint64* retJackId);

MLstatus
mluFindFirstInputJack(MLint64 devId, MLint64* retJackId);

MLstatus
mluFindFirstOutputJack(MLint64 devId, MLint64* retJackId);

MLstatus
mluFindPathFromJack(MLint64 jackId,
MLint64* retPathId, MLint32* retPathAlignment);

MLstatus
mluFindPathToJack(MLint64 jackId,
MLint64* retPathId, MLint32* retPathAlignment);

PARAMETER

sysId	The ML system identifier. Currently, the only valid sysId is ML_SYSTEM_LOCALHOST.
devId	The ML device identifier.
xcodeId	The ML transcoder identifier.
jackId	The ML jack identifier.
name	pointer to a character string corresponding to the ML_NAME of the object being searched.

retDevId	The resulting ML device identifier.
retJackId	The resulting ML jack identifier.
retPathId	The resulting ML path identifier.
retXcodeId	The resulting ML transcoder identifier.
retPipeId	The resulting ML pipe identifier.
retPathAlignment	The buffer alignment required for use with the returned path, may be NULL if you don't require alignment information.

DESCRIPTION

These routines provide a convenient way to search and interpret the ML capabilities tree. That tree is a hierarchy. So, to find a video path, first find a device, then a jack on that device, and then a path to/from that jack.

In cases where there is more than one choice, these routines return the first valid match.

EXAMPLE

This example opens a path from the first input jack on a named device.

```
MLint64 devId=0;
MLint64 jackId=0;
MLint64 pathId=0;
MLint32 memAlignment;
MLopenid openPath;

if( mluFindDeviceByName( ML_SYSTEM_LOCALHOST, desiredDeviceName,
                &devId ))
 {
  fprintf(stderr, "Cannot find device.0);
  return -1;
 }

if( mluFindFirstInputJack( devId, &jackId ))
 {
  fprintf(stderr, "Cannot find a suitable input jack.\n");
  return -1;
 }

if( mluFindPathFromJack( jackId, &pathId, &memAlignment ))
 {
  fprintf(stderr, "Cannot find a path from jack\n");
  return -1;
 }

if( mlOpen( pathId, NULL, &openPath ) )
 {
  fprintf(stderr, "Cannot open path.\n");
  return -1;
 }
```

DIAGNOSTICS

These functions return one of the following:

ML_STATUS_NO_ERROR
: The operation was successful.

ML_STATUS_INVALID_ID
: The specifi ed ml id value is invalid.

ML_STATUS_ARGUMENT
: At least one of the arguments is invalid.

SEE ALSO

mlGetCapabilities(3dm), mlOpen(3dm), mlIntro(3dm).

NAME

mluDefaults: mluComputePathParamsFromTiming − compute path parameters from timing

SYNOPSIS

#include <ML/ml.h>
#include <ML/mlu.h>

MLstatus
mluComputePathParamsFromTiming(MLint32 timing, MLpv* pv, MLint32 flags);

PARAMETER

timing A video timing defi ned by one of the ML_VIDEO_TIMING_INT32 enumerations.

pv A MLpv list consisting of control values described below. The values of the controls will be set according to the specifi ed timing or format.

flags Reserved for future use - must be zero.

bytesPerPixelNumRet, bytesPerPixelDenomRet
Return values whose ratio gives the number of bytes per pixel.

DESCRIPTION

Use mluComputePathDefaultsFromTiming to get typical values for the following path parameters:

ML_VIDEO_WIDTH_INT32,
ML_VIDEO_HEIGHT_F1_INT32,
ML_VIDEO_HEIGHT_F2_INT32,
ML_VIDEO_START_X_INT32,
ML_VIDEO_START_Y_F1_INT32,
ML_VIDEO_START_Y_F2_INT32,
ML_VIDEO_DOMINANCE_INT32,

ML_IMAGE_WIDTH_INT32,
ML_IMAGE_HEIGHT_1_INT32,
ML_IMAGE_HEIGHT_2_INT32,

ML_IMAGE_ROW_BYTES_INT32,
ML_IMAGE_SKIP_PIXELS_INT32,
ML_IMAGE_SKIP_ROWS_INT32,

ML_IMAGE_ORIENTATION_INT32,
ML_IMAGE_TEMPORAL_SAMPLING_INT32,
ML_IMAGE_INTERLEAVE_MODE_INT32

The MLpv list pv must consist of one or more of these parameters. The values that are assigned to the list are derived from the value of timing passed to mluComputePathParamsFromTiming.

DIAGNOSTICS

This function returns one of the following:

ML_STATUS_NO_ERROR
The function returned the requested result.

ML_STATUS_INVALID_VALUE
The timing was not recognized.

ML_STATUS_INVALID_PARAMETER
An invalid parameter was passed.

SEE ALSO

mlImageParameters(3dm), mlVideoParameters(3dm).

NAME

mluImageBufferSize: mluGetImageBufferSize, mluComputeImageBufferSize − calculate a memory buffer size based on the image parameters

SYNOPSIS

#include <ML/ml.h>
#include <ML/mlu.h>

MLstatus
mluGetImageBufferSize(MLopenid openPathOrPipe, MLint32* bufferSize);

MLstatus
mluComputeImageBufferSize(MLpv* imageParams, MLint32* bufferSize);

PARAMETER

openPathOrPipe	An open device handle.
imageParams	A list of parameters describing the image format.
bufferSize	The computed size of the image buffer.

DESCRIPTION

Use mluGetImageBufferSize to obtain the size of a memory buffer required to store an image defined by the controls of open path or pipe, *openPathOrPipe*.

Use mluComputeImageBufferSize to obtain the size of a memory buffer required to store an image defined by the *imageParams*. The function expects *imageParams* to contain values for the following image parameters:

ML_IMAGE_CODING_INT32,
ML_IMAGE_HEIGHT_INT32,
ML_IMAGE_WIDTH_INT32,
ML_IMAGE_COLORSPACE_INT32,
ML_IMAGE_PACKING_INT32,
ML_IMAGE_SAMPLING_INT32,
ML_IMAGE_INTERLEAVE_MODE_INT32,
ML_IMAGE_SKIP_ROWS_INT32,
ML_IMAGE_SKIP_PIXELS_INT32,
ML_IMAGE_ROW_BYTES_INT32

The MLpv list must consist of all the parameters. If any parameter is omitted, ML_INVALID_PARAMETER is returned upon exit.

NOTES

If ML_IMAGE_INTERLEAVE_MODE_INT32 parameter is set to ML_INTERLEAVE_MODE_SINGLE_FIELD, both functions return the size of a memory buffer required to store larger of two fields.

If ML_IMAGE_CODING_INT32 parameter indicates compression scheme, both functions will try to calculate the worse case scenario buffer size required for a given compression type.

EXAMPLE

The example demonstrates use of mluComputeImageBufferSize function.

MLint32 buffSize;
MLpv pv[12];

```
pv[0].param = ML_IMAGE_CODING_INT32;
pv[0].param = ML_CODING_UNCOMPRESSED;
pv[1].param = ML_IMAGE_COLORSPACE_INT32;
pv[1].value.int32 = a_colorspace[c];
pv[2].param = ML_IMAGE_PACKING_INT32;
pv[2].value.int32 = a_packing[p];
pv[3].param = ML_IMAGE_SAMPLING_INT32;
pv[3].value.int32 = a_sampling[s];
pv[4].param = ML_IMAGE_WIDTH_INT32;
pv[4].value.int32 = width;
pv[5].param = ML_IMAGE_HEIGHT_1_INT32;
pv[5].value.int32 = height_f1;
pv[6].param = ML_IMAGE_HEIGHT_2_INT32;
pv[6].value.int32 = height_f2;
pv[7].param = ML_IMAGE_INTERLEAVE_MODE_INT32;
pv[7].value.int32 = ML_INTERLEAVE_MODE_INTERLEAVED;
pv[8].param = ML_IMAGE_SKIP_ROWS_INT32;
pv[8].value.int32 = 0;
pv[9].param = ML_IMAGE_SKIP_PIXELS_INT32;
pv[9].value.int32 = 0;
pv[10].param = ML_IMAGE_ROW_BYTES_INT32;
pv[10].value.int32 = 0;
pv[11].param = ML_END;

if (mluComputeImageBufferSize(pv, &buffSize) != ML_STATUS_NO_ERROR) {
        fprintf(stderr, "mluComputeImageBufferSize failed0);
        return -1;
}
```

DIAGNOSTICS

The functions return one of the following:

ML_STATUS_NO_ERROR
: The function returned the requested result.

ML_STATUS_INVALID_VALUE
: One of the passed parameters had a wrong value.

ML_STATUS_INVALID_PARAMETER
: Not enough parameters were included in MLpv list, or the device does not support one of the required controls.

ML_STATUS_INVALID_ID
: The openPathOrPipe handle was invalid.

SEE ALSO

mlOpen(3dm), mlIntro(3dm).

NAME

mluPv: mluPvPrintMsg − print a MLpv parameter list

SYNOPSIS

#include <ML/ml.h>
#include <ML/mlu.h>

MLstatus
mluPvPrintMsg(MLint64 deviceId, MLpv* params);

PARAMETER

deviceId	A device identifier (either a static id obtained by calling getCapabilties, or an open id obtained by calling mlOpen).
params	a MLpv parameter list.

DESCRIPTION

Use to print a parameter list as it would be interpreted by the specified device.

NOTES

DIAGNOSTICS

This function returns one of the following:

ML_STATUS_NO_ERROR

ML_STATUS_INVALID_ID
: The specified device id is invalid

ML_STATUS_INVALID_VALUE
: One of the parameters in the list has an invalid value

ML_STATUS_INVALID_PARAMETER
: One of the parameters in the list is not recognized by that device

ML_STATUS_INVALID_ARGUMENT
: The mlPV list is invalid (perhaps a NULL pointer?).

SEE ALSO

mlParameters(3dm), mlPvString(3dm), mlIntro(3dm).

NAME

mluSizes: mluGetImageBufferSize, mluComputeImageBufferSize, mluGetImagePixelSixe, mluComputeImagePixelSize, mluGetAudioFrameSize, mluComputeAudioFrameSize − get or compute sizes of ML media

SYNOPSIS

#include <ML/ml.h>
#include <ML/mlu.h>

MLstatus
mluGetImageBufferSize(MLopenid openPathOrPipe, MLint32* size);

MLstatus
mluComputeImageBufferSize(MLpv* params, MLint32* size);

MLstatus
mluGetImagePixelSize(MLopenid openPathOrPipe,
MLint32* numerator, MLint32* denominator);

MLstatus
mluComputeImagePixelSize(MLpv* params,
MLint32* numerator, MLint32* denominator);

MLstatus
mluGetAudioFrameSize(MLopenid openPathOrPipe, MLint32* size);

MLstatus
mluComputeAudioFrameSize(MLpv* params, MLint32* size);

PARAMETER

openPathOrPipe	An open device handle.
params	A list of parameters describing the image or audio data.
size	The computed size.
numerator	The numerator of the computed size fraction.
denominator	The denominator of the computed size fraction.

DESCRIPTION

Use mluGetImageBufferSize to obtain the worst-case size of a memory buffer required to store an image defi ned by the controls of open path or pipe.

Use mluComputeImageBufferSize to compute the worst-case size of a memory buffer required to store an image defi ned by the *params*. The *params* list must contain the parameter:

ML_IMAGE_CODING_INT32

If that is set to an uncompressed coding, then the list must contain:

ML_IMAGE_HEIGHT_INT32 (or ML_IMAGE_HEIGHT_1_INT32)
ML_IMAGE_HEIGHT_2_INT32
ML_IMAGE_WIDTH_INT32,
ML_IMAGE_PACKING_INT32,
ML_IMAGE_SAMPLING_INT32,
ML_IMAGE_INTERLEAVE_MODE_INT32,

In addition, the list may optionally contain the following (if not included, these are assumed to have value 0):

ML_IMAGE_SKIP_ROWS_INT32,
ML_IMAGE_SKIP_PIXELS_INT32,
ML_IMAGE_ROW_BYTES_INT32

If any required parameter is omitted, ML_INVALID_PARAMETER is returned upon exit. Additional parameters may be included, and will be ignored.

Use mluGetImagePixelSize to obtain the number of bytes required to store a pixel as defined by the current control settings on the open path or pipe. Note that the number of bytes per pixel is returned as a fractional number (*numerator*/*denominator*) in units of bytes/pixel. A single scanline of pixels should be an exact mutiple of **numerator** bytes in length and is an exact mutiple of *denominator* pixels in length. Notice that *denominator* need not be one, and that the ratio *numerator*/*denominator* need not be expressed in reduced form (i.e. may have common factors.) This function will fail (returning ML_STATUS_INVALID_VALUE) if the image coding is not uncompressed.

Use mluComputeImagePixelSize compute the number of bytes required to store a pixel defined by the *params*. The function expects *params* to contain values for the following image parameters:

ML_IMAGE_PACKING_INT32,
ML_IMAGE_SAMPLING_INT32.

The MLpv list must consist of all the parameters. If any parameter is omitted, ML_INVALID_PARAMETER is returned upon exit. Additional parameters may be included, and will be ignored.

Use mluGetAudioFrameSize to obtain the worst-case size of a single frame of audio data. (You could multiply this by the number of frames per buffer to obtain the total buffer size).

Use mluComputeAudioFrameSize to compute the worst-case size of a single frame of audio data. The function expects *params* to contain values for the following audio parameters:

ML_AUDIO_FORMAT_INT32,
ML_AUDIO_CHANNELS_INT32.

The MLpv list must consist of all the parameters. If any parameter is omitted, ML_INVALID_PARAMETER is returned upon exit. Additional parameters may be included, and will be ignored.

EXAMPLE

The example demonstrates use of mluComputeImageBufferSize function.

```
MLint32 buffSize;
MLpv pv[12];

pv[0].param = ML_IMAGE_CODING_INT32;
pv[0].param = ML_CODING_UNCOMPRESSED;
pv[1].param = ML_IMAGE_COLORSPACE_INT32;
pv[1].value.int32 = a_colorspace[c];
pv[2].param = ML_IMAGE_PACKING_INT32;
pv[2].value.int32 = a_packing[p];
pv[3].param = ML_IMAGE_SAMPLING_INT32;
pv[3].value.int32 = a_sampling[s];
pv[4].param = ML_IMAGE_WIDTH_INT32;
pv[4].value.int32 = width;
pv[5].param = ML_IMAGE_HEIGHT_1_INT32;
pv[5].value.int32 = height_f1;
```

```
pv[6].param = ML_IMAGE_HEIGHT_2_INT32;
pv[6].value.int32 = height_f2;
pv[7].param = ML_IMAGE_INTERLEAVE_MODE_INT32;
pv[7].value.int32 = ML_INTERLEAVE_MODE_INTERLEAVED;
pv[8].param = ML_IMAGE_SKIP_ROWS_INT32;
pv[8].value.int32 = 0;
pv[9].param = ML_IMAGE_SKIP_PIXELS_INT32;
pv[9].value.int32 = 0;
pv[10].param = ML_IMAGE_ROW_BYTES_INT32;
pv[10].value.int32 = 0;
pv[11].param = ML_END;

if (mluComputeImageBufferSize(pv, &size) != ML_STATUS_NO_ERROR) {
        fprintf(stderr, "mluComputeImageBufferSize failed0);
        return -1;
}
```

DIAGNOSTICS

The functions return one of the following:

ML_STATUS_NO_ERROR
: The function returned the requested result.

ML_STATUS_INVALID_VALUE
: One of the required parameters had an invalid value.

ML_STATUS_INVALID_PARAMETER
: Not enough parameters were included in MLpv list, or the device does not support one of the required controls.

ML_STATUS_INVALID_ID
: The openPathOrPipe handle was invalid.

SEE ALSO

mlOpen(3dm), mlIntro(3dm).

NAME

mluTCAddTC, mluTCAddFrames − digital media timecode mathematics

SYNOPSIS

#include <ML/mlu.h>

MLstatus mluTCAddTC
(MLUtimecode * result,
const MLUtimecode *s1,
const MLUtimecode *s2,
int *overflowunderflow)

MLstatus mluTCAddFrames
(MLUtimecode * result,
const MLUtimecode *s1,
int frames,
int *overflowunderflow)

TYPES

MLUtimecode A structure containing a representation of SMPTE time code on which certain mathematical and utility functions can be performed. Can be used with: **mluTCAddTC(3dm), mluTCAddFrames(3dm), mluTCToString(3dm), mluTCFromSeconds(3dm), mluTCToSeconds(3dm), mluTCFromString(3dm), mluTCFramesPerDay(3dm)** , and **mluTCFramesBetween(3dm).** See also **MLUtimecode(3dm).**

ARGUMENTS

result The result of the addition operation. Note that this must be a valid pointer, and should not point to the same *MLUtimecode* as *s1* or *s2*.

s1, s2 The timecode operand(s) to be used in the addition.

frames The number of frames to add to the operand *s1*.

underflowoverflow An optional argument so that the user of the library can tell if an underflow or overflow condition occurred. If the app doesn't care about overflow/underflow, it should pass in NULL. Otherwise, *underflowoverflow* will be set to 0 (normal), a positive number (overflow), or a negative number (underflow).

DESCRIPTION

mluTCAddTC adds operand *s1* to operand *s2*. It returns the result of the addition in *result*. The tc_type of *result* will be the same as the tc_type of the operands. It is required that the tc_type of *s1* be the same as the tc_type of *s2*. See **NOTES** for information on adding timecodes of differing tc_type. **mluTCAddTC** will return ML_STATUS_INVALID_ARGUMENT if the addition failed. See **RETURN VALUE**.

mluTCAddFrames adds *frames* video frames to operand *s1* and returns the result in *result*. The tc_type of *result* will be the same tc_type as *s1*. It is acceptable for *frames* to be a negative value, but overflowunderflow from **mluTCAddFrames** should be checked to verify that there was no underflow condition.

RETURN VALUE

If a **MLUtimecode** operand (*s1* or *s2*) contains an illegal timecode value (e.g., a negative entry, invalid frame number, these functions will return MLU_STATUS_INVALID_ARGUMENT, and the contents of *result* will be undefi ned.

If the result of the addition overflows or underflows the 24 hour period, these functions will return MLU_STATUS_NO_ERROR, and the contents of *result* will have wrapped to a 24 hour clock. The *overflowunderflow* parameter will have been set, however, on an underflow or overflow condition.

mluTCAddTC and **mluTCAddFrames** return MLU_STATUS_NO_ERROR upon successful completion of the addition operation.

NOTES

In order for **MLUtimecode**'s of differing *tc_type* (e.g., MLU_TC_30_ND and MLU_TC_2997_4FIELD_DROP) to be added, they must first be converted to either frames or seconds, and added as either frames or seconds, as appropriate for the situation. Note that when adding **MLUtimecode**'s with different tc_type's, different results may be obtained by adding them as seconds or as frames--which is why **mluTCAddTC** does not allow two **MLUtimecode**'s of different tc_type's to be added.

SEE ALSO

mluTCFramesBetween(3dm), mluTCFramesPerDay(3dm), mluTCFromSeconds(3dm), mluTCFromString(3dm), mluTCFromString(3dm), mluTCToSeconds(3dm).

NAME

mluTCFramesBetween − Digital Media timecode mathematics

SYNOPSIS

#include <ML/mlu.h>

MLstatus mluTCFramesBetween
(int *result,
const MLUtimecode *a,
const MLUtimecode *b)

TYPES

MLUtimecode A structure containing a representation of SMPTE time code on which certain mathematical and utility functions can be performed. Can be used with: **mluTCAddTC(3dm)**, **mluTCAddFrames(3dm)**, **mluTCToString(3dm)**, **mluTCFromSeconds(3dm)**, **mluTCToSeconds(3dm)**, **mluTCFromString(3dm)**, **mluTCFramesPerDay(3dm)**, and **mluTCFramesBetween(3dm)**. See also **MLUtimecode(3dm)**.

ARGUMENTS

result The result of calculating the difference (b-a), measured in frames.

a, b The two operands of the differencing operation.

DESCRIPTION

dmFramesBetween is used to determine the number of frames between two timecodes. It returns (b - a), as measured in frames. The result of the difference operation may be negative; this is *not* an error condition.

Note that the operands *a* and *b* must have the same *tc_type* as each other.

RETURN VALUE

If a **MLUtimecode** operand (*a* or *b*) contains an illegal timecode value (e.g., a negative entry, invalid frame number, etc.), these functions will return ML_STATUS_INVALID_ARGUMENT, and the contents of *result* will be undefined.

If the *tc_type*'s of *a* and *b* do not match, **mluTCFramesBetween** will return ML_STATUS_INVALID_ARGUMENT, and the contents of *result* will be undefined.

mluTCFramesBetween returns ML_STATUS_NO_ERROR upon successful completion of the subtraction operation.

SEE ALSO

mluTCFramesBetween(3dm), mluTCFramesPerDay(3dm), mluTCFromSeconds(3dm), mluTCFromString(3dm), mluTCFromString(3dm), mluTCToSeconds(3dm).

NAME

mluTCFramesPerDay − Digital Media timecode mathematics

SYNOPSIS

#include <ML/mlu.h>

int mluTCFramesPerDay(const int tc_type)

TYPES

MLUtimecode A structure containing a representation of SMPTE time code on which certain mathematical and utility functions can be performed. Can be used with: **mluTCAddTC(3dm)** , **mluTCAddFrames(3dm)** , **mluTCToString(3dm)** , **mluTCFromSeconds(3dm)** , **mluTCToSeconds(3dm)** , **mluTCFromString(3dm)** , **mluTCFramesPerDay(3dm)** , and **mluTCFramesBetween(3dm)** See also MLUtimecode(3dm)

ARGUMENTS

tc_type The *tc_type* for which the number of frames in a 24 hour period is desired. See **tc_type(3dm)**

DESCRIPTION

mluTCFramesPerDay returns the number of frames that occur during a 24-hour day, in the requested timecode format, as given by *tc_type*. See **3tc_type(3dm)**

SEE ALSO

mluTCFramesBetween(3dm), mluTCFramesPerDay(3dm), mluTCFromSeconds(3dm), mluTCFromString(3dm), mluTCFromString(3dm), mluTCToSeconds(3dm).

NAME

mluTCToSeconds, mluTCFromSeconds − digital media timecode mathematics

SYNOPSIS

#include <ML/mlu.h>

MLstatus mluTCToSeconds
(const MLUtimecode *tc,
double *seconds)

MLstatus mluTCFromSeconds
(MLUtimecode * result,
const int tc_type,
const double seconds)

TYPES

MLUtimecode A structure containing a representation of SMPTE time code on which certain mathematical and utility functions can be performed. Can be used with: **mluTCAddTC(3dm)** , **mluTCAddFrames(3dm)** , **mluTCToString(3dm)** , **mluTCFromSeconds(3dm)** , **mluTCToSeconds(3dm)** , **mluTCFromString(3dm)** , **mluTCFramesPerDay(3dm)** , and **mluTCFramesBetween(3dm)** See also **MLUtimecode(3dm)**

ARGUMENTS

result The result of converting fractional seconds past midnight into a timecode value.

tc The timecode value to convert into seconds.

seconds The number of fractional seconds past midnight to convert, or the result of converting a **MLUtimecode** into fractional seconds past midnight.

tc_type When converting from fractional seconds to a **MLUtimecode**, the timecode format into which the seconds should be converted. See **tc_type(3dm)**

DESCRIPTION

These functions provide a simple means of converting between fractional seconds past midnight and **MLUtimecode** timecodes.

When converting from seconds to timecode, a valid *tc_type* must be supplied (see **tc_type(3dm)**). The *tc_type* of *result* will be the same as the *tc_type* that is passed to **mluTCFromSeconds**.

For **mluTCFromSeconds**, the *seconds* value given will be rounded to the nearest frame.

In order to convert to and from seconds, these routines need to assume some timecode rate in frames per second. They will choose the rate given in the MLU_TC_RATE bits of the specified *tc_type*. For drop-frame timecode, this is **not** the rate of the underlying video signal, and over the long term there will be drift. See **tc_type(3dm)** for more information on this.

RETURN VALUE

If a **MLUtimecode** operand (*tc*) contains an illegal timecode value (e.g., a negative entry, invalid frame number, etc.), **mluTCToSeconds** will return ML_STATUS_INVALID_ARGUMENT, and the contents of *seconds* will be undefined.

If the number of seconds passed to **mluTCFromSeconds** is negative, or greater than the number of seconds in a 24-hour day, **mluTCFromSeconds** will return ML_STATUS_INVALID_ARGUMENT, and the contents of *result* will be undefined.

If an invalid *tc_type* value is passed in to **mluTCFromSeconds**, ML_STATUS_INVALID_ARGUMENT is returned.

mluTCToSeconds and **mluTCFromSeconds** return ML_STATUS_NO_ERROR upon successful completion.

SEE ALSO

mluTCFramesBetween(3dm), mluTCFramesPerDay(3dm), mluTCFromSeconds(3dm), mluTCFromString(3dm), mluTCFromString(3dm), mluTCToSeconds(3dm).

NAME

mluTCToString, mluTCFromString, MLUtimecode, tc_type − digital media timecode mathematics

SYNOPSIS

#include <ML/mlu.h>

MLstatus mluTCToString
(char * outstring,
const MLUtimecode *tc)

MLstatus mluTCFromString
(MLUtimecode * result,
const char * instring,
int tc_type)

TYPES

MLUtimecode A structure containing a representation of SMPTE time code on which certain mathematical and utility functions can be performed. Can be used with: **mluTCAddTC(3dm)**, **mluTCAddFrames(3dm)**, **mluTCToString(3dm)**, **mluTCFromSeconds(3dm)**, **mluTCToSeconds(3dm)**, **mluTCFromString(3dm)**, **mluTCFramesPerDay(3dm)**, and **mluTCFramesBetween(3dm)**.

ARGUMENTS

outstring The string created by converting *tc*.

tc The timecode operand to convert to a string.

result The result of converting *instring* to a **MLUtimecode**.

instring The string to convert into a **MLUtimecode**.

tc_type The timecode type which **mluTCFromString** should assume the string is in. See TC_TYPE below.

DESCRIPTION

These utility functions will convert between strings and **MLUtimecode**'s.

To convert from a string to a **MLUtimecode**, the string format must a colon-separated string (in the form "h:m:s:f"); if fields are missing (i.e., "01:01:04") the string will be interpreted by assuming that the missing fields are on the left, and are "0". Thus, for example, the string "2:14" will be interpreted as "0:0:2:14."

When converting from a **MLUtimecode** to a string, the string returned will be fully justified and contain all fields. (i.e., it will return a fully-justified "00:00:02:14").

TC_TYPE

The mluTC routines, depend on a proper setting of the tc_type field. Often this tc_type field appears as a member of a MLUtimecode passed to or from one of these routines.

The tc_type field tells these routines whether the maximum frame value is 25 or 30, whether 30 frame code is drop frame or not, and if so what kind of drop frame.

The tc_type field, defined in <mlutimecode.h> (which is included by <mlu.h>), consists of an or'ed together bitmask of fields indicating format (MLU_TC_FORMAT_...), timecode rate (MLU_TC_RATE_...), and dropframe status (MLU_TC_*DROP*), but most users will find it much easier just to use one of the pre-constructed, fully-qualified tokens:

MLU_TC_30_ND - non-drop-frame NTSC or M/PAL timecode
MLU_TC_2997_4FIELD_DROP - drop-frame NTSC timecode
MLU_TC_2997_8FIELD_DROP - drop-frame M/PAL timecode
MLU_TC_25_ND - PAL timecode (not M/PAL)
MLU_TC_24_ND - 24 frame per second film timecode
MLU_TC_60_ND - non-drop-frame 60 frame per second HDTV timecode (experiemental)
MLU_TC_5594_8FIELD_DROP - drop-frame 59.94 frame per second HDTV timecode (experiemental)

The most common tokens for US use are MLU_TC_24_ND, MLU_TC_2997_4FIELD_DROP and MLU_TC_30_ND. In Europe, MLU_TC_24_ND and MLU_TC_25_ND will be the most commonly used tokens. "NTSC" and "PAL" above really refer to any 525/60 signal and 625/50 signal, respectively.

Note on rates: the MLU_TC_RATE field within tc_type does not refer to the video signal's rate. It simply refers to whether "30 frame" timecode is drop frame or not. Non-drop frame code has exactly 30 frames per second, drop-frame code has exactly 29.97 frames per second over the full day. The actual video signal rate is neither requested nor required by mluTC and other timecode routines. For color NTSC and M/PAL signals, it happens to be 30000/1001 frames per second, which equals neither 30 nor 29.97.

Because of this, any mluTC routines (such as **mluTCToSeconds**) which go between a timecode value and real time will assume a timecode rate as given by MLU_TC_RATE, **not** as given by the actual video signal's frame rate. This means that over the long term, drop frame timecode will drift away from real time at the rate of about 2 frames per day.

RETURN VALUE

If a **MLUtimecode** operand (*tc*) contains an illegal timecode value (e.g., a negative entry, invalid frame number, etc.), **mluTCToString** will return ML_STATUS_INVALID_ARGUMENT, and the contents of *outstring* will be undefined.

If **mluTCFromString** is unable to interpret an input string *instring*, it will return ML_STATUS_INVALID_ARGUMENT, and the contents of *result* will be undefined.

SEE ALSO

mluTCAddFrames(3dm), mluTCAddTC(3dm), mluTCFramesBetween(3dm), mluTCFramesPerDay(3dm), mluTCFromSeconds(3dm), mluTCToSeconds(3dm).