# Synchronous Streaming via USB

Manual

Version 1.9

Vector Informatik GmbH, Ingersheimer Straße 24, D-70499 Stuttgart
Tel. +49 (0)711 8 06 70-0, Fax +49 (0) 711 8 06 70-5 55,
Email support@vector.com, Internet http://www.vector.com

# Contents

# 1 Overview

One of the main use cases for a MOST system is the transfer of audio data between different ECUs such as the audio disc player, digital signal processing units and amplifier. The majority of the bandwidth in a MOST ring is therefore reserved for the synchronous data area, which is divided into several synchronous channels carrying e.g. audio streams.

There are several ways to access the synchronous data area with CANoe/CANalyzer and the MOST interfaces VN2610 or VN2640 (see fig. 1):

- Line in/headphone out: Access to a single analog stereo audio channel
- S/PDIF in and out: Access to a single digital stereo audio channel
- CANoe/CANalyzer via USB 2.0: Access to all synchronous channels at the same time

More information on the first and second method can be found in the VN2610/VN2640 manual and the CANoe/CANalyzer Online Help (MOST Audio Window; CAPL reference). This document describes the use cases and infrastructure for 'synchronous streaming via USB'.

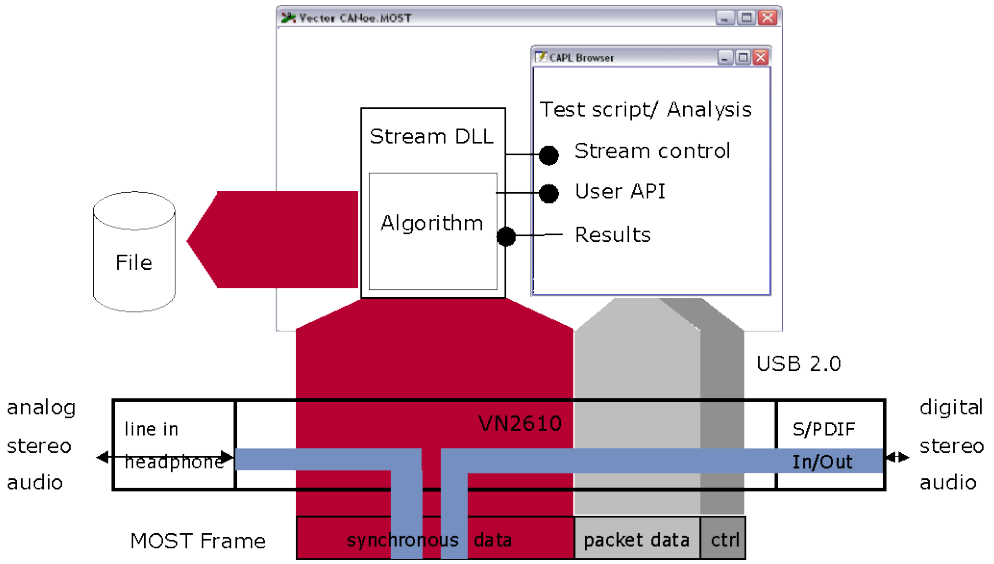



Fig. 1: Synchronous streaming architecture

## 1.1 Use cases

**Simple recording** of synchronous data from the MOST ring can be used for documentation purposes, for replaying identical data into a MOST ring or for offline analysis with external tools.

**Online analysis** of synchronous data can be used to detect test signals, to determine activity status on certain channels (mute detection) or to determine the delay between commands on the control channel and actions on the synchronous channels.

**CODEC** algorithms can be integrated into the streaming framework for both directions, either to test the CODEC implementations themselves or to enable tests with complex or encrypted data streams on the MOST ring.

**Replaying** synchronous data to a ring can be done using previously recorded data, reference signals or test signals generated online.

**Online modification in one Stream DLL** of received synchronous data and streaming back the modified data on the MOST ring (e.g. modifying the amplitude of an audio signal).

For each of the above use cases the streaming can be restricted to a single connection label indicated in the allocation table or to a group of them, allowing parallel processing of multiple streams.

For **low level tests**, any synchronous channel in the synchronous data area can be streamed, regardless of how channels are allocated in the allocation table.

Vector provides a number of universal streaming DLLs (see below). These cover a few of the above mentioned use cases, such as recording or playing back synchronous data. Other use cases can be addressed by using additional streaming DLLs with the streaming framework. These can either be developed by Vector according to customer specifications or by customers themselves with support by Vector.

Please contact Vector support for more information.

Phone: +49 711 80670-200

Web: http://www.vector.com/vi_support_en.html

## 1.2 Features

- Streaming is possible in both directions simultaneously from and to the MOST ring
- With VN2610 (MOST25) synchronous streaming via USB allows access to all synchronous channels in parallel. Depending on the configuration of the synchronous boundary control (SBC), this means up to 60 channels.
- The VN2640 interface (MOST150) can stream labels with up to 152 bytes per frame.
- Data from the MOST synchronous area is transmitted without modification to the PC and vice versa.
- Timestamps and status information can optionally be added to the streaming data when streaming from the MOST ring to the PC (MOST25 only).
- Processing of ".wav" audio files is optionally supported for streaming uncompressed audio data in both directions. This option is limited to streams with a width of two or four channels, corresponding to ".wav" files with one or two 16 bit channels.
- Latency times and buffer sizes can be customized for each use case. Simple recording where higher latencies can be tolerated may be optimized for low CPU load. Online analysis, e.g. for mute detection, typically requires results to be returned quickly.

## 1.3 Architecture

CANoe and CANalyzer are runtime environments for CAPL nodes. CAPL is a C-like programming language which is optimized for bus access and event driven applications. Processing a huge amount of continuous data such as audio streams is not a use case CAPL is prepared for, but CAPL can easily be supplemented by DLLs. Therefore the streaming architecture is set up as shown in fig. 1.

A CAPL node can access the control channel and the allocation table, and therefore can set up and control the streaming easily. The processing of the synchronous data itself is carried out in a ‘streaming DLL’.

The CAPL node first has to initialize the streaming DLL, e.g. by indicating how many channels are to be streamed. Each streaming DLL therefore offers a number of standard APIs (see chapter 3). Each DLL can offer an additional set of APIs, which may be needed to control a specific DLL (see chapter 4).

After initialization, the DLL continuously receives buffers for the synchronous data. When streaming data from the MOST bus, these buffers already contain the synchronous data which then can be processed within the DLL. When streaming to the MOST bus, the DLL has to fill the buffers with data.

The DLL has to indicate successful processing or filling of buffers to the streaming framework, which then either clears the buffers or transmits them to the MOST interface.

As shown in fig. 1, a simple streaming DLL might record the data to a file. With other DLLs, the processing of buffers may lead to results which the DLL can then return to the CAPL node by calling a callback function defined within the CAPL node.

## 1.4 Prerequisites

The following prerequisites must be met in order stream synchronous data:

**MOST25:**

- CANoe / CANalyzer 7.0 SP2 or higher
- **Note: For online modification in one Stream DLL, CANoe / CANalyzer 7.2 SP3 or higher** is needed
- MOST interface VN2610 connected to USB **2.0** port of the PC
- VN2610 driver version: **6.7** or higher

**MOST150:**

- CANoe / CANalyzer 7.6 SP3 or higher
- MOST interface VN2640 connected to USB **2.0** port of the PC
- VN2640 driver version: **7.9** or higher
- **For isochronous streaming:**
    - CANoe / CANalyzer 8.2 or higher
    - Appropriate source (tuner, dvd-player etc.) capable of streaming MPEG transport stream data over the isochronous channel

**General:**

- PC / Laptop:
    - Processor:
        - Recommended: Pentium 4 / 2,6 GHz
        - Minimum: Pentium III, 1 GHz
    - Memory (RAM):
        - Recommended: 1 GB
        - Minimum: 512 MB

# 2 Demo configurations

Vector has prepared a number of streaming DLLs with demo configurations that serve to show how the CAPL API can be used to control streaming. They can be found in the Demo folder of the CANoe/CANalyzer installation (CANoe Sample Configurations\MOST or CANalyzer Sample Configurations\MOST respectively).

## 2.1 Stream Player

The Stream Player configuration makes it possible to record data from all synchronous MOST channels as well as events from the control and asynchronous MOST channels. This requires a VN2610 MOST interface to be built into the ring and a measurement to be started.

The Stream Player configuration also makes it easy to play back and extract PCM audio signals from the recorded data, both during and after the measurement.

Note: This configuration is only available for CANoe.MOST and MOST25

### 2.1.1 Use cases

*Long time recording*

Please note that asynchronous data is saved in an uncompressed state. Sufficient disk space must be available for longer measurements.

With a 48 kHz system, the 60 synchronous MOST25 channels take up 48000 * (60 + 10) bytes per second = 3360000 bytes. This is about 12 GByte per hour. Data is not recorded during phases without a MOST signal (e.g. no lock).

In addition, a log file containing control and asynchronous channel events is written to the output folder.

To reduce the data volume, the number of synchronous channels to be recorded can be limited (see section 2.1.3). This setting is then applied to the entire measurement regardless of the actual SBC (Synchronous Bandwidth Control) value or the channel reservations in the allocation table. A sequential, adjacent set of channels (beginning with channel 0) is then recorded.

Example: If you know that no more than three stereo audio sources are active simultaneously in a given system, the number of channels to be included can be set to 3 x 4 bytes/frame = 12 bytes/frame = 3 quadlets/frame.

*Manual recording*

You can focus on critical areas of synchronous data transmission (e.g. source changes) by starting the recording manually. The adjustable pre-trigger time makes it possible to also record phases that precede the clicking of the Start button by a few seconds.

*Playback*

The Play button is used to play back recorded PCM audio data. To do this, two or four synchronous channels from which the PCM signal is to be extracted must be selected in the user interface. The data from these channels is played back regardless of the actual allocation or the reservations in the allocation table. In the case of non-PCM data, very loud audio interference may occur during source changes or ring interruptions for this reason.

The time needed to extract the audio signal will vary depending on the selected playback period (see section 2.1.3) and amount of data recorded.

*WAV export*

Areas of the synchronous communication can be exported as audio files in WAV format. This makes it possible to document audio interference separately from the memory-intensive recording of all synchronous channels.

### 2.1.2 Main panel

*Start button*

If ‘manual triggering’ is activated in the Options dialog (see 2.1.3), the Start button can be used to start and stop the recording while a measurement is in progress.

The button is disabled if the option to record the ‘entire measurement’ cycle is selected. In this case, data recording begins automatically when the measurement begins.

*Time bar*

The time bar in the upper area of the panel displays the chronological validity of the data. Segments in which no data is recorded are highlighted in red. Segments with valid synchronous data are highlighted in green. Changes in the allocation table are indicated by an ‘A’ above the time bar.

The left mouse button can be used to make a selection within the time bar. This selection is then used for the display in the allocation table and the playback of audio signals.

It is possible to enlarge an area of the time bar area by holding the right mouse button down. One right mouse-click restores the entire time bar.

*Allocation table*

This area is used to display the allocation table at the point in time selected in the time bar. If a synchronous channel is allocated, the channel label is displayed in the corresponding cell.

The left mouse button is used to select channels for audio signal playback. Selected channels are highlighted in gray. Double-clicking a channel label selects all channels allocated to that label.

Note: The channel labels are shown only for informational purposes. Channel numbers that are not allocated to a label can also be selected for audio playback.

*Play button*

Clicking the Play button starts the playback of audio signals via the PC's standard audio output device. The Play button is activated when two or four channels are selected in the allocation table above it.

*Save As button*

This is used to export the audio signal for the selected channels as a WAV file.

*Time bar (bottom)*

The time bar in the lower area of the panel shows an enlarged view of the time segment being played back.

### 2.1.3 Options dialog

Parameters for the recording can be adjusted before the measurement is started via the 'Options' button in the 'MOST Stream Player' panel.

The output folder is specified in the 'General' group.

The number of synchronous channels to be included in the recording can be selected using the 'Quadlets' selection box in the 'Recording' group. A quadlet contains four synchronous channels. A maximum of 15 quadlets can be selected, i.e. all 60 synchronous channels. This setting remains valid regardless of changes made to the SBC register for the entire measurement.

The recording can be started either for the entire measurement or manually. In the case of a manual start, it is possible to set a "pre-trigger time" of up to 30 seconds. The seconds (up to 30 seconds) occurring before the Start button is clicked are then saved to the hard disk as well.

The time period for the playback of audio signals is pre-set in the 'Playback' group.

The frame rate of your MOST system must be entered under 'System frequency'.

## 2.2 Stream to file

This demo is available for MOST25 (see folder StreamToFile) and MOST150 (folder StreamToFile_MOST150). Both versions apply the DLL MostSyncStrmToFile.dll to record synchronous data to hard disc. For MOST150 the DLL provides also the functionality to record MPEG transport stream data from an isochronous channel.

| **Note**: | For isochronous streaming CANoe 8.2 or higher is required. |
|---|---|

### 2.2.1 MOST25 demo

The configuration has a control panel where you can specify the file to which the recorded synchronous data is to be written. As files grow rapidly in size during recording, you can configure a file size limit, using MOST frames, seconds or minutes of recording time as the unit.

Activating a checkbox lets you increment the filename whenever the configured maximum file size is reached. If the checkbox is deactivated, recording stops when the maximum file size is reached.

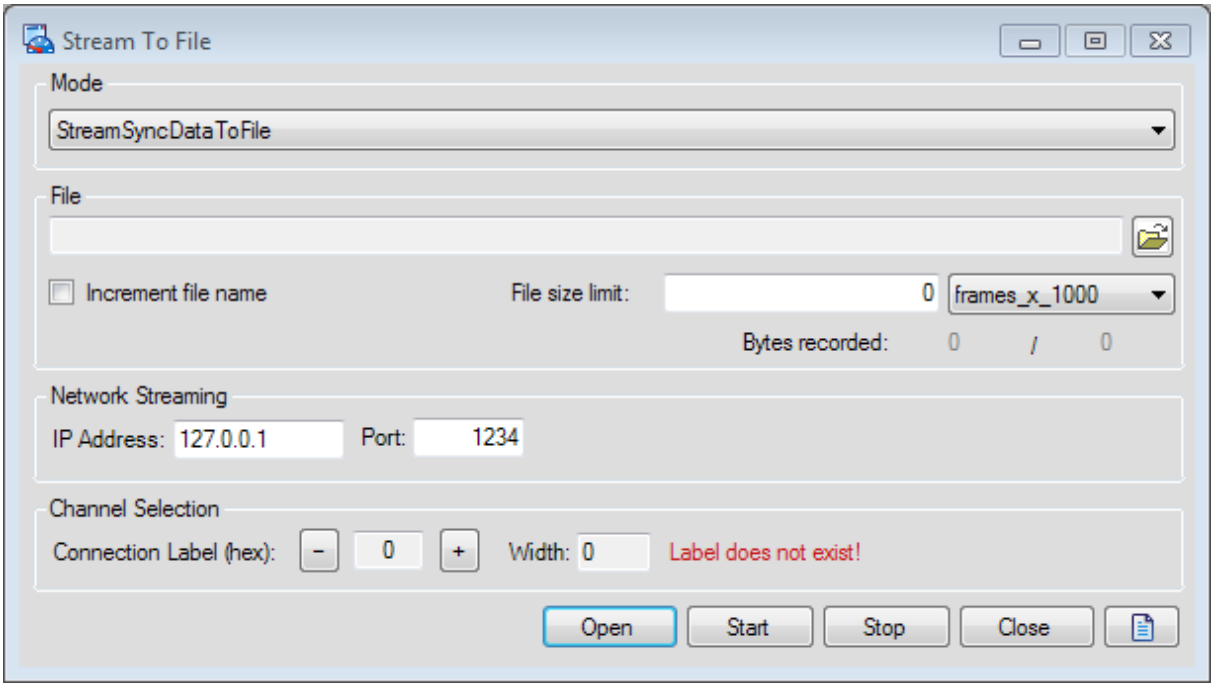
Once a target file and recording mode is chosen, you can select the channels to be recorded. You can do this either by selecting a connection label (see MOST Audio Routing Window) or by manually entering a number of channel indexes in the panel (MOST25 only).

Use the toolbar at the bottom of the panel to initialize (open) the streaming DLL, start and stop the recording, and release (close) the resources used by the DLL.

| | |
|---|---|
| **Note**: | The demo features an implementation in the StreamToFile.can CAPL node, which can handle various error situations. In chapter 6.1 a much simpler implementation is shown to demonstrate how little code is needed to use the DLL for recording streams. |

### 2.2.2 MOST150 demo

***Main panel controls***



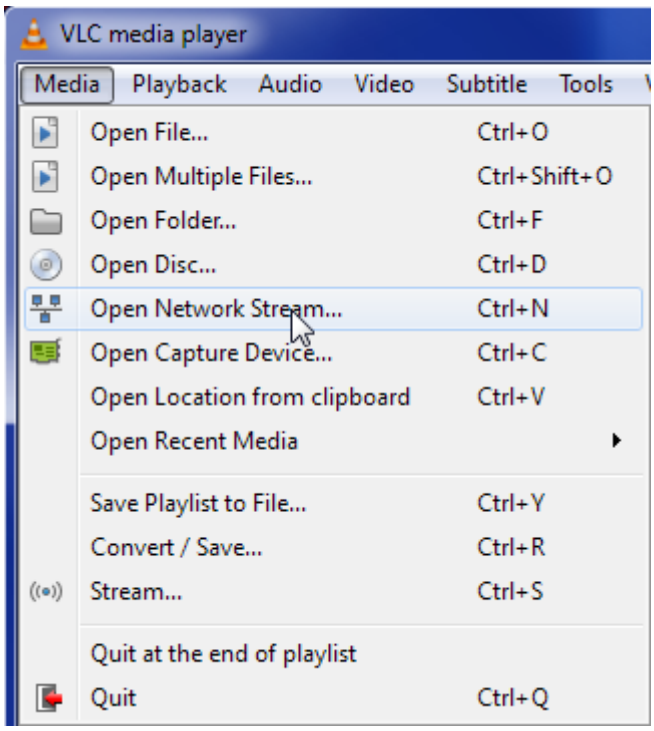

- Mode
  There are four streaming modes to choose between:
  - StreamSyncDataToFile: streams synchronous data to a binary or a wave file depending on file extension (.bin or .wav)
  - StreamIsoDataToFile: streams isochronous data to a binary or a MPEG transport stream file depending on file extension (.bin or .ts).
  - StreamIsoDataToNetwork: streams isochronous MPEG transport stream data to network using an UDP socket connection.
  - StreamIsoDataToFileAndNetwork: combines both previous modes

- File
  Here you can specify the file to which the streaming data is to be written. As files grow rapidly in size during recording, you can configure a file size limit, using MOST frames, seconds or minutes of recording time as the unit. Activating a checkbox lets you increment the filename whenever the configured maximum file size is reached. If the checkbox is deactivated, recording stops when the maximum file size is reached.
- Network Streaming
  Here you can specify the IP address and the port to which a MPEG transport stream is to be sent (if the corresponding mode is chosen).
- Channel selection
  Here you can select the connection label.

***Visualizing network streaming***

To visualize the isochronous MPEG transport stream data the free VLC media player can be used (http://www.videolan.org/vlc/). The following steps describe how to utilize the player.

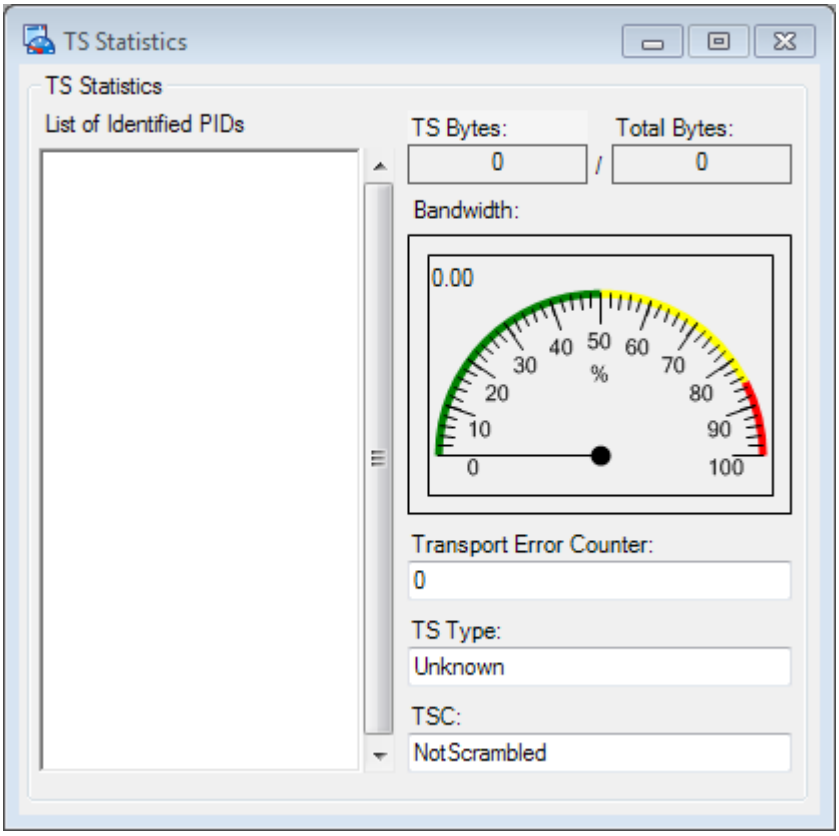
1) Start CANoe, load the demo, start the measurement and specify the required settings in the main panel
2) Start VLC media player and select “Open Network Stream…” from the “Media” menu



3) Enter the network URL in the appearing dialog
   if the player is running on a different PC as the demo:
   udp://<*IP address*>:<*port*>  e.g. udp://192.168.178.20:1234
   if the player is running on the same PC:
   udp://@:<*port*>  e.g. udp://@:1234
4) Push the “Play” button
5) Open and start streaming

### *MPEG transport stream statistics panel*

The Panel can be shown by selecting the “View”->”Panel”->”TS Statistics” menu entry.




The panel displays the following statistics of the MPEG transport stream:

- List of the occured PIDs and their lost frame counter
- Received number of bytes recognised as MPEG transport stream
- Received overall number of bytes on the connection label
- Calculated bandwith (TS Bytes/Total Bytes) * 100
- Transport error counter
- Type of the received MPEG transport stream (Video/Audio/VideoAndAudio)
- Scrabling information (NotScrambled/Unknown/EvenKey/OddKey)

## 2.3 Stream from file

A similar demo is provided for streaming files to the MOST ring.

Here you can select the file and the MOST ring channels to which the file contents are to be streamed, byte for byte.

If repeat mode is activated, the file is streamed continuously to the MOST ring.

This demo is available for MOST25 (see folder StreamFromFile) and MOST150 (folder StreamFromFile_MOST150).

## 2.4 Stream from RX to TX

The demo demonstrates how to use a streaming DLL for modifying an audio stream. Therefore data on synchronous channel is tapped (RX), processed in the DLL and then sent back (TX) to the ring. Available modifications are muting and volume change.

This configuration is only available for CANoe.MOST, MOST25 (see folder StreamRx2Tx) and MOST150 (see folder StreamRx2Tx_MOST150).

### 2.4.1 MOST25 demo

***RX to TX Streaming dialog panel***

*RX Stream*

Here you can choose the channels from which the synchronous data will be tapped.

*TX Stream*

Here you can choose the channels to which the modified synchronous data will be sent.

The state fields show the current state of the respective stream.

*Latency*

Here you can select the streaming latency. Please consider low latency causes higher CPU load.

*Volume*

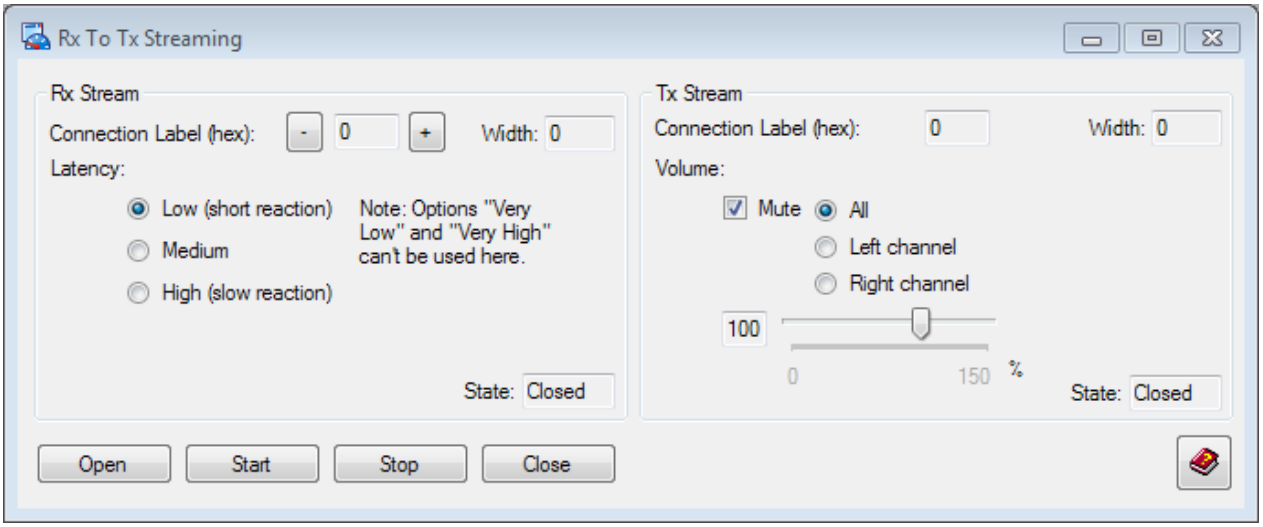
Here you can select which modification will be done. “Left channel” control, “Right channel” control and volume slider control are only available if 4 synchronous channels are allocated.

*Control*

Use the toolbar at the bottom of the panel to initialize (open) the streaming DLL, start and stop streaming, and release (close) resources used by the DLL.

### 2.4.2 MOST150 demo

***RX to TX Streaming dialog panel***




*RX Stream*

Here you can choose the label from which the synchronous data will be tapped and the streaming latency. Please consider low latency causes higher CPU load.

*TX Stream*

Here the connection label to which the modified synchronous data will be sent is shown. The label will be automatically allocated when the TX-stream is opened (s. chapter 3.3 MostSyncStrmOpen).

You can also select which modification on volume will be done. “Left channel” control, “Right channel” control and volume slider control are only available if the width of the allocated connection label is 4 bytes. The width depends on the width of the RX-stream.

*Control*

Use the toolbar at the bottom of the panel to initialize (open) the streaming DLL, start and stop streaming, and release (close) resources used by the DLL.

| | |
|---|---|
| **Note**: | If the streaming suffers from audio dropouts or artifacts please check the “Receive Latency” in the Vector Hardware Configuration. It should be set to “Normal” (Balanced). |

# 3 General CAPL API for streaming

In order to integrate MOST streaming applications into a simulation, testing or analysis environment, a set of intrinsic CAPL functions is provided to control the stream.

Please note:

- Because intensive signal processing may corrupt CANoe/CANalyzer's simulation time accuracy, processing synchronous streaming data in CAPL is neither intended nor recommended. All signal processing tasks should be processed within the streaming DLL. This is executed in the CANoe/CANalyzer process context, thus offering a high performance. This is, however, executed in a separate thread with lower priority than the real time CANoe/CANalyzer core, so that it has little impact on the performance of other analysis and simulation features. CAPL programs can control the streaming DLL and can only receive processing results.
- Controlling a stream with a short reaction time to other bus events requires direct access to the hardware interface. Therefore the streaming API is only available in the CANoe simulation setup and CANalyzer send branch. The API does not work within the measurement setup.
- In order to avoid compromising asynchronous transmissions on the MOST ring, it is only possible to stream data to the ring into the synchronous data area, as indicated by the recent SBC.
- For each MOST interface, only one stream to the MOST ring and one stream from the MOST ring can be instantiated via USB at the same time.
- As shown in the table below, VN2610 (MOST25) features many ways to route synchronous data from and to the ring. Due to hardware limitations, not all combinations are available at the same time. The following table shows the valid combinations:

| | **Line IN** | **Line OUT** | **S/PDIF IN** | **S/PDIF OUT** | **TX Stream via USB** | **RX Stream via USB** |
|---|---|---|---|---|---|---|
| **Line IN** | **-** | **OK** | **OK** | **OK** | **X** | **OK** |
| **Line OUT** | **OK** | **-** | **OK** | **OK** | **OK** | **X** |
| **S/PDIF IN** | **OK** | **OK** | **-** | **OK** | **X** | **OK** |
| **S/PDIF OUT** | **OK** | **OK** | **OK** | **-** | **OK** | **X** |
| **TX v. USB** | **X** | **OK** | **X** | **OK** | **-** | **OK** |
| **RX v. USB** | **OK** | **X** | **OK** | **X** | **OK** | **-** |

- The interface VN2640 for MOST150 supports all combinations. RX and TX streaming via USB is possible for label widths from 1 to 152 bytes per MOST frame. One label can be streamed to the MOST ring (this is automatically allocated on start) and up to 8 labels can be recorded.

| | **Line IN** | **Line OUT** | **S/PDIF IN** | **S/PDIF OUT** | **TX Stream via USB** | **RX Stream via USB** |
|---|---|---|---|---|---|---|
| **Line IN** | - | **OK** | **OK** | **OK** | **OK** | **OK** |
| **Line OUT** | **OK** | - | **OK** | **OK** | **OK** | **OK** |
| **S/PDIF IN** | **OK** | **OK** | - | **OK** | **OK** | **OK** |
| **S/PDIF OUT** | **OK** | **OK** | **OK** | - | **OK** | **OK** |
| **TX v. USB** | **OK** | **OK** | **OK** | **OK** | - | **OK** |
| **RX v. USB** | **OK** | **OK** | **OK** | **OK** | **OK** | - |

### 3.1 MostSyncStrmInit

| | |
|---|---|
| **Syntax** | `dword MostSyncStrmInit(long channel, long direction, char dllPath[])` |
| **Function** | MostSyncStrmInit() loads the streaming application DLL and binds it to the CAPL node. As a result, stream events (e.g. state and error data) are forwarded exclusively to this CAPL node.<br>This function can only be called in the ‘on prestart’ section. |
| **Parameter** | **channel**<br>Channel of the connected bus interface (VN2610, VN2640)<br>**direction**<br>Data flow direction<br>0 (RX): stream data from MOST to PC<br>1 (TX): stream data from PC to MOST<br>**dllPath**<br>Streaming application DLL path (relative to the CANoe/CANalyzer configuration file .cfg) |
| **Return Value** | A unique stream handle is returned. All further operations on the stream are addressed via this handle.<br>If an initialization failure occurs, 0 is returned. The CANoe/CANalyzer Write window provides further information.<br>The most common reasons for an error are:<br>• the streaming DLL path is invalid<br>• the streaming direction is not supported by the DLL<br>• the streaming DLL API is incompatible<br>• another stream with the same streaming direction is already initialized on the channel. |

### 3.2 MostSyncStrmGetHandle

| | |
|---|---|
| **Syntax** | `dword MostSyncStrmGetHandle(long channel, char dllPath[])`<br>`dword MostSyncStrmGetHandle(long channel, long direction, char dllPath[])` |
| **Function** | MostSyncStrmGetHandle() retrieves the handle of a previously initialized stream. This function is needed since the handle returned by the MostSyncStrmInit() call during the prestart phase cannot be stored in a CAPL variable for access during a test module execution.<br>The second signature should be used in case RX and TX streaming is processed in the same DLL. |

| | |
|---|---|
| **Parameter** | **channel**<br>Channel of the connected bus interface (VN2610, VN2640)<br>**direction**<br>Data flow direction (set in MostSyncStrmInit)<br>0 (RX): stream data from MOST to PC<br>1 (TX): stream data from PC to MOST<br>**dllPath**<br>Streaming application DLL path (relative to the CANoe/CANalyzer configuration file .cfg) |
| **Return Value** | A unique stream handle is returned. All further operations on the stream are addressed via this handle.<br>If a stream is not initialized for the specified parameters, 0 is returned. The CANoe/CANalyzer Write window provides further information. |

### 3.3 MostSyncStrmOpen

| | |
|---|---|
| **Syntax** | `long MostSyncStrmOpen(dword handle, long bytesPerFrame, dword options, long latency)` |
| **Function** | Opens a port for streaming. The callback OnMostSyncStrm-State(Opened) is invoked on success.<br>MOST150: MostSyncStrmOpen additionally allocates the label. The callback OnMostSyncStrmTxLabel() (see 3.12) reports the label number. |
| **Parameter** | **handle**<br>Stream handle<br>**bytesPerFrame** Number of bytes per MOST frame to be streamed (MOST25: 1…60; MOST150: 1…152). The concrete label numbers for RX streaming will be defined with MostSyncStrmStart (see section 3.4).<br>**options**<br>MOST25: Direction=RX: bit 0 set: add frame info to RX data (see chapter 6.2)<br>MOST150: not supported, write 0<br>**latency**<br>0 (short reaction time, high CPU load) ... 4 (slow reaction time, low CPU load) |
| **Return Value** | See section 3.13 |

### 3.4 MostSyncStrmStart

| | |
|---|---|
| **Syntax** | `MOST25: long MostSyncStrmStart(dword handle, byte syncChannels[])`<br>`MOST150: long MostSyncStrmStart(dword handle, word labelArray[], word labelWidthArray[], word numLabels)` |
| **Function** | MostSyncStrmStart starts the streaming of synchronous channel data from or to the MOST bus. The stream must already be opened. |
| **Parameter** | **handle**<br>Stream handle<br>**syncChannels[]**<br>MOST25 only: Array of synchronous channel numbers to be streamed. The array must contain bytesPerFrame valid channel numbers (see MostSyncStrmOpen).<br>**labelArray[8]**<br>MOST150 only: Array of labels for streaming from MOST to PC (RX streaming). All labels must exist in the allocation table when calling MostSyncStrmStart.<br>For TX streaming content of labelArray[] is ignored.<br>**labelWidthArray[8]**<br>MOST150 only: Array of label widths (RX streaming). Widths must match the actual width of labels given in labelArray[]. The sum of all widths must be identical to the parameter bytesPerFrame of MostSyncStrmOpen() (see section 3.3).<br>For TX streaming content of labelWidthArray[] is ignored.<br>**numLabels**<br>MOST150 only: Number of valid entries in labelArray[] and labelWidthArray[].<br>TX streaming: Set numLabels = 1 |
| **Return Value** | See section 3.13 |

### 3.5 MostSyncStrmStop

| | |
|---|---|
| **Syntax** | `long MostSyncStrmStop(dword handle)` |
| **Function** | Stops the streaming. The OnMostSyncStrmState(Stopped) callback is invoked on success. |
| **Parameter** | **handle**<br>Stream handle |
| **Return Value** | See section 3.13 |

### 3.6 MostSyncStrmClose

| | |
|---|---|
| **Syntax** | `long MostSyncStrmClose(dword handle)` |
| **Function** | Closes an open streaming port. The OnMostSyncStrmState(Closed) callback is invoked on success. |
| **Parameter** | **handle**<br>Stream handle |
| **Return Value** | See section 3.13 |

### 3.7 MostSyncStrmGetState

| | |
|---|---|
| **Syntax** | `long MostSyncStrmGetState(dword handle)` |
| **Function** | Returns the state of a stream. |
| **Parameter** | **handle**<br>Stream handle |
| **Return Value** | Stream state; see section 3.10 |

### 3.8 MostSyncStrmFillAllBuffers

| | |
|---|---|
| **Syntax** | `long MostSyncStrmFillAllBuffers(dword handle)` |
| **Function** | You can call MostSyncStrmFillAllBuffers() to prepare data for synchronous channel streaming in advance. This ensures minimum delay between the start command (MostSyncStrmStart) and the start of synchronous data streaming on MOST.<br>This function is available only for TX streaming. |
| **Parameter** | **handle**<br>Stream handle |
| **Return Value** | See section 3.13 |

### 3.9 MostSyncStrmClearAllBuffers

| | |
|---|---|
| **Syntax** | `long MostSyncStrmClearAllBuffers(dword handle)` |
| **Function** | MostSyncStrmClearAllBuffers discards all buffers filled by the streaming application. You should call this function after a streaming session is stopped and before the next session is started. This prevents the streaming of buffers that were filled in the previous session.<br>The function is available for TX streaming only. |
| **Parameter** | **handle**<br>Stream handle |
| **Return Value** | See section 3.13 |

### 3.10 Callback: OnMostSyncStrmState

| | |
|---|---|
| **Syntax** | `void OnMostSyncStrmState(dword handle, long state)` |
| **Function** | Defining this CAPL function returns information about stream state changes. |
| **Parameter** | **handle**<br>Stream handle<br>**state**<br>1: Closed<br>2: Opened<br>3: Started<br>4: Stopped |

### 3.11 Callback: OnMostSyncStrmError

| | |
|---|---|
| **Syntax** | `void OnMostSyncStrmError(dword handle, long errorcode)` |
| **Function** | Defining this CAPL function returns information on streaming errors (e.g. buffer overflows). |
| **Parameter** | **handle**<br>Stream handle<br>**errorcode**<br>See section 3.13 |

### 3.12 Callback: OnMostSyncStrmTxLabel

| | |
|---|---|
| **Syntax** | `void OnMostSyncStrmTxLabel(dword handle, long errorcode, long label, long labelWidth)` |
| **Function** | MOST150 only: This CAPL function is invoked after MostSyncStreamOpen has been called for a TX stream. It returns the allocated label for the stream or information on an error occurred while allocating (e.g. bandwidth problems). |
| **Parameter** | **handle**<br>Stream handle<br>**errorcode**<br>See section 3.13<br>**label**<br>Connection label<br>**labelWidth**<br>Width of the connection label in bytes per frame |

## 3.13 Streaming Error Codes

| Symbol | Value | Description |
|---|---|---|
| OK | 0 | |
| OpenFailed | 1 | See Write window for more details |
| CloseFailed | 2 | |
| StartFailed | 3 | |
| StopFailed | 4 | |
| DriverCallFailed | 10 | Bus interface driver call failed |
| DirectionNotSupported | 11 | The streaming direction is not supported by the client |
| NumSyncChannels-NotSupported | 12 | The number of streaming channels is not support-ed by the client |
| OptionsNotSupported | 13 | The streaming options are not supported by the client |
| LatencyNotSupported | 14 | The selected latency is not supported by the client |
| InvalidInterface | 15 | The streaming DLL has detected an invalid CAPL callback API |
| BufferUnderrunHW | 40 | 0 data was streamed (TX only) |
| BufferUnderrunClient | 41 | Client didn't provide buffer in time (TX only) |
| BufferOverflowHW | 42 | Synchronous data got lost (RX only) |
| BufferOverflowClient | 43 | Client didn't provide buffer in time (RX only) |
| BufferErrorClient | 44 | Streaming client was unable to prepare/process a buffer |
| BufferErrorStart | 45 | No buffers provided at start of streaming (TX: 0 data was streamed; RX: data got lost) |
| BufferErrorStop | 46 | Buffer underrun or overflow reported after the stream was stopped |
| NotEnoughBW | 51 | MOST150 only: The desired bandwidth for a TX stream cannot be allocated. |
| NetOffError | 52 | MOST150 only: NetInterface is in state NetOff. No streaming is possible. In case streaming was acti-vated it is automatically stopped. Additionally a TX stream is closed. |
| ConfigNotOKError | 53 | MOST150 only: The Network Configuration state switched to ‘NotOk’. Any active streaming is stopped. Additionally a TX stream is closed. |

| | | |
|---|---|---|
| CLDisappeared | 54 | MOST150 only: Any connection label from the RX stream disappeared, thus streaming is automatically stopped. |
| INICSocketError | 55 | MOST150 only: INIC reported a socket connection error for the TX stream. Streaming is automatically stopped and stream is closed. |
| DeviceModeBypass | 56 | MOST150 only: INIC's bypass was closed by application request. With closed bypass no streaming is possible, so streaming will be stopped automatically. Additionally a TX stream is closed. |
| NIStateNotNetOn | 57 | MOST150 only: NetInterface is not in NetOn, thus no streaming is possible. This error might be reported when opening the TX stream. |
| INICBusy | 58 | MOST150 only: INIC is currently busy processing other requests. The application may perform a retry. |
| CLMissing | 59 | MOST150 only: One or more connection labels are missing when trying to start the RX stream. |
| NumBytesMismatch | 60 | MOST150 only: The number of bytes per MOST frame given by the application does not match the number of bytes actually given by the connection labels for the RX stream. |
| INICCommunicationError | 61 | MOST150 only: A communication error with INIC occurred. |
| UnlockError | 70 | MOST150 only: RX streaming stopped due to an unlock event. As soon as lock is re-gained streaming will be continued. |
| UnknownError | 255 | |

## 3.14 Related CAPL functions

Please refer to the CANoe/CANalyzer Online Help to read more about following intrinsic CAPL functions: MostGetAllocTable, MostAllocTableGetCL, MostAllocTableGetSize, MostAllocTableGetWidth, MostGetSBC, MostReadReg, MostSyncAlloc, MostSyncDealloc, MostSetSyncAudio, MostSetSyncSpdif

Event handlers: OnMostAllocTable, OnMostSBC, OnMostStableLock, OnMostCriticalUnlock, OnMostSyncAudio, OnMostSyncSpdif

# 4 CAPL API test feature set for streaming

When testing applications, it is important to make sure that the sync streaming operations are performed successfully within a given time period. For this reason a set of test feature set functions are available that implement the same functionality as some of the functions described in chapter 3, but that also wait for the reception of an asynchronous acknowledge event from the hardware confirming the actual execution of the operation. Please note that you need to use MostSyncStrmInit() to initialize a stream in the “on prestart” handler of a test module and MostSyncStrmGetHandle() to retrieve the handle for that stream during test module execution.

## 4.1 TestMostSyncStrmOpen

| | |
|---|---|
| **Syntax** | `long TestMostSyncStrmOpen(dword handle, long bytesPerFrame, dword options, long latency, dword timeout)` |
| **Function** | Opens a port for streaming and waits for asynchronous acknowledge from the hardware. The OnMostSyncStrmState(Opened) callback is invoked on success. |
| **Parameter** | **handle**<br>Stream handle; must be retrieved via MostSyncStrmGetHandle()<br>**bytesPerFrame**<br>Number of bytes per MOST frame to be streamed (MOST25: 1…60; MOST150: 1…152). The concrete label numbers will be defined with MostSyncStrmStart (see section 3.4).<br>**options**<br>MOST25: Direction=RX: bit 0 set: add frame info to RX data (see chapter 6.2)<br>MOST150: not supported, write 0<br>**latency**<br>0 (short reaction time, high CPU load) ... 4 (slow reaction time, low CPU load)<br>**timeout**<br>Timeout in ms for to be waited for receipt of an acknowledge event from the hardware confirming successful execution of the operation. |
| **Return Value** | 1: Positive acknowledge received<br>0: Timeout while waiting<br>-3: Negative acknowledge received; execution failed<br>-4: Function call was performed while stream was in an unexpected state e.g. “Opened”, “Started” or “Stopped”.<br>-1: General error: Any other possible type of error |

### 4.2 TestMostSyncStrmStart

| | |
|---|---|
| **Syntax** | `MOST25: long TestMostSyncStrmStart(dword handle, byte syncChannels[], dword timeout)`<br><br>`MOST150: long TestMostSyncStrmStart(dword handle, word labelArray[], word labelWidthArray[], word numLabels, dword timeout)` |
| **Function** | TestMostSyncStrmStart starts the streaming of synchronous channel data from or to the MOST bus and waits for asynchronous acknowledge from the hardware. The stream must already be opened. |
| **Parameter** | **handle**<br><br>Stream handle; must be retrieved via MostSyncStrmGetHandle()<br><br>**syncChannels[]**<br><br>MOST25 only: Array of synchronous channel numbers to be streamed. The array must contain bytesPerFrame valid channel numbers (see MostSyncStrmOpen).<br><br>**labelArray[8]**<br><br>MOST150 only: Array of labels for streaming from MOST to PC (RX streaming). All labels must exist in the allocation table when calling TestMostSyncStrmStart.<br>For TX streaming content of labelArray[] is ignored.<br><br>**labelWidthArray[8]**<br><br>MOST150 only: Array of label widths (RX streaming). Widths must match the actual width of labels given in labelArray[]. The sum of all widths must be identical to the parameter bytesPerFrame of TestMostSyncStrmOpen() (see section 4.1).<br>For TX streaming content of labelWidthArray[] is ignored.<br><br>**numLabels**<br><br>MOST150 only: Number of valid entries in labelArray[] and labelWidthArray[].<br>TX streaming: Set numLabels = 1<br><br>**timeout**<br><br>Timeout in ms to be waited for receipt of an acknowledge event from the hardware confirming successful execution of the operation. |
| **Return Value** | 1: Positive acknowledge received<br><br>0: Timeout while waiting<br><br>-3: Negative acknowledge received; execution failed<br><br>-4: Function call was performed while stream was in an unexpected state, e.g. “Started” or “Closed”.<br><br>-1: General error: Any other possible type of error |

### 4.3 TestMostSyncStrmStop

| | |
|---|---|
| **Syntax** | `long TestMostSyncStrmStop(dword handle, dword timeout)` |
| **Function** | Stops the streaming and waits for asynchronous acknowledge from the hardware. The OnMostSyncStrmState(Stopped) callback is invoked on success. |
| **Parameter** | **handle**<br>Stream handle; must be retrieved via MostSyncStrmGetHandle()<br>**timeout**<br>Timeout in ms to be waited for receipt of an acknowledge event from the hardware confirming successful execution of the operation. |
| **Return Value** | 1: Positive acknowledge received<br>0: Timeout while waiting<br>-3: Negative acknowledge received; execution failed<br>-4: Function call was performed while stream was in an unexpected state, e.g. “Stopped” or “Closed”.<br>-1: General error: Any other possible type of error |

### 4.4 TestMostSyncStrmClose

| | |
|---|---|
| **Syntax** | `long TestMostSyncStrmClose (dword handle, dword timeout)` |
| **Function** | Closes an open streaming port and waits for asynchronous acknowledge from the hardware. The OnMostSyncStrmState(Closed) callback is invoked on success. |
| **Parameter** | **Handle**<br>Stream handle; must be retrieved via MostSyncStrmGetHandle()<br>**timeout**<br>Timeout in ms to be waited for receipt of an acknowledge event from the hardware confirming successful execution of the operation. |
| **Return Value** | 1: Positive acknowledge received<br>0: Timeout while waiting<br>-3: Negative acknowledge received; execution failed<br>-4: Function call was performed while stream was in an unexpected state. e.g. “Stopped” or “Closed”.<br>-1: General error: Any other possible type of error |

# 5 Individual CAPL API

A Streaming DLL extends the CAPL functionality by individual commands for configuring and controlling a data stream (e.g. setting a file to stream data from). In addition, the DLL can invoke user defined CAPL functions (e.g. notification when a file has been streamed completely).

## 5.1 CAPL extension

In order for the CAPL compiler and CAPL browser to recognize the Streaming DLL, you must link it to the CAPL program.

To do this, proceed as follows:

- Enter the DLL in the ‘includes’ section of a CAPL program using the #pragma library command. Use the path relative to the CAPL program. Example:
  #pragma library ("MostSyncStrmFromFile.dll")
- Call MostSyncStrmInit() in ‘on prestart’ to initialize the DLL (see section 3.1)

## 5.2 MostSyncStrmToFile.dll

This section describes the CAPL API for the MostSyncStrmToFile recorder DLL.

Note: Because all streaming DLL functions are called asynchronously, no return values are defined.

### 5.2.1 MostSyncStrmTFSetFile

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetFile(char filename[], long mode)` |
| **Function** | Specifies the file to stream data to. If streaming is already started, the current file is closed and the new file opened. |
| **Parameter** | **filename**<br>Target file for writing data. Only files with .bin, .wav and .ts extension are allowed.<br>If it is a “.wav” file, an error is triggered if MostSyncStrmOpen() was called with a bytesPerFrame parameter value other than 2 or 4, or an options parameter value with bit 0 set (see section 3.2).<br>If it is a “.ts” file the DLL switches to isochronous mode and records an MPEG transport stream (if such is transmitted at the specified connection label).<br>**mode**<br>0: open new file<br>1: append data (not available when using “.wav” files) |

### 5.2.2 MostSyncStrmTFSetSocket

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetSocket(dword address, long port)` |
| **Function** | Specifies an ip address and a port to which isochronous **MPEG transport stream** data is send to. To have any effect this function must be called prior to mostSyncStrmStart. If a valid address and a valid port are specified an UDP socket will be opened and MPEG transport stream will be sent to that socket (if such is transmitted at the specified connection label). The call to mostSyncStrmStop closes the socket and invalidates the address and the port. |
| **Parameter** | **address**<br>IP address for the UDP socket<br>**port**<br>Port number for the UDP socket |

### 5.2.3 MostSyncStrmTFSetDefaultPath

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetDefaultPath(char path[])` |
| **Function** | Sets the default path for target files. Relative file names set with MostSyncStrmTFSetFile are based on this default path. |
| **Parameter** | **path**<br>Default path |

### 5.2.4 MostSyncStrmTFSetMaxFrames

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetMaxFrames(dword frames)` |
| **Function** | Sets a file size limit. |
| **Parameter** | **frames**<br>Number of MOST frames per file; 0 for unlimited<br>The maximum number of bytes per file depends on the number of synchronous channels and the streaming options selected (see section 3.2). |

### 5.2.5 MostSyncStrmTFSetFileInc

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetFileInc(long digits)` |
| **Function** | Activates the file increment mode. When the maximum file size is reached, a new file is opened automatically. |
| **Parameter** | **digits**<br>1…8: number of decimal digits for the file name counter<br>0: deactivate file increment mode |

### 5.2.6 MostSyncStrmTFSetFileCnt

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetFileCnt(dword cnt)` |
| **Function** | Sets the file name counter. |
| **Parameter** | **cnt**<br>File name counter |

### 5.2.7 MostSyncStrmTFSetVerbose

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetVerbose(long level)` |
| **Function** | Sets the verbose level of the DLL. |
| **Parameter** | **level**<br>0: Output all messages<br>1: Output informational messages, warnings and errors<br>2: Output warnings and errors<br>3: Output errors only |

### 5.2.8 MostSyncStrmTSResetStatistics

| | |
|---|---|
| **Syntax** | `MostSyncStrmTSResetStatistics(dword type)` |
| **Function** | Resets MPEG transport stream statistics. |
| **Parameter** | **type**<br>1: PID statistics (PID numbers and lost frames counter)<br>2: General TS values (transport error counter, TS type, scrambling method, byte count) |

### 5.2.9 Callback: OnMostSyncStrmTFState

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmTFState(long state, char file[])` |
| **Function** | Returns the current file state. |
| **Parameter** | **state**<br>0: closed; 1: opened<br>**file**<br>Absolute file path |

### 5.2.10 Callback: OnMostSyncStrmTFError

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmTFError(long fileerror, char file[])` |
| **Function** | Returns errors from the streaming DLL. |

| | |
|---|---|
| **Parameter** | **fileerror**<br>1: No file specified<br>2: Error opening file<br>3: Error writing data<br>5: Maximum file size reached<br>**file**<br>Absolute file path |

### 5.2.11 Callback: OnMostSyncStrmTFProgress

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmTFProgress(dword bytesWitten, dword maxSize)` |
| **Function** | Returns the number of bytes written to the current file. |
| **Parameter** | **bytesWritten**<br>Number of bytes<br>**maxSize**<br>Maximum file size |

### 5.2.12 Callback: OnMostSyncStrmTSPIDStatistics

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmTSPIDStatistics(word bufPIDs[], long bufLostFrames[])` |
| **Function** | Returns the PIDs found in the MPEG transport stream so far (since the start of streaming or last mostSyncStrmTSResetStatistics(1) call). This callback is called periodically every 300 ms. |
| **Parameter** | **bufPIDs**<br>An array containing all occurred PIDs<br>**bufLostFrames**<br>An array containing the number of lost frames for each occurred PID<br>Note:<br>Both arrays contain the same number of entries (reflecting the number of occurred PIDs). The entries on the same index form a set:<br>bufPIDs[x] => PID1<br>bufLostFrames[x] => no of lost frames on PID1 |

### 5.2.13 Callback: OnMostSyncStrmTSStatistics

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmTSStatistics(dword noOfTSBytes, dword noOfAllBytes, dword tsc, dword tErrorCounter, dword tsType)` |
| **Function** | Returns statistics of the MPEG transport stream so far (since start of streaming or last mostSyncStrmTSResetStatistics(2) call). This callback is called periodically every 300 ms. |

| | |
|---|---|
| **Parameter** | **noOfTSBytes**<br>Number of bytes received and recognised as MPEG transport stream since the last callback<br>**noOfAllBytes**<br>Number of all bytes received on the specified connection label since the last callback<br>**tsc**<br>Indicates whether the MPEG transport stream is scrambled<br>**tErrorCounter**<br>Transport error counter<br>**tsType**<br>Type of the MPEG transport stream:<br>0: unknown<br>1: video<br>2: audio<br>3: video and audio<br><br>Note:<br>The overall amount of received TS bytes and/or all bytes have to be calculated respectively in CAPL. |

## 5.3 MostSyncStrmFromFile.dll

This section describes the CAPL API for the MostSyncStrmFromFile playback DLL.

Note: Because all streaming DLL functions are called asynchronously, no return values are defined.

### 5.3.1 MostSyncStrmFFSetFile

| | |
|---|---|
| **Syntax** | `MostSyncStrmFFSetFile(char filename[])` |
| **Function** | Specifies the file to stream data from. If streaming is already started, the current file is closed and the new file opened.<br>Since most TX buffers are already filled and transferred to the driver in this case, the streamed content changes if there is a delay. To avoid such delays, call MostSyncStrmClearAllBuffers(). |
| **Parameter** | **filename**<br>Source file<br>If it is a ".wav" file, an error is triggered if MostSyncStrmOpen() was called with a bytesPerFrame parameter value other than 2 or 4 or an options parameter value with bit 0 was set (see 3.2). |

### 5.3.2 MostSyncStrmFFSetNextFile

| | |
|---|---|
| **Syntax** | `MostSyncStrmFFSetNextFile(char filename[])` |
| **Function** | Specifies the next file to stream data from when the current file is finished streaming. |
| **Parameter** | **filename**<br>Source file |

### 5.3.3 MostSyncStrmFFSetDefaultPath

| | |
|---|---|
| **Syntax** | `MostSyncStrmFFSetDefaultPath(char path[])` |
| **Function** | Sets the default path for source files. Relative file names set with MostSyncStrmFFSetFile are based on this default path. |
| **Parameter** | **path**<br>Default path |

### 5.3.4 MostSyncStrmFFSetRepeat

| | |
|---|---|
| **Syntax** | `MostSyncStrmFFSetRepeat(long mode)` |
| **Function** | Enables/disables repeat mode. If repeat mode is not enabled, the streaming automatically stops as soon as the file is completely transmitted. If repeat mode is activated, the DLL transmits the file repeatedly without pausing. |

| | |
|---|---|
| **Parameter** | **mode**<br>0: Disable<br>1: Repeat current file |

### 5.3.5 MostSyncStrmFFSetVerbose

| | |
|---|---|
| **Syntax** | `MostSyncStrmTFSetVerbose(long level)` |
| **Function** | Sets the verbose level of the DLL. |
| **Parameter** | **level**<br>0: Output all messages<br>1: Output informational messages, warnings and errors<br>2: Output warnings and errors<br>3: Output errors only |

### 5.3.6 Callback: OnMostSyncStrmFFState

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmFFState(long state, char file[])` |
| **Function** | Reports the current file state. |
| **Parameter** | **State**<br>0: closed; 1: opened<br>**file**<br>Absolute file path |

### 5.3.7 Callback: OnMostSyncStrmFFError

| | |
|---|---|
| **Syntax** | `OnMostSyncStrmFFError(long fileerror, char file[])` |
| **Function** | Reports errors from the streaming DLL. |
| **Parameter** | **fileerror**<br>1: No file specified<br>2: Error opening file<br>4: Error reading data from file<br>5: End of file reached; provide a new file to stream without gap<br>6: Streaming of the current file is finished<br>**file**<br>Absolute file path |

## 5.4 MostSyncStrmRx2Tx.dll

### 5.4.1 MostSyncStrmRx2TxMute

| | |
|---|---|
| **Syntax** | `MostSyncStrmRx2TxMute (long mute, long channel)` |
| **Function** | Mutes or unmutes streamed audio. |
| **Parameter** | **mute**<br>0: unmute; 1: mute<br>**channel**<br>Audio channel to mute:<br>0: left audio channel, works only if 4 synchronous channels allocated<br>1: right audio channel, works only if 4 synchronous channels allocated<br>2: both audio channels |

### 5.4.2 MostSyncStrmRx2TxSetVolume

| | |
|---|---|
| **Syntax** | `MostSyncStrmRx2TxSetVolume (long volume)` |
| **Function** | Sets the volume of streamed audio. This function works only if 4 synchronous channels are allocated. |
| **Parameter** | **volume**<br>0…150: sound level in percent. |

### 5.4.3 Callback: OnMostStartTxStream

| | |
|---|---|
| **Syntax** | `OnMostStartTxStream ()` |
| **Function** | Reports that the TX stream is started. |
| **Parameter** | **n/a** |

# 6 Appendix

## 6.1 Example: Usage of MostSyncStrmToFile.DLL

The sample code shows a static way of using the MostSyncStrmToFile.DLL. To keep things simple, it does not show how to determine the channels for an audio stream of interest in a running system, nor how react to all erroneous situations. However the sample code operated in the sequence as shown here is all that is needed to record synchronous streams to a file.

```
includes
{
  // include DLL with file streaming extensions
  #pragma library ("MostSyncStrmToFile.dll")
}
variables
{
  dword gHandle;   // handle for synchronous channel streaming
}

on preStart
{
  // initialize extension DLL for file streaming
  // binds the CAPL node exclusively to the DLL
  // mostSyncStrmInit must be called in preStart
  gHandle = mostSyncStrmInit(1, RX, "MostSyncStrmToFile.dll");
}

on envVar EnvStreamTF_Open
{
  if(@this) return; // debounce button
  // maximum file size
  mostSyncStrmTFSetMaxFrames(441000);
  // file name counter with 3 digits
  mostSyncStrmTFSetFileInc(3);
  mostSyncStrmTFSetFile("..\\Filename.bin", 0);

  // prepare stream with 4 channels, no timestamps and low CPU load
  mostSyncStrmOpen(gHandle, 4, 0, 3);
}

on envVar EnvStreamTF_Start
{
  if(@this) return; // debounce button

  // select channels to stream on StreamOpen (MOST25)
```

```
  byte  channels[60] = { 0,1,2,3 };

  mostSyncStrmStart(gHandle, channels);

  // MOST150
  // word labels[8]; // up to 8 labels can be recorded.
  // word labelWidths[8];
  // labels[0] = 0x043; // set label number
  // labelWidths[0] = 4; // set label width
  // mostSyncStrmStart(gHandle, labels, labelWidths, 1);
}

on envVar EnvStreamTF_Stop
{
  if(@this) return; // debounce button
  mostSyncStrmStop(gHandle);
}

on envVar EnvStreamTF_Close
{
  if(@this) return; // debounce button
  mostSyncStrmClose(gHandle);
}

void OnMostSyncStrmTFError(long fileerror, char file[])
{
  // this streaming dll callback
  // reports a specific streaming error
  switch(fileerror)
  {
  case 1: write("%s: No file specified", gNodeName);   break;
  case 2: write("%s: Failed to open file %s", gNodeName, file); break;
  case 4: write("%s: Failed to read data from file", gNodeName); break;
  case 5: write("%s: End of file reached", gNodeName);
    mostSyncStrmStop(gHandle); break;
  }
}
```

## 6.2 Frame Format (MOST25)

The recorded frames (RX streaming) contain additional data when opening a MOST25 stream with options=0x00000001 (see section 3.2).

| Width | Description |
|---|---|
| 64 bit | Start of frame time stamp from hardware clock; unsynchronized; in 20 ns; LSB first |
| 2, 4, 6... 60 bytes | MOST frame data; the number of bytes depends on the MostSyncStrmOpen bytesPerFrame parameter; for odd values of bytesPerFrame a fill byte (0xFB) is inserted. |
| 8 bit | Reserved |
| 4 bit | SBC (mask: 0b11110000) |
| 1 bit | Light status (mask: 0b00001000) |
| 1 bit | Lock status (mask: 0b00000100) |
| 1 bit | Overflow flag (mask: 0b00000010) |
| 1 bit | Underflow flag (mask: 0b00000001) |

## 6.3 Glossary

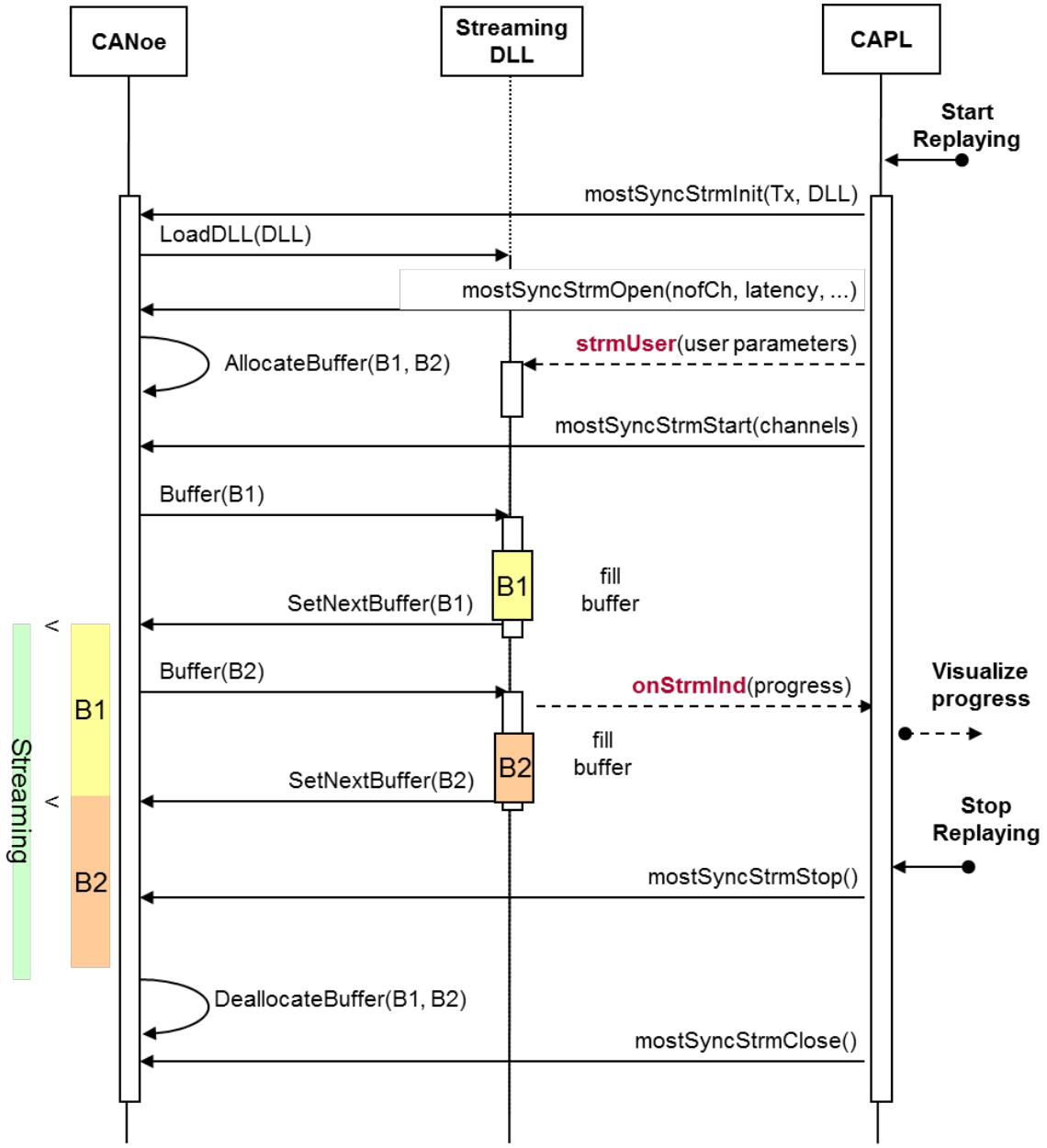
| Name | Explanation |
|---|---|
| Quadlet | A group of four synchronous channels. |
| Synchronous channel | Single byte in the synchronous data area of a MOST frame which can be used individually or with a group of other synchronous channels to transport e.g. audio data. |
| Synchronous Boundary Control | Value transmitted in the header of each MOST frame which indicates how many bytes of the frame are reserved for use. |
| SBC | See Synchronous Boundary Control. |
| Stream | The stream of data transmitted via USB. |
| Synchronous data area | Area in the MOST frame which is administered by the timing master. A node requests the timing master to reserve a number of channels (bytes) for a cyclic transmission of data with a guaranteed bandwidth. |
| Audio stream | A group of synchronous channels, transmitting data from a single audio source. Typically four channels are used for a CD-quality stereo audio transmission. |
| Streaming DLL | DLL running in the CANoe/CANalyzer context that implements the algorithm for processing synchronous data. |
| Connection label | Label shown in the allocation table. The label is assigned to a number of synchronous channels, indicating that this group of channels is used to transmit logically related data, e.g. an audio stream. |

## 6.4 Buffer handling in streaming DLLs

The lists of parameters of the APIs are partly truncated for better readability. Beside the indications during normal operation additional indications are given for buffer overflows or underflows and streaming state changes.

The commands indicated in red are examples for user defined commands, which may either allow a further configuration of the DLL from CAPL or the indication of results from the DLL to the CAPL node. Commands may be used to provide the DLL with additional information like changes of the allocation table, selected control messages or user requests. User defined indications may be used to report analysis results of the streaming DLL to the CAPL node.

### 6.4.1 Streaming to the MOST ring



### 6.4.2 Recording