UM3297

User manual

# Getting started with X-CUBE-AUDIO-KIT

## Introduction

This user manual describes how to get started with X-CUBE-AUDIO-KIT, the STM32 ecosystem for audio flow edition and tuning:

- On the device side, the firmware infrastructure called AudioChain allows the implementation of data flows in a generic manner.
- On the computer side, a graphical interface called LiveTune allows the design and tuning of data flows.

This document provides the basic knowledge to start a development with X-CUBE-AUDIO-KIT. More details are available in the reference manual embedded in the Expansion Package.

# 1 General information

The X-CUBE-AUDIO-KIT STM32Cube Expansion Package runs on the STM32H7 microcontrollers based on the Arm® Cortex®-M7 processor, and on the STM32H5 microcontrollers based on the Arm® Cortex®-M33 processor.

Note: *Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.*

## 1.1 Terms and definitions

The Table 1 defines the terms used in this document.

Table 1. Term definitions

| Term | Definition |
|---|---|
| acChunk | LiveTune audio bus |
| acGraph | LiveTune graphical bus |
| acMsg | LiveTune message bus |
| Algorithm | Audio-handling plugin |
| AudioChain | Multipath audio processing framework |
| Chunk | Multiframe buffer to connect algorithms |
| CPU | Central processing unit (Arm® Cortex®) |
| Data flow | A set of elements connected by a multiframe buffer |
| Designer | LiveTune communication component |
| Element | A part of a data flow |
| Grph | LiveTune graphical pin |
| IDE | Integrated development environment |
| In | LiveTune input audio pin |
| LiveTune | A designer canvas to build a data flow graphically |
| Msg | LiveTune message pin |
| Out | LiveTune output audio pin |
| UART | Universal asynchronous receiver transmitter |
| USB | Universal serial bus |

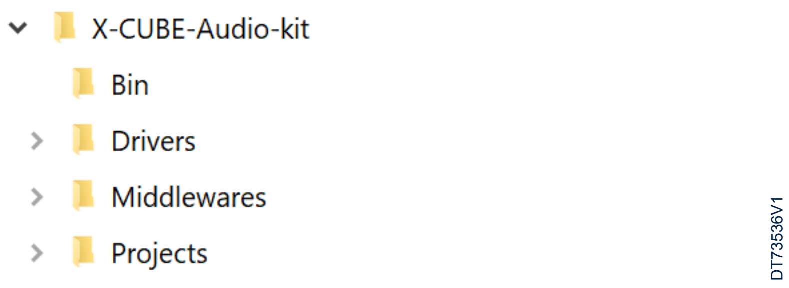
## 1.2 References

- Reference manual, available in LiveTune in HTML ([**Help**] button)
- *Integrated development environment for STM32 products* data brief (DB3871)
- *STM32CubeIDE user guide* user manual (UM2609)
- *STM32H563/H573 and STM32H562 Arm®-based 32-bit MCUs* reference manual (RM0481)
- *STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs* reference manual (RM0399)

# 2 X-CUBE-AUDIO-KIT Expansion Package architecture

The X-CUBE-AUDIO-KIT Expansion Package complies with the standard STM32 firmware package organization presented in Figure 1.

**Figure 1. STM32 firmware package structure**



- `Drivers`: level-0 components including the CMSIS core files.
- `Middlewares`: native STMicroelectronics and third-party stacks and protocol-based libraries.
- `Projects`: set of applicative projects that explain and provide use cases for the different product features:
  – Based on the product hardware (boards, STM32 core features, interconnections, and peripherals).
  – Built around the different firmware components.
- `Utilities`: miscellaneous software utilities that provide:
  – Additional system and media resource services such as CPU usage calculation with FreeRTOS™ kernel, and trace utilities.
  – Standard library services such as memory, string, and time and math services.

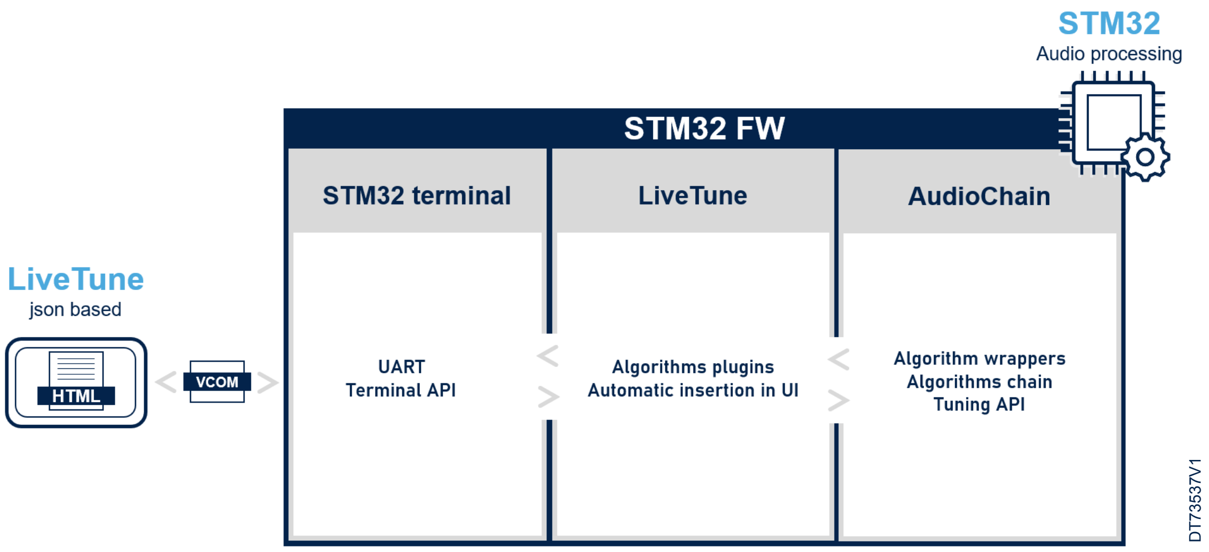
  Some utilities can be specific to the package.

X-CUBE-AUDIO-KIT is mainly implemented in the following folders:

- `Middlewares/ST/Audio-kit`
- `Projects`
- `Utilities`

# 3 System overview

Figure 2. System overview



## 3.1 AudioChain overview

AudioChain is a multipath audio processing framework. It offers a generic way of building simple to complex audio flows made of different processing blocks.

AudioChain receives its input data from audio buffers and provides its results to output buffers. As a result, it is independent from hardware and BSP source code. It can work on many different types of audio data such as:

- Temporal or spectral domain
- Fixed or floating-point
- Interleaved or not
- Mono, stereo, or a wider range of channels

AudioChain supports a wide variety of audio-handling components such as routing blocks, gains, audio analysis blocks, audio processing, and voice audio denoising.

These elements are provided as plugins called *“algorithms”* that the user can add and remove to create a complete data flow. It is also possible to encapsulate external or third-party processing blocks to insert them in an AudioChain data flow.

Connecting algorithms together is done through multiframe buffers called *“chunks”*. The chunks used to interface with external audio hardware management (such as microphones, USB, audio codec, or others) have some specificities. These specific chunks are called system chunks. More details are given in a dedicated section of the reference manual, called *AudioChain connection to hardware*.

*Note:* *The data flow can be implemented by using LiveTune and also by writing C code manually.*

## 3.2 LiveTune overview

LiveTune is a designer canvas to build a data flow graphically. The result of the design can be tested in three ways:

- Immediately from the tool:
  - Using the [**Start**] button.
- Programming the design from the tool:
  - Using the [**Flash**] button.
    The initial executable file that allows the use of the LiveTune designer does not necessarily implement any data flow. As long as no data flow has been programmed, it does not execute any audio processing.
    The advantage of this way of working is to program rapidly one data flow or another.
- Generating the C code version of the data flow:
  - Using the [**Generate code**] button.
    This is used to create more compact executables, and specific intermediate and final firmware (refer to Section 7: Prepare the release and extra code integration and Section 8: Build the final release).

LiveTune uses an HTML5 compliant browser. The communication with the device is using the Virtual COM port through the ST-LINK USB interface.

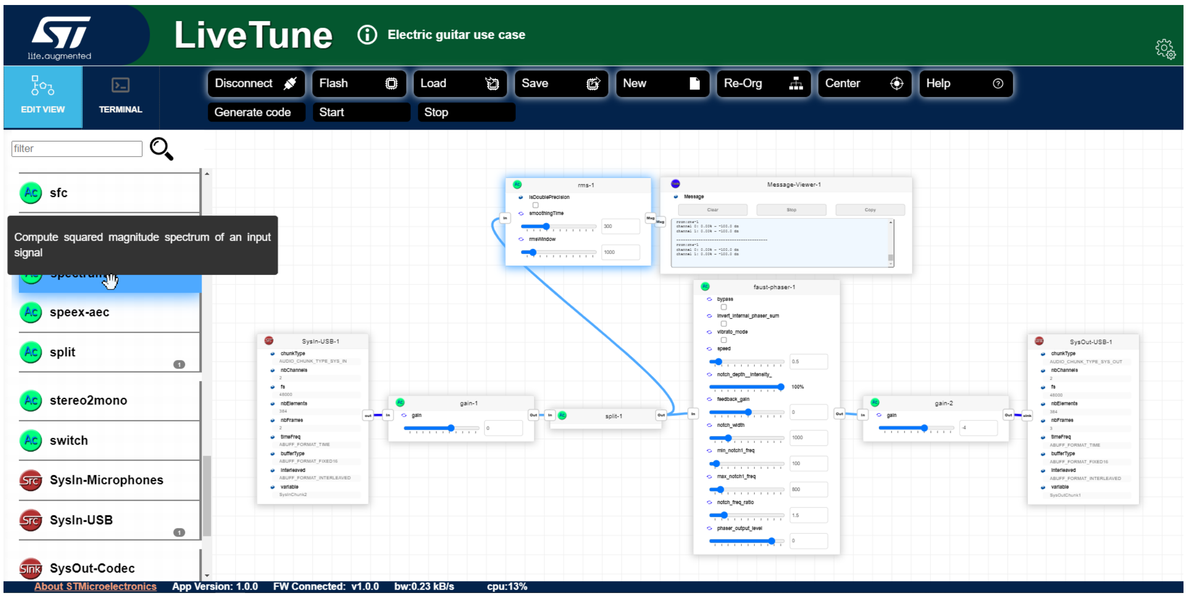
LiveTune also offers a terminal to:

- Get a feedback trace from the device
- Interact with the device through commands, for example for:
  - Any user interface event
  - Traces and cycles count activation
  - Traces and cycles count deactivation
  - USB record data flow selection

  A `help` command is available to list all the possible interactions.

Figure 3 and Figure 4 show the *EDIT VIEW* and *TERMINAL* views of LiveTune.

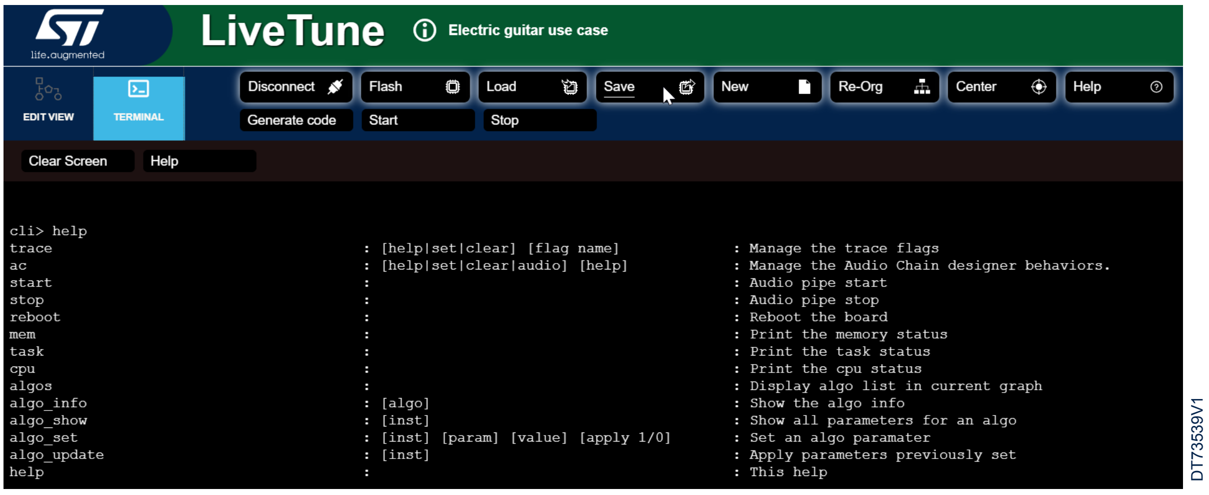
**Figure 3. LiveTune *EDIT VIEW***



Compute squared magnitude spectrum of an input signal

About STMicroelectronics App Version: 1.0.0 FW Connected: v1.0.0 bw:0.23 kB/s cpu:13%

Figure 4. LiveTune *TERMINAL*



```
cli> help
trace                                  : [help|set|clear] [flag name]            : Manage the trace flags
ac                                     : [help|set|clear|audio] [help]           : Manage the Audio Chain designer behaviors.
start                                  :                                         : Audio pipe start
stop                                   :                                         : Audio pipe stop
reboot                                 :                                         : Reboot the board
mem                                    :                                         : Print the memory status
task                                   :                                         : Print the task status
cpu                                    :                                         : Print the cpu status
algos                                  :                                         : Display algo list in current graph
algo_info                              : [algo]                                  : Show the algo info
algo_show                              : [inst]                                  : Show all parameters for an algo
algo_set                               : [inst] [param] [value] [apply 1/0]      : Set an algo paramater
algo_update                            : [inst]                                  : Apply parameters previously set
help                                   :                                         : This help
```

Upon a successful connection to a device, LiveTune displays a left panel with the algorithm plugins available for the data flow construction. The list of elements exposed in LiveTune depends on the firmware build. Refer to the *Developer API* chapter in the reference manual for a complete description of the algorithms.

- Some buttons previously grayed or hidden become available according to the board firmware configuration.
- The graph panel becomes editable in the *EDIT VIEW*.
- The *TERMINAL* view starts logging traces. It also allows interactions through the command line.

Use the [**Connect**] button to connect to the device. This button is replaced with the [**Disconnect**] button when the connection is successful. The connection is successful only if the firmware running on the device includes the *Designer* communication firmware.

# 4 Firmware installation

The X-CUBE-AUDIO-KIT Expansion Package comes with preported boards. To install the firmware, the user can either build it from scratch, or directly program a prebuilt firmware.

Two flavors are possible to build the firmware:

- IAR Systems® IAR Embedded Workbench® using a custom toolchain
- STMicroelectronics STM32CubeIDE using the GCC toolchain

## 4.1 Build configurations

Each project produces several configurations for use at different development steps.

### 4.1.1 LiveTune

This configuration includes the *Designer* component. This component handles the communication between LiveTune and the firmware. It builds the pipe and generates the code for a use case. It also exposes a serial terminal including various tools such as CPU, memory, and task monitors. Furthermore, the *Designer* component adds support for extra memory and external flash memory.

This configuration allows changes in the audio configuration using the command-line interface: `cli>ac audio`.

### 4.1.2 Devel_code_generated_[UC name]

These configurations are meant for firmware prerelease. They use a source code version of the data flow generated from LiveTune. The default one given as an example is `audio_chain_generated_echo.c`.

Since a specific data flow is running, the host is not aware of the full audio capabilities of AudioChain, but rather of the capabilities of the data flow. In these configurations, the LiveTune support is partial. For instance, the designing capabilities are disabled but the tuning ones are still available. This is called the *"tuner mode"*.

In such configurations, the use case starts immediately. Defines are used to set the audio configuration, which cannot be changed dynamically. The terminal remains active, and the user can tune the firmware in two ways:

- Use a serial terminal like Tera Term to interact with the board
- Use the LiveTune tuner mode, a subset limited to tuning, meaning that all the design capabilities are disabled

### 4.1.3 Release_code_generated_[UC name]

These configurations are optimized for the release firmware.

In such configurations, the use case starts immediately. Some flags are set to remove all unnecessary pieces of codes.

These configurations are similar to Devel_code_generated_[UC name], with the exception that:

- The terminal support is removed
- The build is optimized for speed
- The define ALGO_USE_LIST is added to the project

The configuration does not produce any traces; Only the errors are notified.

## 4.2 Program a prebuilt firmware

X-CUBE-AUDIO-KIT contains prebuilt example configurations for each flavor. The folders named `Bin[toolset name]` contain the appropriate firmware to program.

The user can use STM32CubeProgrammer (STM32CubeProg) to program directly the `.hex` file.

## 4.3 Build firmware from scratch

### 4.3.1 IAR Embedded Workbench®

1. Navigate to the project folder `EWARM`
2. Double-click on the file with the extension `.eww`
3. Select the configuration
4. Make the firmware using the key **F7** or the menu [**Project**]

### 4.3.2 STM32CubeIDE

1. Navigate to the project folder `STM32CubeIDE`, where each configuration has a specific subfolder
2. Open the subfolder of the configuration to be built
3. Double-click on the file `.cproject`
4. Wait until STM32CubeIDE has opened and imported the project
5. See that the project name is displayed in the window *Project Explorer*
6. Right-click on the configuration name and select [**Build project**] in the pop-up menu

## 4.4 Build an audio use case

Audio use cases can be designed programmatically or using LiveTune. LiveTune offers a way to design a use case graphically and to generate the AudioChain-compliant source code. AudioChain is the middleware component in charge of building and rendering the audio data flow on the target. It can be tested and validated from LiveTune. The code always runs on the target; There is no emulation on the PC.

LiveTune uses the ST-LINK serial COM port to communicate with the firmware. While starting a design session, the firmware notifies LiveTune with its audio capabilities: supported algorithms, saved projects, and others. The tool gathers this information and exposes it to the user.

LiveTune also has a terminal emulation to interact with the firmware using a command-line interface (CLI). Press the **Enter** key to enter the CLI mode.

Finally, LiveTune includes a complete reference manual in HTML accessible through the [**Help**] button.

# 5 LiveTune concepts

LiveTune is built around three notions:

- Elements
- Pins
- Buses

## 5.1 LiveTune elements

AudioChain uses the concept of data flow. An audio data flow is the set of elements connected by a multiframe buffer. LiveTune uses several types of elements symbolized by dedicated icons. An element represents a connectable item. An element can be:

- An algorithm or processing block:
  - Some are already available in AudioChain
  - Others can be external or third-party solutions
    (Refer to the Speex echo canceler for instance, for which proper wrapping and integration was done)
- A sink or a source element to handle a connection with hardware
- A tool

Table 2. **LiveTune icons**

| Icon | Type | Comment |
|---|---|---|
| Ac | AudioChain algorithm element | It is a processing block. It processes or generates or analyzes the audio data provided and delivered in the shape of audio chunks. It also has the possibility to generate control events and notifications. |
| Sink | AudioChain sink element | It is an element connected to the hardware or to an audio output. Most of the time, it is a bus connected to a loudspeaker, a USB, or other audio data provider. However, other sink components can also be added. |
| Src | AudioChain source element | It is an element connected to the hardware or to an audio source. Most of the time it is the bus connected to a microphone, a USB, or other audio data consumer. However, other source components can also be added. |
| Tools | LiveTune tool | It is an element not using an AudioChain bus. It takes the form of a visualizer or any independent tool. |

An element has its own capabilities. Depending on the element implementation, it might support one or different possibilities for:

- The sampling frequency
- The number of channels
- The number of input and output buses
- The type of interleaving format
- The sample format (such as float, q15, q31, or others)
- Temporal or spectral data

On the contrary, some elements have the possibility to support a different tuning of these parameters.

## 5.2 LiveTune pin types

Creating a data flow involves the connection of several elements from a source pin to a destination pin. Each element exposes named pins.

There are three distinct types of pins:

- Audio pins:
  - Often called *In* and *Out* by default
  - Sometimes with a specific name, such as *Err* for the element called nlms
  - Used for audio connection
- *Msg*:
  - It can be connected to an *acMsg* bus
  - It provides control data that message viewers can display
- *Grph*:
  - It can be connected to an *acGraph* bus
  - It provides data that message viewers can display

The elements can have different capabilities. A connection between two elements requires that their capabilities are consistent. LiveTune detects most of the inconsistency cases and prevents the errors. Other errors are detected at start time by the firmware.

## 5.3 LiveTune bus types

The connections between elements are using distinct types of buses according to the type of the pins. There are three types of buses:

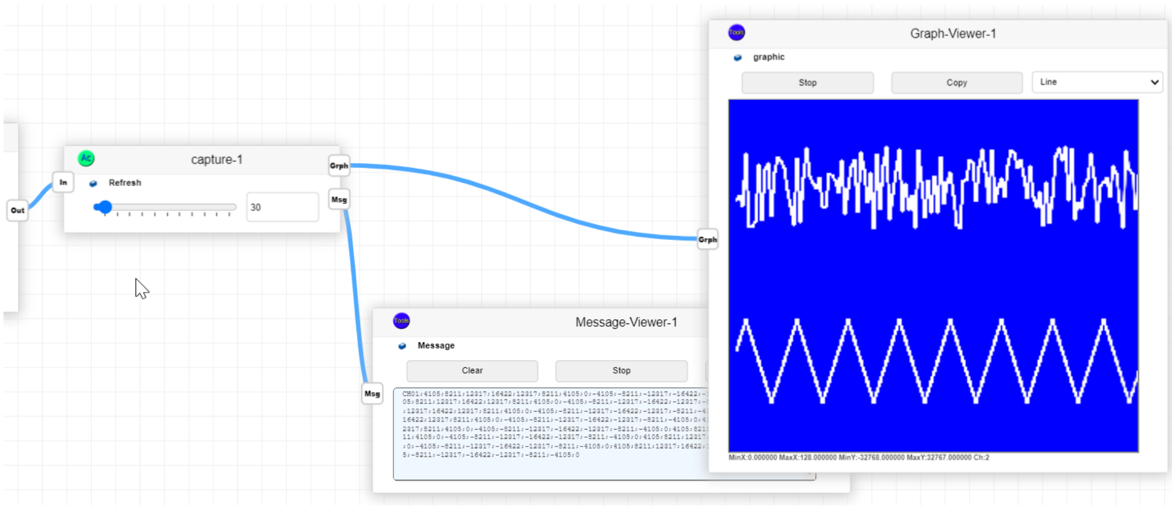
- The *acChunk* bus is used to connect audio pins together. This type of bus is internal to AudioChain. The term chunk refers to the AudioChain firmware. `audio_chunk_t` is used for the multiframe audio buffer structures.
- The *acGraph* bus is used to connect *Grph* pins together. It sends data to a graphical viewer. This bus is not internal to AudioChain. It is a connection between the device and LiveTune, meaning that the data is transmitted from the device to LiveTune.
- The *acMsg* bus is used to connect *Msg* pins together. It connects text data to a message viewer. This bus is not internal to AudioChain. It is a connection between the device and LiveTune, meaning that the data is transmitted from the device to LiveTune.

As a result, it is not allowed to connect audio pins to *Msg* or *Grph* pins. The authorized connections are:

- Audio pins to audio pins by means of the *acChunk* bus
- *Grph* pins to *Grph* pins by means of the *acGraph* bus
- *Msg* pins to *Msg* pins by means of the *acMsg* bus

To send audio data to a graphical or message viewer, a bus converter must be used such as the element called capture. Figure 5 illustrates a bus conversion using the capture element.

**Figure 5. Bus conversion example**



```
CH01;4105;8211;12317;16422;12317;8211;4105;0;-4105;-8211;-12317;-16422;-
05;8211;12317;16422;12317;8211;4105;0;-4105;-8211;-12317;-16422;-12317;-
;12317;16422;12317;8211;4105;0;-4105;-8211;-12317;-16422;-12317;-8211;-4
16422;12317;8211;4105;0;-4105;-8211;-12317;-16422;-12317;-8211;-4105;0;4
2317;8211;4105;0;-4105;-8211;-12317;-16422;-12317;-8211;-4105;0;4105;821
11;4105;0;-4105;-8211;-12317;-16422;-12317;-8211;-4105;0;4105;8211;12317
;0;-4105;-8211;-12317;-16422;-12317;-8211;-4105;0;4105;8211;12317;16422;
5;-8211;-12317;-16422;-12317;-8211;-4105;0
```

```
MinX:0.000000 MaxX:128.000000 MinY:-32768.000000 MaxY:32767.000000 Ch:2
```

DT73540V1

# 6 Build a data flow

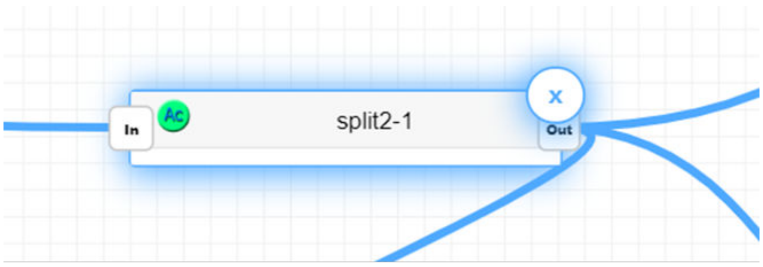
To build a data flow:
- Create the element instances
- Connect the elements
- Tune the elements

To create an element instance as part of a data flow, select it from the element panel and drag-and-drop it on the graph panel.

*Tip:* *Hovering over an element in the element panel pops up a tooltip with a short element description.*

To delete an element from a data flow, right-click on the element instance. A connection is removed by clicking on the cross symbol as shown in Figure 6. It is also possible to select the block and press the **DEL** key.

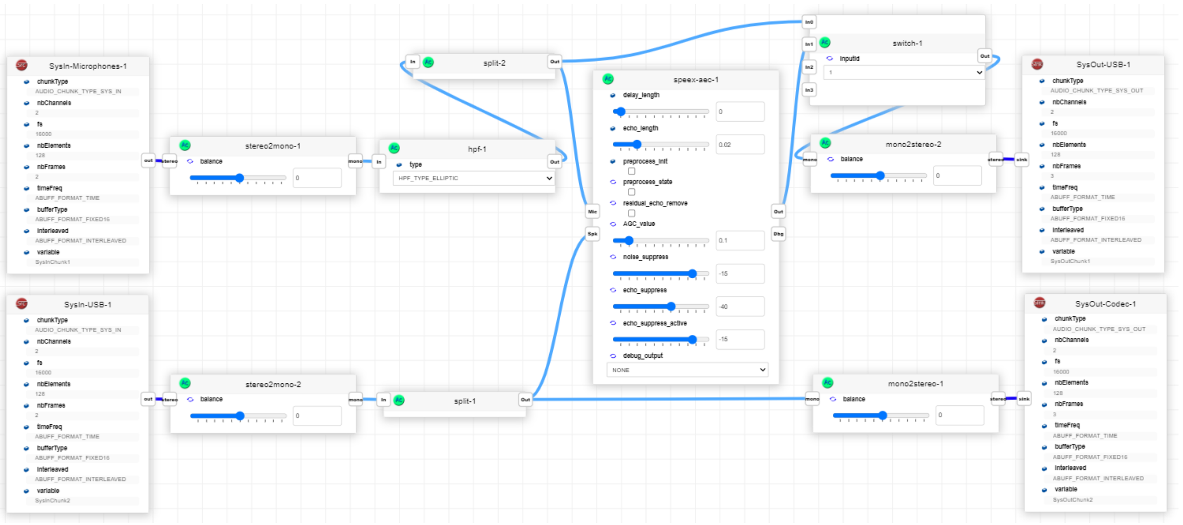
Figure 6. **Element deletion symbol**



The split component is the only component having multiple output pin capability. This component is used to duplicate an output signal.

Figure 7 presents a data flow example using various types of components and connections.

Figure 7. **Data flow example**



SysIn-Microphones-1
chunkType: AUDIO_CHUNK_TYPE_SYS_IN
nbChannels: 2
fs: 16000
nbElements: 128
nbFrames: 2
timeFreq: ABUFF_FORMAT_TIME
bufferType: ABUFF_FORMAT_FIXED16
interleaved: ABUFF_FORMAT_INTERLEAVED
variable: SysInChunk1

SysIn-USB-1
chunkType: AUDIO_CHUNK_TYPE_SYS_IN
nbChannels: 2
fs: 16000
nbElements: 128
nbFrames: 2
timeFreq: ABUFF_FORMAT_TIME
bufferType: ABUFF_FORMAT_FIXED16
interleaved: ABUFF_FORMAT_INTERLEAVED
variable: SysInChunk2

stereo2mono-1
balance: 0

stereo2mono-2
balance: 0

hpf-1
type: HPF_TYPE_ELLIPTIC

speex-aec-1
delay_length: 0
echo_length: 0.02
preprocess_init
preprocess_state
residual_echo_remove
AGC_value: 0.1
noise_suppress: -15
echo_suppress: -40
echo_suppress_active: -15
debug_output: NONE

switch-1
inputId: 1

mono2stereo-2
balance: 0

mono2stereo-1
balance: 0

SysOut-USB-1
chunkType: AUDIO_CHUNK_TYPE_SYS_OUT
nbChannels: 2
fs: 16000
nbElements: 128
nbFrames: 3
timeFreq: ABUFF_FORMAT_TIME
bufferType: ABUFF_FORMAT_FIXED16
interleaved: ABUFF_FORMAT_INTERLEAVED
variable: SysOutChunk1

SysOut-Codec-1
chunkType: AUDIO_CHUNK_TYPE_SYS_OUT
nbChannels: 2
fs: 16000
nbElements: 128
nbFrames: 3
timeFreq: ABUFF_FORMAT_TIME
bufferType: ABUFF_FORMAT_FIXED16
interleaved: ABUFF_FORMAT_INTERLEAVED
variable: SysOutChunk2

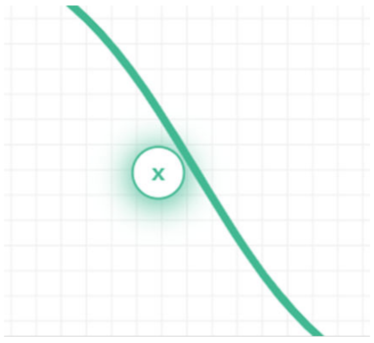
## 6.1 Connect elements in the designer

To build a connection, left-click on the source pin and drag the connection onto the destination pin. The connection is then created when the mouse button is released.

If the connection is revoked, check the information on the terminal. An inconsistency between pins might have been raised and logged. In that case, consider using an intermediate element such as the SFC (signal format converter), a router, or any other block that can convert formats.

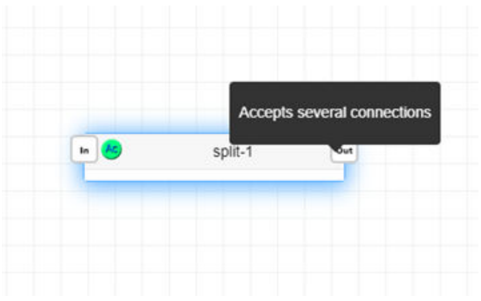
To delete a connection, right-click on the connection wire so that the deletion icon pops up. The connection is removed by clicking on the cross deletion symbol as shown in Figure 8.

Figure 8. **Connection deletion symbol**



An element can expose several pins with various names. A tooltip is displayed when the cursor hovers over the pin. The message gives some information concerning the number of connections accepted or the limitation of this pin. The following figure illustrates this for the processing component called a router.

Figure 9. **Pin labeling**



The same tooltip mechanism is available for the connections. It specifies the bus type, the connection name, and optionally if some parameters can be changed (that is if several capabilities are supported).

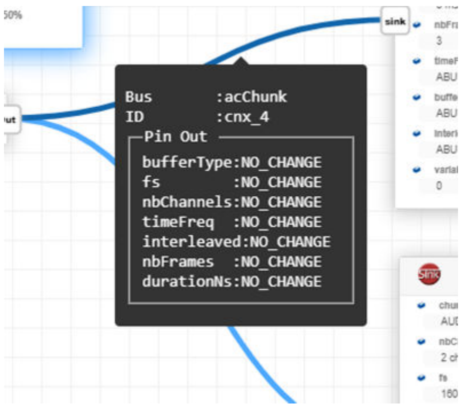
The NO_CHANGE indication, as shown in Figure 10, means that the acChunk parameters are inherited from the source element. This behavior is referred to as capability propagation.

For instance:

- If a router takes some floating-point *acChunk* as input and if it delivers into an output *acChunk* that has a buffer type as "bufferType: NO_CHANGE", the data type is unchanged.
- If a router takes some floating-point acChunk as input and if the user changes the buffer type to "bufferType: ABUFF_FORMAT_FIXED16", the data type is converted.

The same mechanism occurs for the interleaving type.

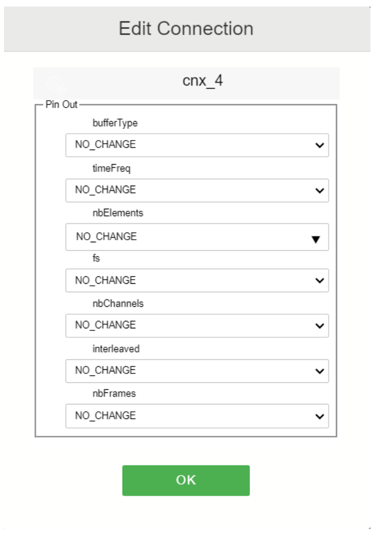
Figure 10. Connection labeling



```
Bus        :acChunk
ID         :cnx_4
Pin Out
  bufferType:NO_CHANGE
  fs         :NO_CHANGE
  nbChannels:NO_CHANGE
  timeFreq   :NO_CHANGE
  interleaved:NO_CHANGE
  nbFrames   :NO_CHANGE
  durationNs:NO_CHANGE
```

Changing the connection settings is done by left-clicking on the connection wire. The following figure shows the type of popup dialog box that allows the connection tuning.

Figure 11. Connection setting dialog box



Edit Connection

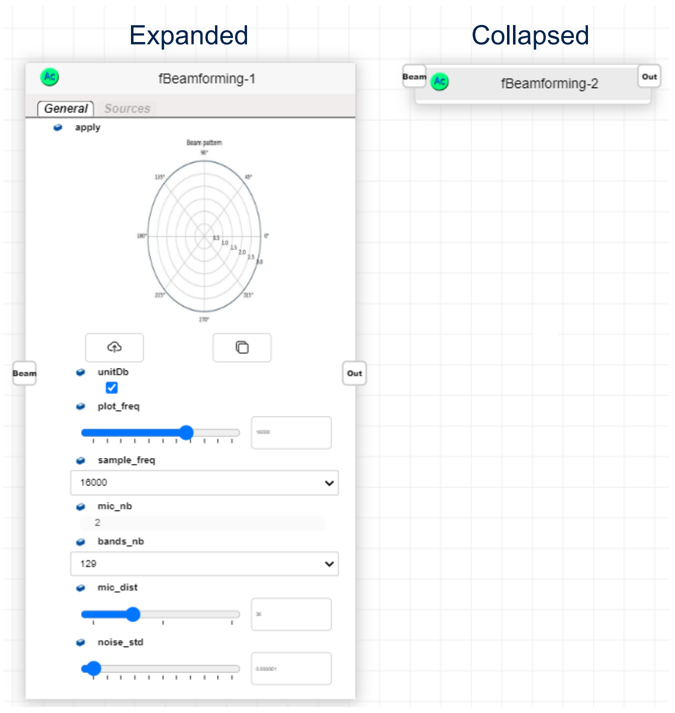
cnx_4

Pin Out

| | |
|---|---|
| bufferType | NO_CHANGE |
| timeFreq | NO_CHANGE |
| nbElements | NO_CHANGE |
| fs | NO_CHANGE |
| nbChannels | NO_CHANGE |
| interleaved | NO_CHANGE |
| nbFrames | NO_CHANGE |

When an algorithm setting is done, it is possible to collapse all parameters to reduce the box size by double-clicking on the algorithm icon. The first time the box is collapsed, the second time the box is expanded.

Figure 12. **Expanded and collapsed element views**



## 6.2 Navigation

To make the edition of a data flow and the navigation in a data flow easier, LiveTune provides zooming and translation capabilities. Table 3 the edition and navigation shortcuts available in LiveTune.

Table 3. **LiveTune edition and navigation shortcuts**

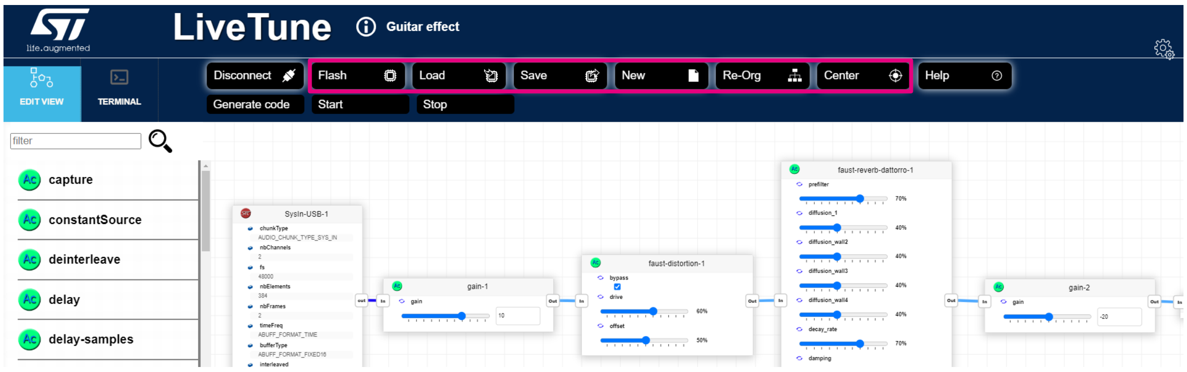
| Shortcut | Description |
|---|---|
| **Ctrl** + ‘**+**’ | Zoom in the complete browser view |
| **Ctrl** + ’**-**‘ | Zoom out the complete browser view |
| **Ctrl** + ‘**scroll**’ | Zoom the data flow in and out |
| **Ctrl** + ‘**mouse middle click**' + '**move**’ | Translate the data flow |
| '**Left-click on a free space**' + '**move**' | Translate the data flow |
| '**Right-click on a free space**' + '**move**' | Translate the data flow |
| '**Right-click**' on an object | Show the deletion icon |

## 6.3 Save and open designs

The LiveTune user interface offers different buttons to manage the data flow edition and backup:

- [**Flash**]: saves the design on the device. The firmware has a persistent storage in the flash memory. When this button is pressed, the design is automatically loaded at reboot.
- [**Load**]: opens a design in the canvas. It is also possible to drag-and-drop the `xxx.LiveTune` file onto the canvas to load an existing design in LiveTune. The project is then loaded and transmitted to the board but not persistently.
- [**Save**]: saves the design currently in the canvas area. The file extension is `.LiveTune`.
- [**New**]: clears the canvas.
- [**Re-Org**]: distributes all blocks evenly.
- [**Center**]: distributes the data flow in the center of the canvas.

Figure 13. **LiveTune user interface**

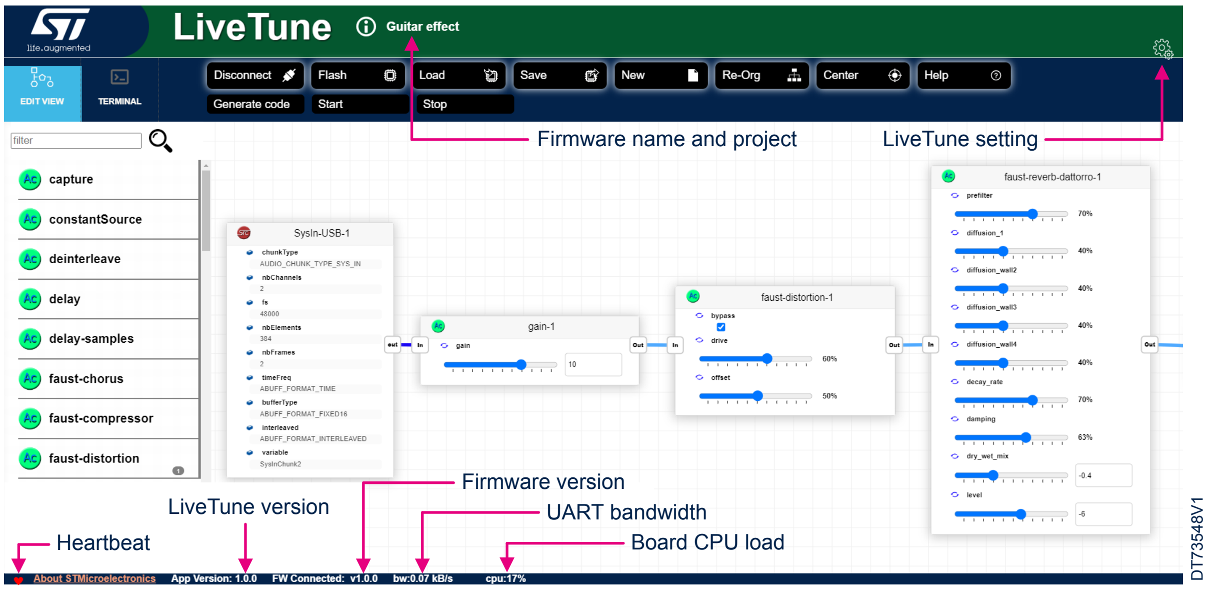


## 6.4 Status bar

When a project is loaded in LiveTune, the status bar and the header display information about the firmware:

- Firmware name:
  - Shows the firmware name, which exposes a configuration or a special build for a given use case.
- Firmware version:
  - Shows the firmware version.
- Heartbeat, which shows the board activity:
  - No heart: not connected.
  - Beating white heart: the board is connected; The data flow is not started.
  - Beating red heart: the board is connected, and the data flow is running.
  - Static heart: the board is disconnected or crashed.
- LiveTune version:
  - Shows the LiveTune version number.
- UART bandwidth:
  - Show the real-time transmission speed.
    The maximum theoretical speed is 100 Kbyte/s for a UART at 921 600 bauds, which is the default value.
    The color turns to red when the bandwidth becomes close to the limit.
- CPU load:
  - Shows the real-time load of the board CPU.
- LiveTune setting access.

Figure 14. **LiveTune header and status bar**



## 6.5 Start the data flow

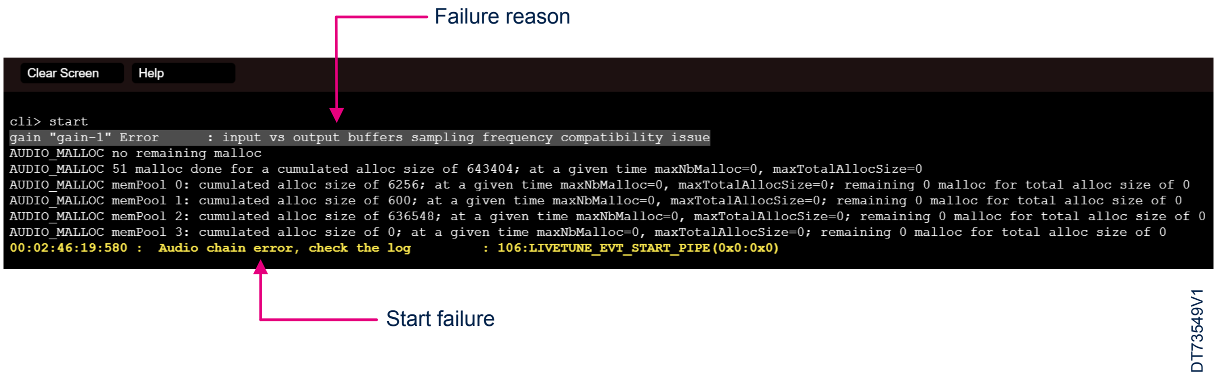
When the audio flow is designed, the user can run it and check the processing result. This is done by pressing the [**Start**] button. If the data flow is correctly configured, the heartbeat icon toggles red and the page header turns to green.

The user has then the possibility to

1. Modify any parameters according to a given use case.
2. Check the real-time processing using the board microphones, the loudspeaker, the USB, or other sources and sinks.

However, sometimes, the data flow exhibits incompatible capabilities or settings that prevent it from starting. In such a case, the heart remains white. The log traces inside the terminal can help to understand the reason of the failure. Refer to the reference manual to get more information and methods to fix a start failure.

Figure 15. **Start failure example**



```
cli> start
gain "gain-1" Error       : input vs output buffers sampling frequency compatibility issue
AUDIO_MALLOC no remaining malloc
AUDIO_MALLOC 51 malloc done for a cumulated alloc size of 643404; at a given time maxNbMalloc=0, maxTotalAllocSize=0
AUDIO_MALLOC memPool 0: cumulated alloc size of 6256; at a given time maxNbMalloc=0, maxTotalAllocSize=0; remaining 0 malloc for total alloc size of 0
AUDIO_MALLOC memPool 1: cumulated alloc size of 600; at a given time maxNbMalloc=0, maxTotalAllocSize=0; remaining 0 malloc for total alloc size of 0
AUDIO_MALLOC memPool 2: cumulated alloc size of 636548; at a given time maxNbMalloc=0, maxTotalAllocSize=0; remaining 0 malloc for total alloc size of 0
AUDIO_MALLOC memPool 3: cumulated alloc size of 0; at a given time maxNbMalloc=0, maxTotalAllocSize=0; remaining 0 malloc for total alloc size of 0
00:02:46:19:580 :  Audio chain error, check the log         : 106:LIVETUNE_EVT_START_PIPE(0x0:0x0)
```

When the design is completed, the user can enable the auto_start flag and use LiveTune in a demonstration mode.

The auto_start flag informs the board to start the data flow embedded in the persistent storage immediately at the boot.

The user can enable or disable the auto_start flag using the following CLI terminal commands:

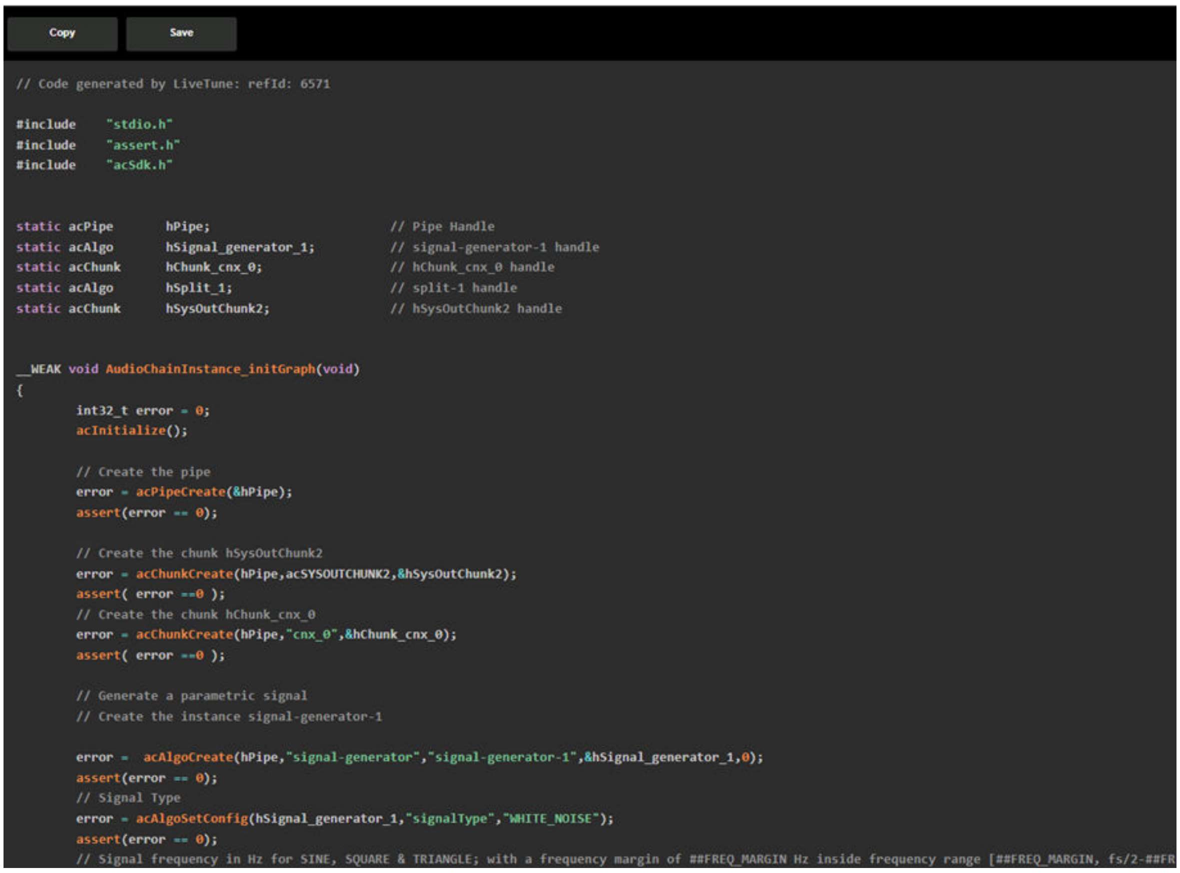
- `cli>ac set auto_start`: enables the auto start.
- `cli>ac clear auto_start`: disables the auto start.

# 7 Prepare the release and extra code integration

When the design is done and tuned, the user might want to run it without the designer. At this development step, the code to manage the designer is no longer mandatory and only the minimum code can be kept to generate the design. LiveTune allows the automatic generation of this code using the button [**Generate code**]. LiveTune then shows a preview of the code generated in another browser tab.

Use the button [**Copy**] to copy the source code in the clipboard and the button [**Save**] to save the source code in a project folder. It is then possible to link this file to the project C code to generate a standalone executable.

Figure 16. **Source code preview**



```
// Code generated by LiveTune: refId: 6571

#include    "stdio.h"
#include    "assert.h"
#include    "acSdk.h"


static acPipe       hPipe;                      // Pipe Handle
static acAlgo       hSignal_generator_1;        // signal-generator-1 handle
static acChunk      hChunk_cnx_0;               // hChunk_cnx_0 handle
static acAlgo       hSplit_1;                   // split-1 handle
static acChunk      hSysOutChunk2;              // hSysOutChunk2 handle


__WEAK void AudioChainInstance_initGraph(void)
{
        int32_t error = 0;
        acInitialize();

        // Create the pipe
        error = acPipeCreate(&hPipe);
        assert(error == 0);

        // Create the chunk hSysOutChunk2
        error = acChunkCreate(hPipe,acSYSOUTCHUNK2,&hSysOutChunk2);
        assert( error ==0 );
        // Create the chunk hChunk_cnx_0
        error = acChunkCreate(hPipe,"cnx_0",&hChunk_cnx_0);
        assert( error ==0 );

        // Generate a parametric signal
        // Create the instance signal-generator-1

        error =  acAlgoCreate(hPipe,"signal-generator","signal-generator-1",&hSignal_generator_1,0);
        assert(error == 0);
        // Signal Type
        error = acAlgoSetConfig(hSignal_generator_1,"signalType","WHITE_NOISE");
        assert(error == 0);
        // Signal frequency in Hz for SINE, SQUARE & TRIANGLE; with a frequency margin of ##FREQ_MARGIN Hz inside frequency range [##FREQ_MARGIN, fs/2-##FR
```

User-dependent pieces of code such as control callbacks are not generated. However, LiveTune generates empty callbacks to make the work of the developer easier while implementing the final application.

For example, the spectrum algorithm exposes a control callback that provides the spectrum calculation. In the designer context, the spectrum values are sent to LiveTune to draw a graph. Upon code generation, such callbacks are generated with an empty body.

The sequence to follow is:

1. Save the code generated in the file `src/audio_chain_last_code_generated_MyUC.c`.
2. In the IDE, clone the configuration *Devel_code_generated_echo* as *Devel_code_generated_myUC*.
3. In the project, replace the file `audio_chain_generated_echo.c` with the file `audio_chain_last_code_generated_MyUC.c` and rebuild the project.
4. Add the own specific code.

*Note: By default, the firmware runs on a standalone target with an optimization for the speed. However, it is still possible to use command lines to tune the use case further:*

- *From a serial terminal:*
  1. *Press the **Enter** key to enter in the CLI mode*
  2. *Enter `help` to show all the available commands*
- *From LiveTune, using its tuner mode:*
  – *Use the graphical user interface that allows the modification of parameters values only (no design capabilities)*

# 8 Build the final release

The release is the final firmware, which must run with the minimum memory and the maximum speed.

To prepare the release:

1. Clone the configuration *Release_code_generated_echo* for example as *Release_code_generated_MyUC*.
2. Generate the code and add it to the project.
3. Add the own specific code.
4. Rebuild the project.

The firmware runs the use case in the standalone mode with a speed optimization and a minimum footprint. There is not terminal and no development trace.

## Revision history

Table 4. **Document revision history**

| Date | Revision | Changes |
|---|---|---|
| 29-Jan-2024 | 1 | Initial release. |

## Contents

# List of tables

## List of figures