User Guide

DTS® Audio Processing ARM® / Android™ v1.x

Version: 1.0.x

Status: Released

Date: November 22, 2017

Legal Notices

CONFIDENTIAL

COPYRIGHT AND TRADEMARK

NO WARRANTY

Version History

Version	Date	Status & Description
1.0.x	November 22, 2017	Initial release of the User Guide

Table of Contents

List of Figures

List of Tables

Acronyms and Abbreviations

Acronym	Expansion
APK	Android Package Kit
SDK	Software Development Kit
AOSP	Android Open Source Project
DSP	Digital Signal Processing
GEQ	Graphic Equalizer
USB	Universal Serial Bus
API	Application Programming Interface
EULA	End User License Agreement
kHz	Kilohertz
32b	32-bit
64b	64-bit

Related Resources

Item	Description
Platform User Guide	"UG_DTS_Audio_Android_Platform_Integration", provides the user with information regarding the platform (AOSP) integration.

1 Overview

DTS Audio Processing is an essential audio processing solution that is designed to improve the audio quality for small or micro speaker devices with two speakers. It maximizes the bass, loudness and clarity on both the headphone and speaker paths while minimizing distortion and providing digital signal protection.

This implementation has been optimized to run on Android based mobile devices, utilizing one or more ARM cores. A full feature list, and description of the features are provided in Section 6, but the key features are summarized below.

1.1 Key Features

The product contains the following key features:

- Tuning for Symmetrical and Asymmetrical Speakers
- Enhanced loudness maximization and speaker protection
- Micro-speaker distortion prevention
- Enhanced bass response with improved harmonics for speaker and headphone routes
- High Resolution Audio and extended bit depth support
- In-box headphone tuning
- 3 configurable content/environmental modes
- Stereo Preference for Headphones and Speakers
- Dialog Enhancement
- T.U.N.E. tuning application to enable custom device tuning and reduce integration time.

2 System Architecture Overview

The product is composed of two basic components: a platform and an application, as illustrated below in Figure 1 below.

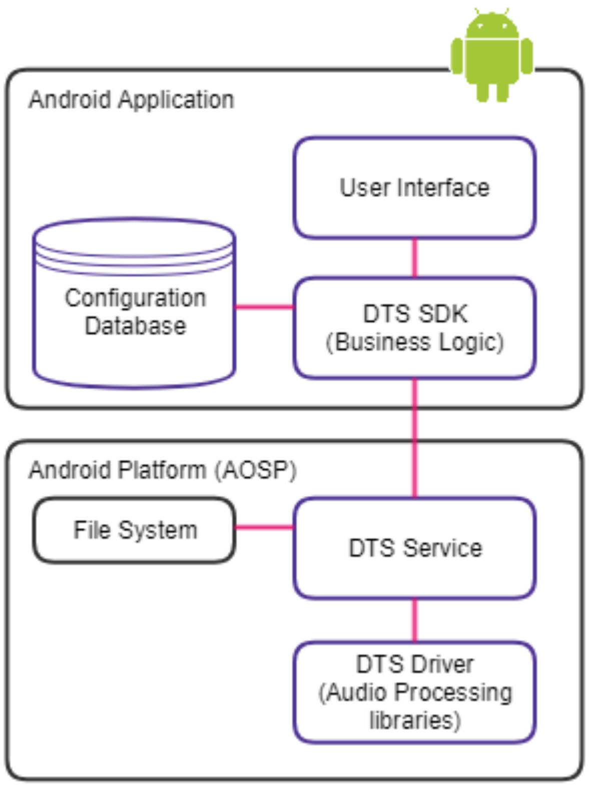


Figure 1: High-level system architecture

Within this structure, the following product components work together to enable the DTS audio effect to be applied to any audio stream being processed by the Android Media framework.

- An **Android application** is the primary user interface and initializes, configures, and queries the underlying DTS core processing library.
- The java **DTS SDK** in turn uses a service to communicate between the application layer and the underlying system components. The SDK is the interface between the public and private APIs and contains the DTS business logic.
- The **DTS Service** is responsible for maintaining the product state and persistence. It starts at boot time, and continues to run in the background to ensure that the irrespective of the state of the application, audio processing is always enabled.
- The **DTS Driver** contains the DTS core processing SDKs, which are delivered as a set of audio processing libraries integrated into AudioFlinger. These core libraries enable DTS processing to be applied to any audio streams playing on the device.

- The application includes a local **Configuration Database** which contains all the device configuration and tuning data.

Note: This user guide focuses on the Android Application components, while a separate Platform User Guide (UG_DTS_Audio_Android_Platform_Integration) describes the installation, integration and configuration of the Platform components.

3 Integration Overview

The DTS Audio Processing ARM / Android system solution consists of a number of components spanning many layers in the Android system.

3.1 Platform level integration

At the platform level, DTS provides a reference integration for the Android Open Source Project (AOSP) that has been tested on a Pixel C device. The platform components can be found under the “platform” folder in the release package and include open-source components, which need to be merged with the OEM’s Android source tree and precompiled binary components, which need to be added into the device image.

Inside the proprietary/doc folder you will find the Platform User Guide (UG_DTS_Audio_Android_Platform_Integration.pdf), which describes in detail how to integrate the platform components of the DTS system.

You will also find Release Notes (RN_DTS_Audio_Android_Platform_Integration.pdf) for the platform component.

The first thing a Licensee needs to do is to integrate the platform components into their own Android build before continuing with application level integration.

3.2 Application integration

At the highest level, a Java SDK is provided that a Licensee will need to integrate into their own application, for their own device. The application requires that the platform components have been integrated before it can successfully configure the underlying core audio processing libraries.

The Application level components can be found under the folder named “application” in the release package.

3.2.1 Android User Interface application

A reference user interface is provided to enable the Licensee to reduce their integration time. The application can be used as-is, or may be modified by the Licensee and either delivered by them as a stand-alone application, or as a “system” application integrated into a Licensee’s audio settings menu.

The user interface can also be used as a learning aid to help the Licensee better understand how to use public APIs exposed by the SDK.

Note: The SDK and the reference application will only work on a system that has all the DTS platform components integrated.

3.2.2 Installing the pre-build application

A pre-built application (dts-audio-processing-ui-<device><region>-release.apk) can be found in the application/bin directory and can be used as part of development to debug potential problems and validate the basic operation of the DTS Audio Processing product.

To install the prebuilt reference application, it's recommended to use either the "`install_demo.sh`" script at the top level of the release package or the the "`install_app.sh`" script in the "application" folder in the release package.

3.2.2.1 install_demo.sh

The `install_demo.sh` script installs both the DTS Service Platform component and the reference application, and is mainly targeted to be used for quickly setting up e.g. a demo or development DTS device with the latest version of these components.

(The `install_demo.sh` script essentially runs the two subscripts "`proprietary/platform/vX.X.X.X.X/service/install-scripts/install_service.sh`" followed by "`proprietary/application/install_app.sh`")

3.2.2.2 proprietary/application/install_app.sh

The `install_app.sh` installs the application on a device that has previously completed the platform integration. It will first uninstall any old version of the application before installing the new one. It will also complete a license check, as well as remove the current system settings file from the device to avoid possible data mismatch errors, in cases where the offline (configuration) database in a new release has changed significantly from a prior installation.

3.2.2.3 proprietary/platform/vX.X.X.X.X /service/install-scripts/install_service.sh

The `install_service.sh` installs the Service components on a device that that has previously completed the platform integration. It will first uninstall any old version of the service before installing the new one. It will also do a license check, as well as remove the current system settings file from the device.

4 Building the User Interface Application

To build the application with the DTS SDK, please refer to the official Android Studio build guide at https://developer.android.com/tools/building/building-studio.html.

Note: Android Studio 3.0+ is recommended.

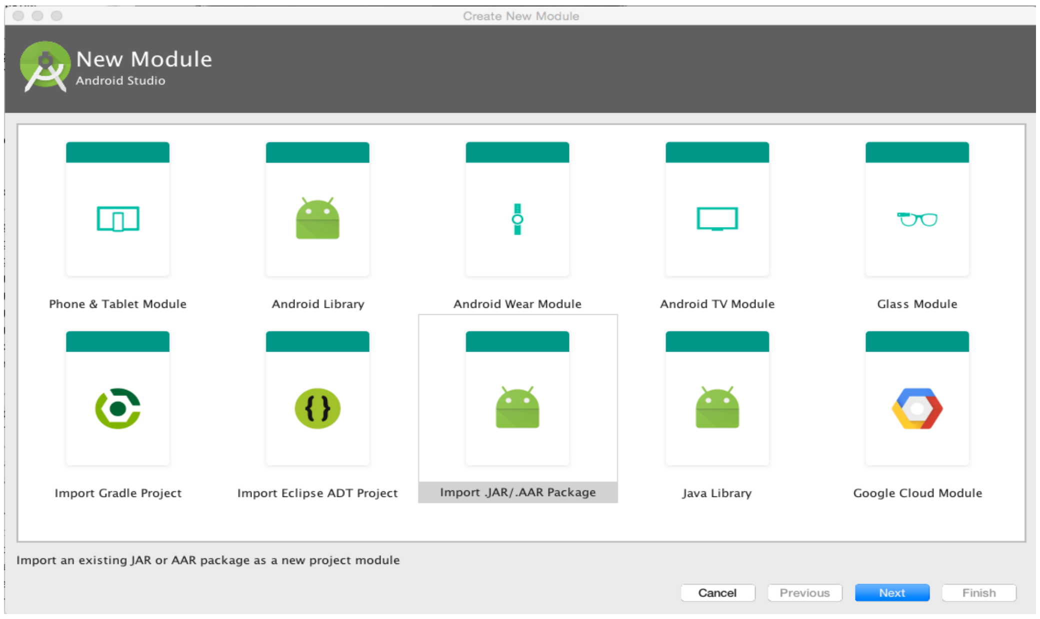
4.1 SDK integration with Android Studio

This section will walk through how to integrate the DTS SDK to the Licensees own application using Android Studio version 2.3.3. or higher.

4.1.1 Add the DTS SDK as a new module

From Android Studio, click "*File*" → "*New* → "*New Module*"

Select "*Import .JAR or .AAR Package*" as the module type and press "*Next*".



Import an existing JAR or AAR package as a new project module

Figure 2: Android Studio New Module dialog.

Set the location of the `dts-sdk-release.aar` file using the browse menu. The "*Subproject name*" field should be automatically set to "`dts-sdk-release`". Press "*Finish*".

Check that the module has been added successfully. It should appear in the Project explorer menu as below:

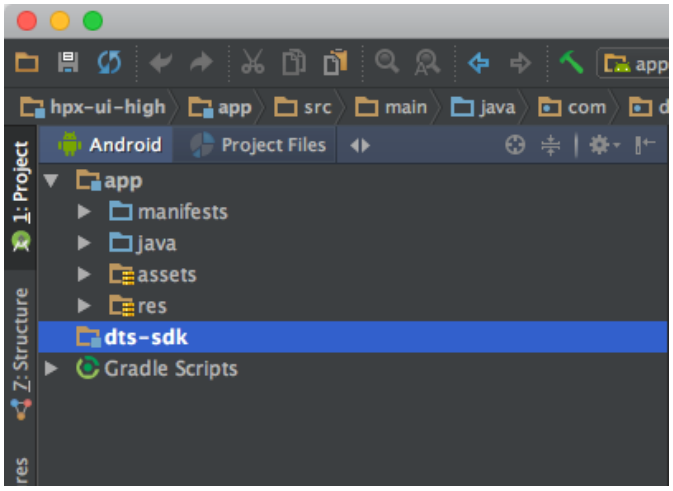
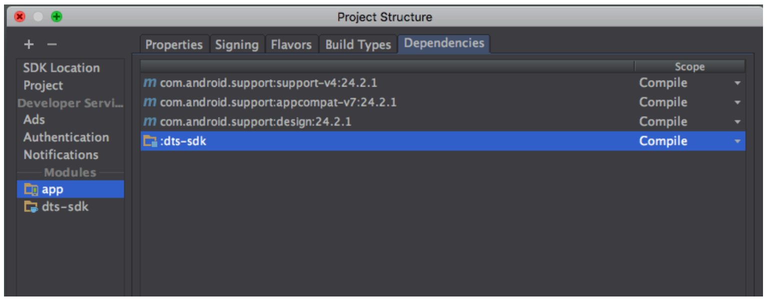


Figure 3: Check that SDK module is added to Android Studio project.

4.1.2 Add the DTS SDK module as a dependency.

Click “*File*” → “*Project Structure…*”. Select the “*Dependencies*” tab.

Click the “+” button in the bottom left corner and select “*Module dependency*”.


Project Structure

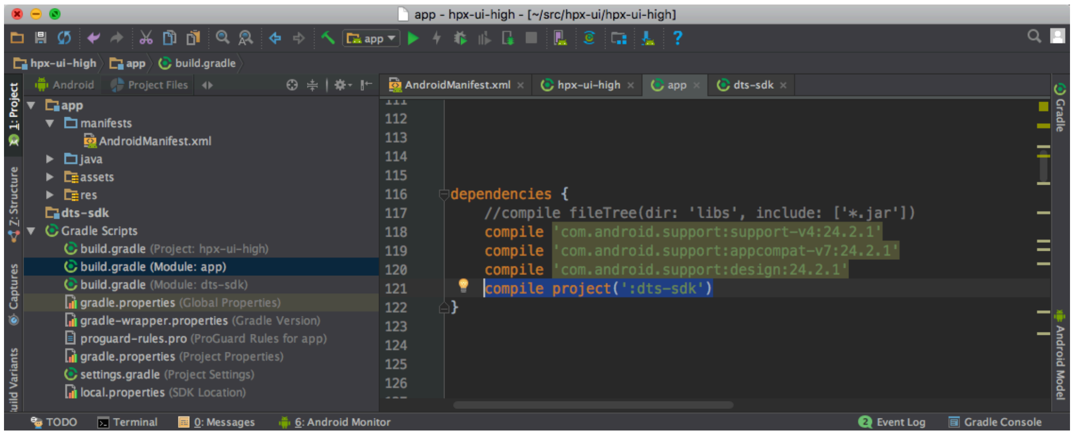
	Scope
com.android.support:support-v4:24.2.1	Compile
com.android.support:appcompat-v7:24.2.1	Compile
com.android.support:design:24.2.1	Compile
:dts-sdk	Compile

Figure 4: *Add module dependency in Android Studio.*

A “*Choose Modules*” window will appear. Select “*dts-sdk-release*” and press “*OK*”

The “`dts-sdk-release`” module should now be added as one of the dependencies. Press “*OK*” to exit the Project Structure window.

The above steps can be verified by checking the `build.gradle` file of the application. “`dts-sdk-release`” should be listed there as one of the dependencies as shown below.


app - hpx-ui-high - [~/src/hpx-ui/hpx-ui-high]

```
dependencies {
    //compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:24.2.1'
    compile 'com.android.support:appcompat-v7:24.2.1'
    compile 'com.android.support:design:24.2.1'
    compile project(':dts-sdk')
}
```

Figure 5: Module Dependency Added Correctly.

4.1.3 Copy the configuration database, EULA and third-party assets

Copy the folder `application/src/dts-audio-processing-ui/app/src/main/assets`

to `app/src/main folder of your own application`

As an example, execute the following command using command line:

```
cp -r application/src/dts-audio-processing-ui/app/src/main/assets/
<CODE_LOCATION>/MySampleApplication/app/src/main/assets
```

Check that it was added correctly to you project.

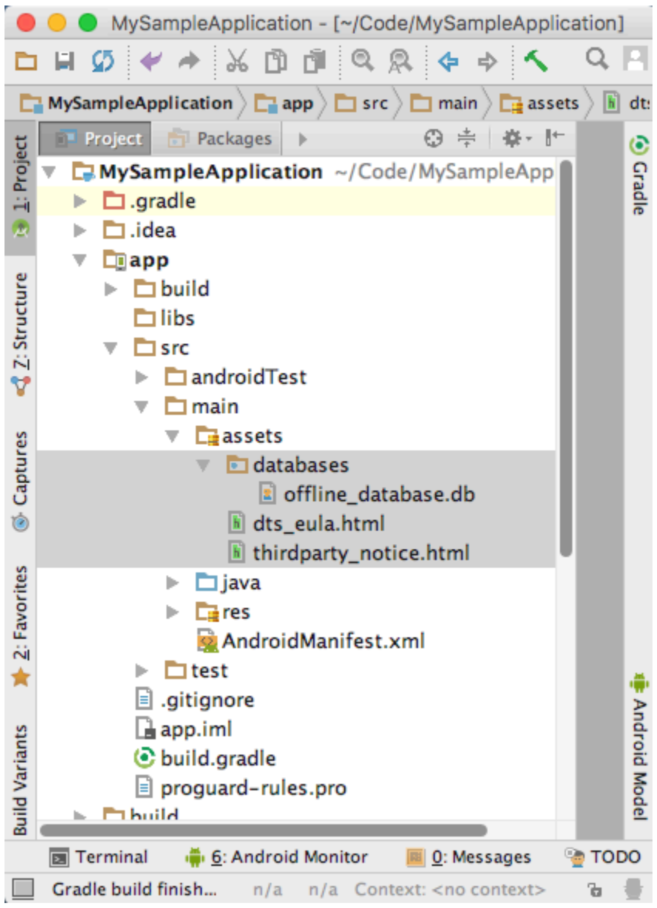


Figure 6: Check that assets are added to Android Studio project.

Note: The configuration database in the release package is an example database. OEMs will need to get their own database for use in a final product.

4.2 Updating the DTS SDK

To update the DTS SDK, simply drag and drop the "`dts-sdk-release.aar`" file into the dts-sdk-release directory (as shown in Section 4.1.1) to overwrite the existing .aar file. Perform a "*clean*" and the project should be updated.

5 DTS SDK legal requirements

This section describes how to use the DTS SDK. Detailed API information, along with the DTS Result class that is the return type for most functions, may be found in the DTS SDK documentation included with the release in this location:
`proprietary/application/doc/api/index.html`

5.1 Licensee legal requirements

5.1.1 Privacy Policy

DTS requires that a licensee display certain privacy policies through the end application. Please contact your DTS representative or Licensee Manual for details of these requirements.

5.1.2 End User License Agreement (EULA)

DTS requires that a Licensee incorporates the DTS End User License Agreement (EULA) into their own EULA or Terms of Service if they are using the provided user interface application. A copy of the EULA can be found in the application source.

6 SDK Features

This section outlines the individual product features and documents how to query and access them via the public DTS SDK APIs.

6.1 Full Feature List

The product supports the following list of features shown Section 6.1.1 through Section 6.1.2 and is described fully in the following sections. The only features that can be configured by the DTS application are the UI controls. Section 6.2 to Section 6.7 document how the Licensee application engineer can interact with the features.

6.1.1 Audio Features

The following audio features have been implemented in the product.

- Tuning for Symmetrical and Asymmetrical Speakers
- Enhanced loudness maximization and speaker protection
- Micro-speaker distortion prevention
- Enhanced bass response with improved harmonics for speaker and headphone routes
- Dialog Enhancement
- Signal protection for the headphone audio routes

6.1.2 Platform integration features

The following features are available to support the individual platform integration.

- High Resolution Audio and extended bit depth support
- Configurable audio output routes
- USB-C Headphone support
- Supported input and output channel layouts
- Configurable audio routes
- License Configuration

6.1.3 UI Controls exposed in the DTS Application

The following UI controls are exposed to an end user of the device (Consumer).

- On/Off Control
- 3 configurable content/environmental modes
- Dialog Boost
- Bass Boost
- Treble Boost
- 5 band GEQ
- Stereo Preference for Headphones and Speakers

The remainder of this section describes the public APIs and how to initialize, query and manage these Consumer facing features.

6.2 Initialization

Before any other features are queried, the DTS SDK and underlying driver must first be initialized. It is safe to initialize the driver multiple times, and has no effect for redundant calls.

In the DTS application, initialize is called in the splash screen to both initialize DTS, and to check if the service is installed properly and is responsive.

The initialize function will return an error code (via `DtsResult` object) if it detects that the service is unresponsive.

```
DtsResult com.dts.dtssdk.DtsManager.initialize(@NonNull Context
context)
```

All remaining features are created and managed using the `DtsManager` class.

6.3 On/Off Control

The Product contains an On/Off control that enables a Consumer to enable or disable the audio processing. Note that even when the technology is off, speaker tuning and protection remain activated to ensure that device speakers are protected. Limiting and downmixing are also always enabled to ensure that the consumer will always hear the audio signal.

Turning DTS processing on and off is done using the following API from the `DtsManager` class:

```
DtsResult com.dts.dtssdk.DtsManager.setDtsEnabled(Boolean enabled)
```

If the value of the enabled parameter is true, DTS processing will be turned on.

To turn DTS off, the same API is called but with the value of enabled set to false.

The following code example demonstrates how to use DTS on/off with DtsManager.

```
// *** Get the DTS status ***

DtsResult<Boolean> resultBool =
DtsManager.getInstance().getDtsEnabled();

// Note that the getDtsEnabled() function will return a

// DtsResult<Boolean> object that contains a Boolean value indicating
// whether or not DTS is enabled plus few other helpful functions to

// see if DTS is running properly.

// Check if DTS is working

if (resultBool.isResultOk()) {

    // DTS is working.

    // Get DTS ON/OFF status

    boolean DtsEnabled = resultBool.getData();

} else {

    // Get DTS returned error

    int resultCode = resultBool.getResultCode();

    String resultMessage = resultBool.getResultMessage();

    // Handle error

}

// *** Set DTS ON/OFF ***

// Set DTS ON (true for ON; false for OFF)

DtsResult result = DtsManager.getInstance().setDtsEnabled(true);

// setDtsEnabled() function will return a DtsResult object that

// contains the result code and result message.

// Check if setDtsEnabled() was successful

if (result.isResultOK()) {

    // Set DTS ON received successfully

} else {
    // Set DTS ON returned error

    int resultCode = resultBool.getResultCode();

    String resultMessage = resultBool.getResultMessage();

    // Handle error

}
```

6.4 3 configurable content/environmental modes

The Product can store up to three tuning modes in addition to the default. This enables the Licensee to create custom independent tunings for internal speaker and headphone routes that are optimized for different content and/or environmental modes. The modes are labelled mode 1 through mode 3 and can be mapped by the Licensee to an appropriate name in their reference application.
Although only three modes are presented, the Licensee can tune two versions of each mode; one for all headphone routes and one for the internal speaker route.

The product contains source code in the provided reference application, which shows the Licensee how to set, and query the currently active speaker mode. The modes provided in the reference application are Music, Movies and Games. The following code examples will demonstrate how to use the Content Mode Selection API with `DtsManager` class.

The examples use the `CountDownLatch` class to synchronize the asynchronous methods to make the code more readable. It also uses class member variables to pass data from the callback to the main thread of execution.

```
  class Foo {

      DtsResult mDtsResult;

      List<ContentMode> mContentModeList;

      …
```

Also for readability the example code uses asserts instead of if statements for checking of results and the method `getContext()` is assumed to return the current context.

6.4.1 Set content mode

```
  Log.d(TAG, "testSetContentMode" );

  final CountDownLatch countDownLatch1 = new CountDownLatch(1);

  // Query for all content modes

  DtsManager.getInstance().getAllContentModes(getContext(),new
  QueryCallback<ContentMode>() {
      @Override

      public void onQueryComplete(DtsResult result,List<ContentMode>
  mContentModelist) {

          mDtsResult = result;

          countDownLatch1.countDown();

      }

  countDownLatch1.await(); // Wait for query to finish

  assertTrue(mDtsResult.isResultOk());

  assertFalse(mContentModeList.isEmpty());

  // Choose the first content mode in the list for this example

  ContentMode theChosenContentMode = mContentModeList.get(0);

  final CountDownLatch countDownLatch2 = new CountDownLatch(1);


  // Set the content mode

  DtsManager.getInstance().setContentMode(getContext(),
  theChosenContentMode, new OnCompleteCallback() {

      @Override

      public void onComplete(DtsResult result) {

          mDtsResult = result;

          countDownLatch2.countDown();

      }

  });

  countDownLatch2.await(); // Wait for selection to finish


  assertTrue (mDtsResult.isResultOk());
```

Note that the names and images returned from the SDK will be generic, e.g. “Attached2 – Content Mode (mode 1)” etc. It is not expected these names should be displayed to the user but the Licensee shall override them with their own custom defined name and images for each content mode, e.g. Music, Sport etc.

6.5 Boost Controls

6.5.1 Dialog Boost

The User Interface in the provided application includes a vocal or dialog boost user control to allow a Consumer to enable or disable additional gain being applied to the vocals so that they are easier to hear in noisy environments, or in cases where the content being played back has a high dynamic range.

6.5.2 Bass Boost

The User Interface in the provided application includes a bass boost user control to allow a Consumer to enable or disable additional bass boost on both the internal speaker and headphone audio paths. This user bass control is additive to the bass tuning provided as part of any given content mode.

6.5.3 Treble Boost

The User Interface in the provided application includes a treble boost user control to allow a Consumer to enable or disable additional gain being applied to the mid-high frequency range on both the internal speaker and headphone audio paths.

The above three controls Treble, Dialog and Bass boost levels can be enabled/disabled using the treble, dialog and bass APIs offered in `DtsManager` class.

Here is an example for how to get the current treble boost, dialog enhancement and bass boost level state

```
DtsResult<Boolean> resultTreble =
DtsManager.getInstance().getTrebleBoostEnabled();


if (resultTreble.isResultOk()) {

    // The treble boost level state will always be either true or
false

    boolean trebleBoostEnabled = resultTreble.getData();

}

DtsResult<Boolean> resultDialog =
DtsManager.getInstance().getDialogEnhancementEnabled();


if (resultDialog.isResultOk()) {
```

```
//The dialog enhancement level state will always be either true or
//false

    boolean dialogEnhancementEnabled = resultDialog.getData();

}

DtsResult<Boolean> resultBass =
DtsManager.getInstance().getBassBoostEnabled ();


if (resultBass.isResultOk()) {

    // The bass boost level state will always be either true or false

    boolean bassBoostEnabled = resultBass.getData();

}
```

And examples for how to enable treble boost, dialog enhancement and bass boost levels

```
// Enable or disable treble boost
DtsResult result =
DtsManager.getInstance().setTrebleBoostEnabled(mContext,true);
if (result.isResultOk()) {
     // Setting treble boost was successful

}



// Enable or disable dialog enhancement
result =
DtsManager.getInstance().setDialogEnhancementEnabled(mContext,true);
if (result.isResultOk()) {
     // Setting dialog enhancement was successful

}




// Enable or disable bass boost
result = DtsManager.getInstance().setBassBoostEnabled(mContext,true);
if (result.isResultOk()) {
     // Setting bass boost was successful

}
```

When treble boost is disabled (which is default), the GEQ slider will show -12dB to +12dB range. If treble boost is enabled explicitly in customer configuration file, the GEQ GUI will show the treble boost button and the GEQ bands will only have a range of -10dB to +10dB. In Customer configuration file, the option `disable_treble_level = false` is used to enable the treble boost. Bass boost is disabled by default unless and until customer configuration file enables using `disable_bass_level = true` option. Dialog enhancement is disabled by default unless and until customer configuration file enables using `disable_dialog_level = true` option.

6.6 5 band GEQ

The product includes a 5-band graphic equalizer so that a Consumer can have further control over the color of the output audio signal. The GEQ can be enabled or disabled via the user interface and provides gain/attenuation of between ± 10dB per band or ± 12dB, depending on whether additional treble boost controls are available to the Consumer.

There are five bands of Graphic Equalization provided. They are:

- Band 0 (100Hz).
- Band 1 (300Hz).
- Band 2 (1kHz).
- Band 3 (3KHz).
- Band 4 (10KHz)

Gains for each band may be set individually, or all five may be set at once. The latter functionality allows for features such as presets or a reset button to be included in the application. The GEQ can be turned on or off via a master switch. The API calls for this functionality are as follows:

- `DtsResult setGEQEnabled(boolean enable)`: Turns GEQ on and off
- `DtsResult<Boolean> getGEQEnabled()`: Reads the on/off state of GEQ
- `DtsResult setGEQ5Gain(int band, int gain)`: Sets the gain of the named individual band.
- `DtsResult<Integer> getGEQ5Gain(int band)`: Returns the gain of the named individual band.
- `DtsResult setAllGEQ5Gains(List<Integer> gainList)` or `setAllGEQ5Gains(int [] gainArray)`: Set the gains of all five bands at once.
- `DtsResult<List<Integer>> getAllGEQ5Gain()`: Read the gains of all five bands at once.

Note: Graphic equalization is only applied to the audio when DTS is turned on.

By default, the GEQ is turned off even if DTS is enabled. The user must explicitly enable its use. The on/off state of GEQ is retained going forward, even if DTS is turned off and back on again.

6.7 Stereo Preference for Headphones and Speakers

To deliver a richer and immersive experience for stereo content, the audio renderer for headphone and internal speakers provides three consumer controlled modes which deliver the following audio effects;

- *Wide mode* widens the sound stage for the audio coming out of the speakers, while maintaining a neutral timbre. For headphones, this essentially pulls the audio outside of the head which can lead to lower listener fatigue. For internal speakers, this provides the listener with a more immersive experience as the audio is perceived to be coming from external speakers further out to the left and right of the listener

- *Front mode* moves the audio closer to the center of the device, narrowing the sound field to mimic playing the content back over two high quality stereo speakers placed in front of the consumer.

- *Traditional mode* removes any widening or narrowing of the sound stage and is provided as a reference for traditional audio playback.

The API provides set and get functions for the user to choose the Stereo Mode they prefer. The example code below provides examples of how to use the APIs.

```
// Get current stereo preference setting

DtsResult<StereoPreference> resultStereoPreference =
DtsManager.getInstance().getStereoPreference();


// Check that the system is running correctly, and if so, get the

// StereoPreference

if (!resultStereoPreference.isResultOk()) {

    // getStereoPreference() returned error. Handle error here

} else {

    StereoPreference = resultStereoPreference.getData();

}


// Set stereo preference to StereoPreference.WIDE

// Note that the parameter in setStereoPreference () must be either

// WIDE, FRONT or TRADITIONAL. Passing StereoMode.UNKNOWN will result
// in an error
```

```
DtsResult result=
DtsManager.getInstance().setStereoPreference(StereoMode.WIDE);


// Check that the system processed the set request correctly

if (!result.isResultOk()) {

    // setStereoPreference() returned error. Handle error here

}
```

The remainder of this section provides a brief description of the features that are not configurable via the DTS applciation and is provided as a reference to the reader.

6.8 Audio Feature Descriptions

6.8.1 Tuning for Symmetrical and Asymmetrical Speakers

The product contains several technologies that allow the device internal speakers to be tuned to compensate for unwanted audio characteristics such as non-ideal and/or asymmetric spectral response, reduction in the overall audio clarity, and positioning of the sound stage relative to the speaker position. This version includes a proprietary Gain-Phase EQ (GPEQ).

6.8.2 Enhanced loudness maximization and speaker protection

The product includes a new loudness maximization and speaker protection algorithm based on a time domain multi-band dynamic range compression and hard-limiter coupled with additional distortion mitigation via additional signal processing. The feature includes a configurable 3-band dynamic range compressor and wide-band limiter. This generation offers improvements in signal envelope estimation and dynamics processing.

It is designed for small or micro speaker devices that are typically over-driven to derive the maximum possible perceived loudness.

6.8.3 Micro-speaker distortion prevention

The product includes a new proprietary distortion mitigation algorithm based on narrow band signal suppression that is designed for small or micro speaker devices that are typically over-driven to derive the maximum possible perceived loudness. The individual parameters and configuration is considered DTS proprietary IP and so not exposed to the Licensee.

6.8.4 Enhanced bass response with improved harmonics for speaker and headphone routes

The product delivers an enhanced bass response with improved harmonics generation for both mono or stereo content in the 500Hz - 1kHz range. This product introduces further dynamics controls and support for a wider range of micro-speakers and exposes controls to enable a tuning engineer to vary the amount, and shape, of the harmonic content that's generated relative to the original. The resulting algorithm is more flexible and robust.

Finally, the bass enhancement is now available on both the internal speaker and headphone audio routes when enabled by a tuning engineer.

6.8.5 Dialog Enhancement

The product enables Consumers and Licensees to enhance the perceived volume and clarity of dialog contained within stereo and multichannel content, while leaving the ambient content in the signal unchanged. This is achieved by identifying dialog information, applying proprietary enhancement and then mixing the dialog back into the original signal while maintaining overall signal quality.

6.8.6 Signal protection for the headphone audio routes

The product improves the end-point signal protection for the headphone audio routes by adding support for a hard limiter. This has been introduced to manage audio signal saturation due to Consumer or Licensee introduced gains. The feature uses the same underlying limiter algorithm used on the internal speaker path but with added support for high sample rates when the product is disabled.

6.9 Platform integration features

The following features are available to support the individual platform integration.

6.9.1 High Resolution Audio and extended bit depth support

To enable various High-Resolution audio use cases, the product supports processing at sample frequencies up to 192kHz at an output bit depth of up to 32 bits. To balance processor load and flexible sample rate support, DTS has employed power-efficient multirate algorithms.

The following sample frequencies and channel layouts are supported.

Table 1: Supported Sampling Frequencies.

Audio Route	Input Sampling Frequency (kHz)	Output Sampling Frequency (kHz)	Channel Layout
Internal Speaker	44.1	-	-
	48	**48**	2.0, 5.1, 5.1.2, 7.1
	88.2	-	-
	96	96	2.0
	176.4	-	-
	192	192	2.0
Line-out	44.1	-	-
	48	**48**	2.0, 5.1, 5.1.2, 7.1
	88.2	-	-
	96	96	2.0
	176.4	-	-
	192	192	2.0
Bluetooth	44.1	**44.1**	2.0, 5.1, 5.1.2, 7.1
	48	48	2.0, 5.1, 5.1.2, 7.1
	88.2	88.2	2.0
	96	96	2.0
	176.4	176.4	2.0
	192	192	2.0
USB-C	44.1	44.1	2.0, 5.1, 5.1.2, 7.1
	48	48	2.0, 5.1, 5.1.2, 7.1
	88.2	88.2	2.0
	96	96	2.0
	176.4	176.4	2.0
	192	192	2.0

Note 1: The items in **bold** in Table 1 above show the Android default output sampling frequencies. To fully support high resolution audio, the audio endpoint must report itself as being capable of supporting the desired frequency. The platform behavior is to re-sample content prior to DTS audio processing to match the sample rate capabilities of the audio endpoint.

6.9.1.1 Input/output Bit Depth

The following input and output bit depths are supported.

Table 2: Supported input and output bit-depth.

Input bit depth	Word	Endian
32 bit float	32 bit	native
16 bit integer in a 16 bit word	16 bit	native
24 bit integer packed in a 32 bit word	32 bit	native
Output bit depth		
32 bit float	32 bit	native
32 bit integer	32 bit	native
24 bit integer packed in a 32 bit word	32 bit	native
16 bit integer in a 16 bit word	16 bit	native

6.9.2 Configurable audio output routes

To provide increased flexibility and facilitate ease-of-integration, a Licensee can configure the reference application and SDK to enable or disable audio processing on the Bluetooth, line-out and USB-C audio route.

6.9.3 USB-C Headphone support

The product enables Licensees to apply audio processing on the USB-C audio endpoint, so that they can support Consumers USB-C headphones and wired headphones connected via a USB-C adaptor.

6.9.4 Supported input and output channel layouts

To facilitate the playback of channel based audio content the product contains a multichannel downmixer which enables the following;

- Down-mix multichannel content to stereo before further processing is applied on the internal speaker audio route.

- Down-mix multichannel content to stereo before further processing is applied on any headphone (line-out, Bluetooth, or USB-C) audio route.

When the SDK is in bypass mode on either internal speaker or stereo headphone audio endpoints, multichannel audio is first down-mixed to stereo and then passed through a limiter to protect the audio endpoint.

The following input channels and resulting output channels are shown for clarity.

Table 3: Input channel layout and resulting output.

Input channel format	Resulting audio output
2.0	2.0
5.1	2.0 stereo downmix prior to additional audio processing over the headphone or internal speaker paths
7.1	2.0 stereo downmix prior to additional audio processing over the headphone or internal speaker paths

6.9.5 Configurable audio routes

To provide increased flexibility and facilitate ease-of-integration, a Licensee can configure the reference application and SDK to enable or disable audio processing on the Bluetooth, line-out and USB-C audio route.

6.9.6 License Configuration

To support Licensee integration requirements, the product allows the Licensee to configure the license path (the directory containing the DTS product licenses) to make the integrations more flexible. The license files are queried and validated upon device boot before any audio processing occurs. If the product is unlicensed, then all processing is bypassed. The core audio SDK is not instantiated and bypassed entirely in AudioFlinger. Further information can be found in the platform user guide.

7 Product Configuration

7.1 Licensed features and routes

To facilitate integration of multiple DTS products into a Licensees device portfolio, the DTS SDK has been designed to a common API across all DTS audio products. This enables the Licensee to support different products from DTS, potentially using the same APK and same SDK to ease integration and maintenance.

The choice of product and features exposed by the DTS SDK is determined by the DTS License key, installed as part of the platform integration.

The trade-off for this flexibility is that certain methods or classes exposed by the SDK API may not be appropriate for the product being integrated. If, as a Licensee, you call a function for a non-licensed feature, you will get the error: `FEATURE_NOT_LICENSED`.

Using the documentation (`proprietary/application/doc/api/index.html`), the integrator can find out what features and audio routes are available, by reviewing the descriptions in the *DtsFeature* and the *util/AudioRoute* classes. At runtime the integrator can use the `DtsFeatureManager` to find out what features/routes are available.

For instance, to find out if the SDK / license supports the treble boost feature you can use the following code:

```
DtsFeatureManager.getInstance().hasFeature(DtsFeature.TREBLE_LEVEL)
```

Additionally, by querying the `DtsFeatureManager:`**`getAvailableFeatures`** method, a full listing of supported features can be retrieved.

Note: This user guide (and associated documentation), only documents the relevant classes and methods for the specific level of the release package, and others will be hidden.

7.2 Customer configuration file

The DTS System has a customer configuration file (`customer.cfg`) that resides in the system level. This enables Licensees to easily customize the product to support custom integrations and use cases.

If a Licensee wishes to modify or hide certain features to provide differentiation across their device portfolio while maintaining a single APK and code repository, this is possible by simply adding a different `customer.cfg` file on each target device.

Customization are queried by both the platform and the User Interface, can range from controlling which audio routes have the DTS audio effects processing applied, to which user controls are exposed at the UI level.

7.2.1 File location

By default, the `customer.cfg` file is located in `/etc/dts/`although this path may be modified by the Licensee integrator as part of the platform integration.

7.2.2 File format

The configuration file is a plain text file, with each line containing a single key-value pair.

Example:

```
disable_route_bluetooth=true

disable_geq_5=false
```

Note: Care must be taken when applying the key-value pairs not to include trailing white spaces after the value. There is a bug in the current release that does not correctly parse the values if there are trailing white spaces.

7.2.3 Supported key value pairs

The following key-value pairs are used by the DTS Platform and/or the application components. The Licensee is free to create and use their own custom key value pairs, that can be read by the SDK to e.g. reconfigure the app accordingly. For more information about the ranges of the controls, please see the T.U.N.E. application documentation.

Key Name	**Possible Values** (Default is Bold)	**Description**
disable_route_bluetooth	true, **false**	**SDK** If true, SDK will abort any set accessory calls for the Bluetooth route. **Platform** If true, platform will disable DTS processing on the bluetooth route.
disable_route_line_out	true, **false**	**SDK** If true, SDK will abort any set accessory calls for the line out route. **Platform** If true, platform will disable DTS processing on the line out route.
disable_route_usb	true, **false**	**SDK** If true, SDK will abort any set accessory calls for the USB route. **Platform** If true, platform will disable DTS processing on the USB route.

Key Name	Possible Values (Default is Bold)	Description
disable_geq_5	true, **false**	**Reference UI** If true, the reference UI will hide the GEQ screen. **SDK** If true, the SDK will abort any 5 band GEQ-related API calls. **Platform** If true, 5 band GEQ will always be turned off. Anything other than "true", 5 band GEQ initial state will be ON.
disable_treble_level	**true**, false	**Reference UI** If true (or not set), the reference UI will hide the treble slider/button. If false, the reference UI will show the treble slider/button. **SDK** If true (or not set), the SDK will abort any treble related API calls. If false, the SDK will enable all treble related API calls. **Platform** If false, GEQ bands will be boosted by system per curve dB. If true, GEQ bands will not be boosted at all.
disable_bass_level	**true**, false	**Reference UI** If true (or not set), the reference UI will hide the bass slider/button. If false, the reference UI will show the bass slider/button. **SDK** If true (or not set), the SDK will abort any bass related API calls. If false, the SDK will enable all bass related API calls. **Platform** If false, bass boost will be applied to Eagle. If true, no bass boost will be applied.
ui_hide_geq_5	true, **false**	**Reference UI** If true, the reference UI will hide the GEQ screen.
setting_speaker_bass_level_delta	0.01 - **1.00**	**Platform** Maximum bass enhancement for the speaker path.
setting_accessory_bass_level_delta	0.01 - **1.00**	**Platform** Maximum bass enhancement for the line out, USB, and bluetooth paths.
setting_speaker_dialog_level_delta	0.01 - **1.00**	**Platform** Maximum bass enhancement for the speaker path.
setting_accessory_dialog_level_delta	0.01 - **1.00**	**Platform** Maximum bass enhancement for the line out, USB, and bluetooth paths.
setting_system_api_timeout	0 – 10000 (default is **3000**)	**SDK** The SDK will abort the system calls to DTS Audio Service when the timeout is expired. The timeout is measured in milliseconds.

7.2.4 SDK api for reading customer configuration settings

Use the `DtsResult<Map<String,String>> DtsFeatureManager:`**`getCustomerConfig()`** to get a hash map of key value pairs defined in the customer configuration file. This can be used as needed.