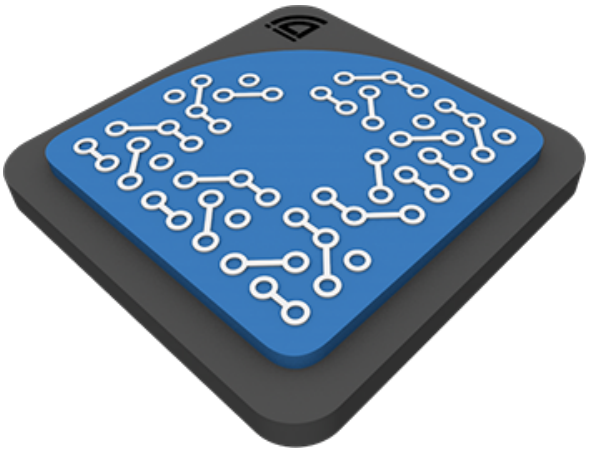
Dante Embedded Platform Programmer's Guide

For Linux on x86_64 and ARM



Document version: 2.29

Document name: AUD-MAN-DEP_Programmers-Guide-v2.29

Published: Friday, September 26, 2025

Feedback: if you would like to suggest improvements to this information, please feel free to email us at documentation@audinate.com.

Dante®

Copyright

Legal Notice and Disclaimer

Audinate retains ownership of all intellectual property in this document.

The information and materials presented in this document are provided as an information source only. While effort has been made to ensure the accuracy and completeness of the information, no guarantee is given nor responsibility taken by Audinate for errors or omissions in the data.

Audinate is not liable for any loss or damage that may be suffered or incurred in any way as a result of acting on information in this document. The information is provided solely on the basis that readers will be responsible for making their own assessment, and are advised to verify all relevant representation, statements and information with their own professional advisers.

Software Licensing Notice

Audinate distributes products which are covered by Audinate license agreements and third-party license agreements.

For further information and to access copies of each of these licenses, please visit our website:

www.audinate.com/software-licensing-notice

Contacts

Audinate Pty Ltd

Level 7, 64 Kippax Street
Surry Hills NSW 2010
Australia
Tel. +61 2 8090 1000
info@audinate.com
www.audinate.com

Audinate Inc

4380 S Macadam Avenue
Suite 255
Portland, OR 97239
USA
Tel: +1.503.224.2998
Fax. +1.503.360.1155

European Office

Audinate Ltd
Future Business Centre
Kings Hedges Rd
Cambridge CB4 2HY
United Kingdom
Tel. +44 (0) 1273 921695

Asia Pacific Office

Audinate Limited
Suite 1106-08, 11/F Tai Yau Building
No 181 Johnston Road
Wanchai, Hong Kong
澳迪耐特有限公司
香港灣仔莊士敦道181號
大有大廈11樓1106-8室
Tel. +(852)-3588 0030
+(852)-3588 0031
Fax. +(852)-2975 8042

Warning

To avoid the possibility of personal physical injury or damage to the equipment, always take sensible precautions when using this equipment including but not limited to the following:

- Do not attempt to modify the equipment or disassemble it.
- Disconnect the power cables of audio devices before installing this equipment. Turn off any peripheral devices connected to the audio device and unplug all related cables.
- Handle the equipment carefully.
- Drain any static electricity from your body or clothing before handling the equipment.
- Do not drop the equipment or subject it to physical shock.
- Avoid loose screws or other extraneous metal objects coming into contact with the equipment when installed and powered up.
- Do not expose the equipment to sudden temperature changes from cold to hot.
- Do not expose the equipment to rain or moisture.
- Avoid installing this equipment where foreign objects may fall onto this unit and/or this equipment may be exposed to liquid dripping or splashing.

Audinate cannot be held responsible for any loss or destruction of data or damage caused by improper use of or modifications to the device.

Important: Audinate products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, or any other application that invokes the potential risks of death, personal injury or property or environmental damage ("critical applications"). Use of products in critical applications is at the sole risk of the customer, subject to applicable laws and regulations. All specifications are subject to change without notice.

Contents

Copyright 1
About Audinate 8
About Dante 8
One Connection. Endless Possibilities. 8
Revision History 9
Introduction 14
Building a Product Using DEP 14
Assumed Knowledge 15
Familiarise Yourself with DEP 16
Configuration Files Setup 17
config.json 17
dante.json 17
Starting and Stopping DEP 18
Logging 18
Installing DEP 19
Prerequisites 19
Steps 19
Updating DEP 22
DEP Field Updatable Firmware 22
Updating the Host Platform 24
Persistent Data 24
Upgrading to DEP v1.5+ and Moving to cgroup v2 25
Hosts where init is sysvinit 25
Prerequisites 25
Steps 25
Hosts where init is systemd 26
Prerequisites 26
Steps 26
Reducing Disk Space Usage 27
Achieving CPU Isolation 28
Introduction 28
Using Kernel Parameters for CPU Isolation 28
isolcpus 28
nohz_full 28
rcu_nocbs 29
A Practical Example 29
Alternatives to Modifying the Kernel Parameters 29
systemd 29
sysvinit 30
Activating DEP 31
Activation 31

Device Activation 32
Trial Mode 32
Example Applications and Configuration 32
Applications 32
Configurations 34
DEP Container Configurations 34
DEP Device Configurations 34
System and Configuration Checks 34
PTP Timestamping Test Tool 35
Collecting Information for Audinate Support 35
DEP Performance 37
Un-encrypted Flow Testing 37
Encrypted Flow Testing 39
Networking 41
DSA Example: DEP-EVK-IMX8 41
DSA with Packet Timestamping 43
Dante Daisy Chain Mode on a Platform with a Switch 43
Dante Redundancy on a Platform with a Switch 45
Platform with a Single PHY 47
DEP Redundancy on a Platform with Multiple PHYs 48
DHCP Configuration 49
Secondary Port Link Local Addressing 49
Firewall Configuration 49
Network Interrupt Coalescing 49
mDNS Visibility 50
Other Configuration 50
Clocking 51
IEEE 1588 Precision Time Protocol (PTP) 51
Hardware Timestamping 51
Audio Muting 51
Hardware Clocks 51
Clock Feedback 52
I2S / TDM Clocks 53
Bit Clock Configuration 53
Sample Rate Changes 53
Supported Clocking Circuits 54
Si5351B with MCP47CVB02 54
Configurable Parameters 54
Configuration of Si5351B Output Clock Lines 54
Audio Interface 59
DEP Audio Interface 59
Audio Interface Timing 59
Audio Interface Metadata Block 59
Sample Rate Change 60
Example Audio Applications 60

DepLoopback 60
DepAlsaBridge 61
dinfo 62
dplay 62
drecord 63
dsoundcard 64
dtest 65
ALSA ASRC 65
Running ALSA ASRC 66
Configuring the Container for ALSA 66
Testing ALSA in the Container 68
List ALSA Devices 68
Testing Device Audio 68
Configuring ALSA ASRC in DEP 68
DEP On-Device Monitoring and Control 71
Overview 71
Host System Setup 71
Common 71
eDAPI 72
DDP 72
Dante Device-Host Interface 73
Appendices 75
Appendix 1: Kernel and Platform Configuration 75
Enable Cgroup Support 75
Cgroups Versions 75
Disable Fine-Grained Real-Time Scheduling 76
Enable Namespace Support 76
Enable SquashFS Support 76
Enable Loop Device Support 76
Low Latency Configuration 76
CPU frequency scaling options 77
Enable real time optimizations by enabling the following kernel config option 77
Network Configuration 77
Hardware RNG 77
i.MX EVK Network Driver Configuration 77
Freescale Ethernet Driver Checksum Issue 77
Freescale Ethernet Driver Coalesce Parameter Reset Issue 78
DSA Duplicate Multicast Data Issue 78
Daisy Chaining on Network Without IGMP Querier 78
Dante AV-H 78
Appendix 2: Container Configuration 78
hooks 79
Selecting the CPU cores used by DEP 79
For DEP Versions Prior to 1.5.0 79
Examples 79

Error conditions 80
For DEP Version 1.5.0 and Newer 80
process 80
mounts 80
linux : devices 81
linux : resources : devices 81
Appendix 3: Dante Configuration 81
Platform 81
Logging 82
Audio Options 82
Network Settings 82
Clock 83
Hardware Clock 83
hostcpu 83
ALSA ASRC 83
Product Information 83
Miscellaneous 83
Example Configuration File 83
JSON Field Descriptions 85
Assigning Default Channel Names 91
Creating Channel Groups 91
Specifying Per-Channel Encodings 92
Validating DEP JSON Files Using a JSON Schema 93
Validating in an IDE 94
Validating in the Command Line 95
Appendix 4: dhcpcd Patch for 172.31.x.x Link Local Addressing 96
Appendix 5: ALSA ASRC Integration 96
ALSA Diagnostic Tool 96
atest Usage 96
Interpreting atest Output 98
ALSA Parameter Space 98
Status Line 98
Tuning ALSA ASRC 99
System Tuning 99
ALSA Device Tuning 100
Index **101**

About Audinate

Audinate® is the leading provider of professional AV networking technologies globally. Audinate's Dante platform distributes digital audio and video signals over computer networks, and is designed to bring the benefits of IT networking to the professional AV industry. AV-over-IP (AVoIP) using Dante-enabled products ensures interoperability between AV devices and allows end users to enjoy high quality, flexible solutions – typically with a lower total cost of ownership.

About Dante

One Connection. Endless Possibilities.

Dante replaces all audio and video connections with a computer network, effortlessly sending video or hundreds of channels of audio over slender Ethernet cables with perfect digital fidelity.

Adopted by hundreds of manufacturers in thousands of products, Dante is the de facto standard for modern AV connectivity.

For more information, please visit the Audinate website at getdante.com.

Revision History

Version	Date	Change
2.29	25th September 2025	■ Add Dante Device-Host Interface section ■ Update "Clock Feedback" section to describe the new extclkin-gpio driver and phase alignment. ■ Add "extLrclockInputDev" to "hardwareClock" section of dante.json in Appendix 3: Dante Configuration ■ Add "cpuAffinity" to "alsaAsrc" section of dante.json in Appendix 3: Dante Configuration
2.28	9th September 2025	■ Updated sections describing how to install and update DEP. Includes section on updating from croups V1 to cgroups V2. ■ Add section about CPU isolation ■ Add ALSA ASRC section to Audio Interface chapter, and associated section in Json Field Descriptions. ■ Add Appendix 5: ALSA ASRC Integration ■ Added new "ddhi" dante.json parameters ■ Update "Logging" description in the Platform section and add "logLevel" dante.json field ■ Remove "DEP and Dante Updater" section ■ Add CPU usage numbers for encrypted flows ■ Add PTP Timestamping Test Tool section ■ Remove Configuration Fixer section ■ Remove references to ethtool in description of enableHwTimestamping ■ Fix incorrect default encoding setting for per-channel encoding JSON file ■ "percentCpuShare" deprecated
2.27	22nd May 2025	Update the trial timer duration in Activation > Trial Mode

Version	Date	Change
2.26	24th March 2025	■ Add "defaultChannelNamesFile" and new "video" section to JSON Field Descriptions ■ Add "followerOnly" to "clock" section of dante.json in Appendix 3: Dante Configuration ■ Increase "maxNumLogs" default to 6 ■ In DHCP Configuration, emphasise that DHCP and link local must be supported. ■ Indicate that hardware timestamping has more than one mode in System and Configuration Checks ■ Expand description of "enableHwTimestamping" to describe the new "v1" value and when it should be used ■ Added new mDNS Visibility section
2.25	12th February 2025	Fix broken link
2.24	16th May 2024	■ Add "perChannelEncodingsFile" and "silenceHeadDelayMs" to "audio" section of dante.json in Appendix 3: Dante Configuration ■ Add new "Specifying Per-Channel Encodings" section to Appendix 3: Dante Configuration ■ Added entry re. DDM to Backwards Compatibility
2.23	29th November 2023	■ Update to reflect new examples package structure ■ Added additional kernel options required for Dante AV-H v1.0.7 onwards ■ Added 'Validating DEP JSON Files Using a JSON Schema' in Appendix 3: Dante Configuration
2.22	31st August 2023	■ Add "misc" section and "enableIdentify" parameter for dante.json (Appendix 3: Dante Configuration) ■ Add "enableSelfSubscription" parameter to "audio" section of dante.json ■ Add "webSocketPort" to "network" section of dante.json ■ Add "loadCapacitance" parameter to "hardwareClock" section of dante.json ■ Mark "defaultEncoding" as deprecated and update "supportedEncodings" description
2.21	18th April 2023	■ Add Configuration of Si5351B Output Clock Lines section ■ Add aes67Supported parameter for dante.json (Appendix 3: Dante Configuration) ■ Add Hardware Timestamping section

Version	Date	Change
2.20	13th December 2022	■ Describe new trial mode for DEP artifact (Activation) ■ Add description of Cgroups v1 vs v2 (Appendix 1: Kernel and Platform Configuration) ■ Add “maxLogSize” dante.json parameter (Appendix 3: Dante Configuration) ■ DEP Performance section: ◦ Updated performance numbers and added explanatory notes ◦ Explain source of unusual CPU usage at low/zero channel count ■ Add description of Channel Groups feature (Appendix 3: Dante Configuration)
2.19	4th October 2022	Clarify minimum required kernel version
2.18	25th July 2022	■ Appendix 3: Dante Configuration: ◦ `maxRxFlows`, `maxTxFlows`, `defaultChannelNamesFile` and `hostcpu` (`enableDdp`) added to JSON field descriptions ◦ 'Assigning default channel names': New section ■ Quick Start: Note added re. using root access account. sudo syntax added to relevant steps. ■ 'Deploying DEP': Backwards Compatibility section added ■ DEP Performance: Paragraph added re. changing the maximum number of flows from the default values ■ DEP On-Device Monitoring and Control: New section ■ Appendix 1: Kernel and Platform Configuration: Disable Fine-Grained Real-Time Scheduling added
2.17	24th January 2022	■ Add “maxNumLogs” for log file rotation in example configuration file ■ Add eDAPI section ■ Update to Low Latency Configuration in Appendix 1: Kernel and Platform Configuration ■ New section: Selecting the CPU cores used by DEP in Appendix 2: Container Configuration
2.16	24th May 2021	Fix supported channel count in DEP Audio Interface

Version	Date	Change
2.15	9th March 2021	■ Supported channel count increased to 128x128 ■ 'Building a Product Using DEP': New content ■ Additional optional steps added to Quick Start ■ DEP activation command updated in Quick Start (step 11) ■ New example applications added: ◦ dinfo ◦ dplay ◦ drecord ◦ dsoundcard ◦ dtest ■ System and Configuration Checks: New content ■ Configuration Fixer: New content ■ Additional information regarding Linux network interface configuration added to Networking ■ 'Mute Signal' updated to 'Audio Muting ■ New content regarding secondary network interfaces added to 'linux : devices' in Appendix 2: Container Configuration ■ New line regarding >3 cores added to 'numDepCores' in 'JSON Field Descriptions' in Appendix 3: Dante Configuration ■ Inclusive language updates
2.14	11th November 2020	■ 'PDK' changed to 'EVK' throughout ■ Directory trees converted to text ■ DSA with Packet Timestamping: Updated content re. when DSA is being used with a switch ■ Appendix 3: Dante Configuration: ◦ `"preferredLinkSpeed"` added to example config file and 'JSON Field Descriptions' ◦ `"percentCpuShare"` added to example config file and 'JSON Field Descriptions' ◦ `"networkLatencyDefaultMs"` updated in 'JSON Field Descriptions' ◦ `"numDepCores"`- updated in 'JSON Field Descriptions' ◦ `"dsaTaggedPackets"` - added to 'JSON Field Descriptions' ■ Appendix 4: dhcpcd Patch for 172.31.x.x Link Local Addressing updated

Version	Date	Change
2.13	9th October 2020	■ Note re. use of console vs SSH added to Networking ■ Information re. daisy chain & redundant modes added to `interfaceMode` in the 'JSON Field Descriptions' section of Appendix 3: Dante Configuration
2.12	1st September 2020	■ Added 'Secondary Port Link Local Addressing' to DHCP Configuration ■ Added Appendix 4: dhcpcd Patch for 172.31.x.x Link Local Addressing
2.11	24th August 2020	Modifications to Hardware Clocks section
2.10	13th August 2020	Hardware RNG added to Appendix 1
2.9	22nd July 2020	DEP 1.0 EVK for i.MX8 first edition
2.8	17th April 2020	Renamed to Programmer's Guide
2.7	7th January 2020	Added sections on DEP folder layout and DEP update process
2.6	4th December 2019	Updated required kernel version

Introduction

Dante Embedded Platform (DEP) is a software implementation of Dante, the world's most widely used AV-over-IP solution. It addresses emerging needs for an industry transitioning to a more software-focused model of system and product design, providing the tools manufacturers need to make AV as software a reality.

Dante Embedded Platform (DEP) is designed for products based on modern 64-bit Intel / AMD processors (x86_64), 32-bit ARM (armv7-a), or 64-bit ARM (armv8-a, aarch64) architectures running the Linux operating system. It is packaged as an Open Containers Initiative (OCI) container.

DEP presents a shared memory audio API, enabling your audio application to send and receive audio to and from a Dante network.

The DEP system will appear on the Dante network as a standard Dante device with up to 128x128 networked audio channels.

DEP supports the following Dante network configuration:

Feature	Supported
Network audio channels	128x128 (input x output; configurable)
Network sample rate/s	44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz
Sample rate pull-ups	Not available
Network encoding	■ PCM 16-bit ■ PCM 24-bit ■ PCM 32-bit
Network latency	Configurable
Can be clock leader	Yes
Dante redundancy	Yes
Daisy chaining	Yes

Building a Product Using DEP

To build a product with Dante audio networking provided by DEP requires integration in several different areas. This manual provides detailed information on how to integrate DEP in each area, along with example code and other documents and files included in the DEP package. At a high level, these areas of integration include:

- DEP container integration, configuration, start-up and update handling
- Activation of the desired DEP configuration for deployment (licensing)
- Network configuration including optional switch configuration

- Audio input and output interface with other local software
- Clock bridging between Dante clock and local audio clock, either via interface with the reference clock synchronization hardware or via asynchronous sample-rate conversion (ASRC)
- Kernel and platform configuration
- Testing and validation of performance

Assumed Knowledge

It is expected the reader of this programmer's guide has the following prior knowledge:

- An overall familiarity with the Linux operating system
- A familiarity with installing and managing a Linux system service
- Knowledge of the JSON format to work with DEP configuration files

Familiarise Yourself with DEP

Once extracted, the DEP archive (a gzip-compressed tar file) has the following structure:

```
dep
└── dante_package
    ├── bundle
    │   ├── config.json -> ../dante_data/capability/config.json
    │   └── rootfs
    ├── crun
    ├── dante_data
    │   ├── activation
    │   ├── capability
    │   ├── config
    │   └── images
    │       ├── 0
    │       │   └── rootfs_squash
    │       ├── 1
    │       └── active
    ├── dep_check.sh -> development/dep_check.sh
    ├── depconfig
    ├── dep.sh
    ├── dep_support_collection.sh
    ├── development
    │   ├── dep_check.sh
    │   ├── dep.service
    │   └── tools
    │       ├── jq
    │       └── ptp_timestamping_test
    └── version

 10 directories, 13 files
```

Figure 1 - DEP package archive layout

Under `dante_package` are the following directories:

- `bundle` - the container bundle directory. This also contains a link to the `config.json` file used to configure the DEP device container.
- `bundle/rootfs` - the container rootfs folder is deliberately empty. When DEP is started, a DEP image will be mounted into this folder as a read-only root filesystem.
- `dante_data` - files which are read and written during normal operation of DEP, such as activation, runtime configuration and DEP images. The `dante_data` directory must reside on a read-write flash partition and is bind mounted into the container. All data written by DEP is confined to the `dante_data` folder. The activation, capability and config directories are deliberately provided empty to permit simple upgrades of the `dante_package`. Extracting a new `dante_package` over the top of an existing installation should not overwrite any existing activation, capability or configuration files.
- `development` - scripts, files and tools useful during integration of DEP. If disk storage is a concern, can be safely removed from production firmware.

Configuration Files Setup

For the correct functioning of DEP there are two configuration files that need to be set up prior to starting DEP. Before starting DEP for the first time the configuration files must be created in the `dante_package/dante_data/capability` directory.

config.json

`config.json` is the standard OCI configuration file. It contains metadata that describes how to setup and run an OCI container. The syntax and format are described as part of the OCI run time specification which can be found at https://github.com/opencontainers/runtime-spec/blob/master/config.md. There are example `config.json` files in the DEP examples archive within the config directory. Use the architecture specific version as a starting point.

`config.json` contains references to absolute paths that must be modified to reflect the installation path for DEP. The examples assume that the DEP container has been installed into `/opt`. The following sed command may be used to replace all the `/opt` paths with the DEP install path under `/usr/local`.

```
sed -i -e "s%/opt/dep/dante_package%/usr/local/dep/dante_package%g" config.json
```

Please refer to Appendix 3: Dante Configuration for details on the elements of `config.json` that may need to be modified to suit your platform.

dante.json

`dante.json` is the DEP configuration file. It contains settings that control the Dante device behaviour and is described in more detail in Appendix 3: Dante Configuration.

There are a number of example `dante.json` files in the DEP examples archive located at `config/dante.`

Starting and Stopping DEP

The `dante_package` directory contains the script `dep.sh` which can be used to start and stop DEP as follows:

```
# sudo ./dep.sh start
# sudo ./dep.sh stop
```

For systems where init is systemd, `development/dep.service` can be used for automatic handling of DEP. Instructions on how to install and enable new systemd service units are available online.

Logging

By default, all logs for the DEP sub-processes are found in:

```
/var/log/
```

- prefixed with dante_

The logging parameters can be configured in the DEP configuration file. See Appendix 3: Dante Configuration.

Installing DEP

This section provides installation instructions for DEP, guiding the user through the specific characteristics of the target platform.

DEP is packaged as an OCI-compatible Linux container. A Linux container includes its own runtime environment and thus is independent of the Linux distribution on which it runs. It is important that the OEM retain this package structure and not attempt to run Dante binaries natively as Audinate could change its runtime environment in future updates rendering it incompatible with the current native runtime.

DEP requires cgroups and namespace support in the Linux kernel. The minimum supported kernel version is 4.9. However, starting from DEP 1.5.0, both cgroupv1 and cgroupv2 are supported, with cgroupv1 planned for deprecation in the future. As such, the recommended minimum kernel version for DEP is 5.10.

Future versions of the Dante Embedded Platform will take advantage of features available only in more recent kernels to optimize resource usage and maximize the number of audio channels supported. Therefore, we recommend using Linux kernel version 5.10 LTS or newer to ensure compatibility with future, optimized versions of DEP. Please refer to Appendix 1 for details on required kernel and platform configuration.

Note: an up-to-date x86_64 Linux distribution such as Ubuntu 24.04 LTS will have the kernel configuration options required by DEP enabled by default. We recommend you use a minimal workstation distribution to simplify platform optimisation.

Prerequisites

The following resources are required in this chapter:

- DEP container release package for the target platform architecture
  - Example: for an x86 host device, use `dep_cp_activation-1.5.0_x86_64_Linux.tgz`
- DEP examples release package
  - Example: for an x86 host device, use `dep_examples-1.2.0.2_x86_64_Linux.tgz`
- Root privileges on the target Linux system (e.g., the ability to execute commands as the root user)
- At least 25 MB of available disk space on the partition where DEP will be installed
- Windows PC with Dante Controller (DC) and Dante Activator tool (dante_activator.exe) installed.
  - Note: both DC and Dante Activator are also available for macOS
- Another Dante device connected to the Dante network to send/receive test flows to/from DEP. These instructions use Dante Virtual Soundcard (DVS) as an example.

Steps

1. Transfer and extract the DEP container and examples release packages onto the DEP host device. This guide assumes the files have been transferred to the device's `/tmp` directory and that the environment variable `DEP_INSTALL_DIR` has been set with the example install directory path `/usr/local`.

   Note: you will need to run DEP with root permissions, by (for example) using sudo or running as the root user.

```
user@host:~ # export DEP_INSTALL_DIR=/usr/local
user@host:~ # cd $DEP_INSTALL_DIR
user@host:/usr/local # sudo tar xvf /tmp/dep_cp-1.5.0_aarch64_Linux.tgz
user@host:/usr/local # sudo tar xvf /tmp/dep_examples-1.2.0.0_aarch64_Linux.tgz
```

2. From the DEP examples package, select the relevant container configuration to use as a base depending on which cgroup version your Linux host provides. Copy the example config file into the required DEP install location.

```
user@host:/usr/local # sudo cp dep_examples/config/config.cgroup_v2.json dep/dante_
package/dante_data/capability/config.json
```

3. Patch `config.json` to update paths to point to the install directory.

```
user@host:/usr/local # sudo sed -i -e "s%/opt/dep/dante_package%$DEP_INSTALL_DIR/dep/dante_
package%g" dep/dante_package/dante_data/capability/config.json
```

4. From the DEP examples package, select the relevant Dante configuration to use as a base. This guide uses the `dante.max.json`. Copy the example config file into the required DEP install location.

```
user@host:/usr/local # sudo cp dep_examples/config/dante.max.json dep/dante_package/dante_
data/capability/dante.json
```

5. Edit the dante.json file to customise DEP to run on your Linux host. In particular:
    a. The `"network.interfaces"` value needs to be modified to list the Ethernet interface on your Linux host that will be used to connect to the Dante network.
    b. Set `"platform.cgroupVersion"` to the appropriate value (1 or 2).
    c. If you don't own a valid license, DEP can be started in trial mode (providing ~15 minutes of runtime with full functionality) by adding the `"trialMode": true` field. The first lines of `dante.json` would then look like:

```
{
    "trialMode": true,
    "platform":
      {
          "logDirectory" : "/var/log",
          "cgroupVersion": 2
      },
```

    d. If necessary, change the default logging location for DEP by setting the value of `"platform.logDirectory"`.
    e. For other settings in `dante.json`, refer to Appendix 3: Dante Configuration.
6. Run the `dep_check.sh` script.

```
user@host:/usr/local # cd dep/dante_package/development
user@host:/usr/local/dep/dante_package/development # ./dep.sh /usr/local/dep
```

The script looks for common missing, incompatible or incorrect settings on the system and in the DEP configuration files.

While most warning messages from `dep_check.sh` should not prevent DEP from starting correctly, follow the advice provided by any error message before proceeding. More details about `dep_check.sh` can be found in the System and Configuration Checks section.

7. Start DEP:

```
user@host:/usr/local/dep/dante_package # sudo ./dep.sh start
```

8. The DEP device should be up and running and listed in Dante Controller as a Dante device: if you own a valid license, DEP can be activated to run in full production mode. See Activation for activation requirements and procedures. Below is an example command that can be run on a Windows PC connected to the Dante network to activate a DEP device:

```
dante_activator.exe activate --i="Ethernet" --d="DEP-12345" --e="1234-5678-9abc-def0-1234"
```

9. Start the DepLoopack application found in the example package as background process:

```
user@host:/usr/local/dep/dante_package # cd $DEP_INSTALL_DIR/dep_examples
user@host:/usr/local/dep_examples # sudo apps/DepLoopback &
```

10. In Dante Controller, subscribe DEP RX channels to DVS TX channels and on the return side, subscribe DVS RX channels to DEP TX channels. Send audio out DVS and it will be looped back from DEP and the DepLoopback application to DVS.

Updating DEP

Over the course of a product lifecycle it is expected that the DEP based device will require updates to both the DEP container and the host platform.

Before updating DEP, please review the Backwards Compatibility table below.

Table 1 - Backwards Compatibility

Changes	Previous Releases Status	Current Release Status

Change of mount point for DEP Unix socket shared with container and host OS for communication	In DEP v1.0.4.x and earlier, the default Unix socket location was in:
	`/var/run`
		From DEP v1.0.5.x and newer, the default location is:
		`/var/run/dante`
		This means that while updating, the following mount point in the config.json configuration file:
		`{ "destination": "/var/run/dante", "type": "tmpfs", "source": "tmpfs", "options": [ "nosuid", "strictatime", "mode=755", "size=65536k" ] },`
		must be updated to point to the `/var/run/dante` folder.
		See Appendix 2: Container Configuration for more details.

Dante Domain Manager (DDM) versions older than 1.4 will not be compatible with DEP 1.3.3 or newer	DEP versions 1.3.1.x and earlier works with all versions of DDM.	DEP version 1.3.3.x and newer will only work with DDM versions 1.4 and newer.

DEP Field Updatable Firmware

A DEP device can update its own container via the network. DEP provides an A/B partitioning scheme for firmware images. This means that there are always two DEP images on the system. One is designated as the active image and one as the inactive image. Each time DEP starts up it will run the currently selected active image. When a firmware update is initiated, DEP will attempt to download the specified new firmware and overwrite the inactive image with the new firmware. If the update process is successful, then the new firmware will become the currently selected active image. The new firmware will be run the next time DEP is restarted. If for any reason the update process fails, then the active image is unaffected.

The example program, "conmon_audinate_controller", can be used to initiate a firmware update on a given DEP device. The format of the command to download an image from an image repository server and use it to update DEP is as follows:

```
conmon_audinate_controller <device_name> upgrade3 http=<server_ipv4>[:server_port] path=<file_path>
[:tag]
```

Example:

```
conmon_audinate_controller DEP-123456 upgrade3 http=10.1.1.1:5000 path= v2/audinate/dep_cp_
activation-aarch64-1.0.0.4:latest
```

Updating the DEP container via the Dante Updater (DU) will be supported in a future DU release.

Updating the Host Platform

It is OEM's responsibility to update the host platform. This includes all software outside of the DEP container. An OEM could also include DEP updates in their update process. When updating the DEP device, it is important to consider the preservation of persistent data.

Persistent Data

DEP stores persistent data in the `dante_data` folder. Persistent data includes all device activation information and any Dante user configuration, for example, a device name set by the user.

A DEP container-only update will not delete the `dante_data` folder. If the OEM performs a platform update, the `dante_data` directory must be retained or restored after the update.

Extracting a new `dante_package` release over the top of an existing installation should not overwrite existing activation, configuration or capability files. It is OEM's responsibility to verify such an upgrade procedure and compatibility with new versions prior to end-user field updates being made available.

Important: If the `dante_data` folder is not restored after an update, the device will go back to an un-activated (unlicensed) state and all user settings will be lost. This would render the Dante functionality of the device inoperable. It is advised to have a process in place for retaining and repopulating the license file on the platform should it be removed by a firmware update in the field.

Upgrading to DEP v1.5+ and Moving to cgroup v2

DEP v1.5 introduced support to cgroup v2. While support to cgroup v1 is still available, new and existing users are encouraged to run DEP with cgroup v2.

Hosts where init is sysvinit

Prerequisites

- DEP 1.5.0 archive and corresponding examples archive
- Linux kernel 5.10+ is strongly recommended
- `CONFIG_CGROUP2=y` and `CONFIG_TMPFS=y` in the kernel config

Steps

1. Stop DEP.
2. Set `platform.cgroupVersion` to 2 in `dante.json`.
3. Extract the new DEP archive onto existing installation path.
4. On sysvinit systems, cgroups mounts must be performed manually. Depending on the specific system setup, existing cgroups v1 mountpoints might be handled in different ways on your device:
    a. A script in /etc/init.d manually mounting cgroups
    b. A few entries in /etc/fstab that specify the mountpoints

    Identify the appropriate script/file that is responsible for mounting cgroups v1 on your system and update it to mount the cgroups v2 hierarchy instead. It is important that no cgroups v1 mount points are left when configuring cgroups v2 to avoid ending up with cgroups configured in “hybrid mode”, which is not supported by DEP’s container runtime.
5. `config.json` needs to be updated to match the cgroups v2 schema:
    a. Remove the hooks section responsible for running `depconfig` - please take note of any set of specific CPU cores that were specifically assigned to DEP in dante.json, if any.
    b. Change the type and source of the `/sys/fs/cgroup` mount destination under the `"mounts"` section to `cgroup2`.
    c. Remove the `linux.resources` object entirely.
    d. Add the `linux.cgroupsPath` field and set it to `"dante.slice"`
    e. Add a new `"type"` field in the `linux.namespaces` object and set its value to `"cgroup"`
    f. You can refer to the `config.cgroup_v2.json` example provided in the examples archive for more guidance.

6. If you did customise your `config.json` to specify which CPU cores were specifically assigned to DEP, use the `"audio.numDepCores"` in `dante.json` to achieve the same outcome.

   For instance, if you had `numDepCores: 3` set in `dante.json` and CPU cores 3,4,5 set in `config.json` (passed as arguments to the depconfig tool) you should set `numDepCores: [3, 4, 5]` in `dante.json`.
7. Run `dep_check.sh` to make sure all platform prerequisites are met.
8. If no errors are returned from the script, start DEP and check normal functionality is restored.

Please refer to the “Cleaning up old files” section next.

Hosts where init is systemd

Prerequisites

- DEP 1.5.0 archive and corresponding examples archive
- Linux 5.10+ is strongly recommended
- `CONFIG_CGROUP2=y` and `CONFIG_TMPFS=y` in the kernel config
- Full cgroup v2 support in systemd arrived in version 241

Steps

1. Stop DEP.
2. Set `platform.cgroupVersion` to `2` in `dante.json`.
3. Extract the new DEP archive onto existing installation path.
4. If the kernel command line was previously modified to force systemd to use cgroups v1 - remove those additional options and reboot before proceeding.
5. `config.json` needs to be updated to match the cgroups v2 schema:
   a. Remove the hooks section responsible for running depconfig - please take note of any set of specific CPU cores that were specifically assigned to DEP in `dante.json`, if any.
   b. Change the type and source of the `/sys/fs/cgroup` mount destination under the `"mounts"` section to `cgroup2`.
   c. Remove the linux.resources object entirely.
   d. Add the `linux.cgroupsPath` field and set it to `"dante.slice"`
   e. Add a new `"type"` field in the `linux.namespaces` object and set its value to `"cgroup"`

      You can refer to the cgroupv2 `config.cgroup_v2.json` example provided in the examples archive for more guidance.
6. If you did customise your `config.json` to specify which CPU cores were specifically assigned to DEP, use the `"audio.numDepCores"` in `dante.json` to achieve the same outcome.

   For instance, if you had `numDepCores: 3` and CPU cores 3,4,5 set in `config.json`, you should set `numDepCores: [3, 4, 5]` in `dante.json`.
7. Run `dep_check.sh` to make sure all platform prerequisites are still met.
8. If no errors are returned from the script, start DEP and check its functionality,

Please refer to the “Cleaning up old files” section next.

Reducing Disk Space Usage

As of DEP v1.5.0, the following list of files are not needed for production systems running DEP and can be removed to save storage space. The files are either only needed during DEP system development / integration or are files from earlier DEP releases which have been deprecated.

File (all paths relative to dep/dante_package)	Reason
`development` (entire directory and contents)	Development files not needed for production
`runc`	Old container runtime replaced by crun
`fixer`	Deprecated utility
`dep_util.sh`	Deprecated source file
`dep.service`	Old development file (newer version in development directory)
`select_image.sh`	Deprecated source file
`dante_data/images/1/rootfs_squash`	Use of second rootfs has been deprecated

Achieving CPU Isolation

Introduction

Real-time digital audio processing requires predictable execution with minimal latency. When audio data is transmitted and received over a network, any delay or jitter in the processing pipeline can result in audible artifacts, degraded performance, or even complete audio dropouts.

On a Linux system, tasks normally share CPU resources with the kernel scheduler distributing workloads across available cores. This default behaviour ensures fair distribution of compute time, but it is not suitable for applications that must deliver deterministic performance. Interrupts, kernel housekeeping tasks, and unrelated userspace processes can all preempt a DEP processing thread at inopportune times, causing scheduling jitter.

For this reason, isolating a subset of CPUs for the exclusive use of DEP is essential. When tasks running in userspace are bound to dedicated cores that are shielded from unrelated kernel and userspace activity, DEP can achieve the consistent timing guarantees required for glitchless networked audio.

DEP itself uses the `sched_setaffinity()` system call to bind its processing threads to user-configured CPUs. However, this step alone does not ensure isolation: it simply requests that the kernel schedule the relevant threads on specific cores. When building a product with DEP It is the user responsibility to ensure that these CPUs are not interrupted by background daemons, housekeeping activities, or other user applications.

Note: The guidelines presented in this chapter are intended to introduce the core concepts of CPU isolation and platform tuning. Due to variations in hardware architectures, kernel versions, and deployment scenarios, the exact configuration that yields optimal results may differ from system to system. It is therefore the responsibility of the user to evaluate, experiment, and refine these settings in order to achieve the best performance on the specific platform in use.

The following sections describe how to achieve proper CPU isolation, with a few examples.

Using Kernel Parameters for CPU Isolation

Linux provides mechanisms at the kernel level to reserve specific CPUs for dedicated workloads. These key parameters are relevant:

isolcpus

This kernel command line parameter marks designated CPUs as isolated from the general scheduler. Regular processes will not be scheduled on these cores unless explicitly bound to them via affinity. This prevents interference from non-critical workloads.

nohz_full

This parameter complements `isolcpus` by reducing the frequency of kernel timer interrupts on the isolated CPUs. Normally, the Linux kernel generates periodic timer ticks for task accounting and scheduling purposes. On CPUs

listed under `nohz_full`, these ticks are suppressed when a single task is running, allowing uninterrupted execution and further reducing jitter.

rcu_nocbs

Even when CPUs are isolated and running "tickles" (`nohz_full`), the kernel may still schedule Read-Copy-Update (RCU) callback processing on them. RCU is an internal synchronization mechanism used heavily inside the kernel, and its callbacks can introduce unpredictable latency. The `rcu_nocbs` parameter instructs the kernel to offload these callbacks from the isolated CPUs onto the housekeeping CPUs, ensuring that real-time workloads remain undisturbed.

A Practical Example

Assuming an 8-core system and DEP configured via `numDepCores` option to run exclusively on CPU cores 1,2,3 we then have these two groups of CPU cores:

- Isolated CPUs: 1,2,3 reserved for real-time audio tasks
- Housekeeping CPUs: 0,4,5,6,7 to handle interrupts, background services, etc...

The resulting kernel command line would look like this:

```
BOOT_IMAGE=/boot/vmlinuz-6.8.0-71-generic root=UUID=94abcf72-8c9a-4eed-a070-60c2e364f4e5 ro
isolcpus=1,2,3 nohz_full=1,2,3 rcu_nocbs=1,2,3
```

Modifying the kernel command line requires different steps depending on the Linux host.

On Ubuntu and other distributions that use GRUB, this typically involves editing the GRUB configuration file and regenerating its runtime configuration. On Yocto, Buildroot, or other embedded Linux systems, users should consult the relevant documentation for the proper procedure. On Raspberry Pi OS, the kernel command line is usually stored as a text file in the boot partition and can be edited directly with a text editor. In all cases, an online search will provide several methods specific to the system in use.

Alternatives to Modifying the Kernel Parameters

If modifying the kernel command line is not possible, different - but less effective - approaches are available depending on the characteristics of the Linux host.

systemd

In systemd, slices are special unit types that group together processes for the purpose of resource management. They form a hierarchical structure (e.g., system.slice, user.slice, dante.slice) and are used to apply limits on CPU, memory, and other resources to all processes within them using Linux cgroups.

One way to achieve CPU affinity with systemd is by setting a different `AllowedCPUs` value for the different systemd slices. The provided `dep.service` can be used as example: assuming DEP is running on a 4-cores CPU and that core 0 is the one that should only be used by non-Dante processes, the following `dep.service` will configure the system to match such prerequisites:

```
[Unit]
Description=Dante
Documentation=
After=network.target

[Service]
Type=simple
ExecStartPre=/usr/bin/systemctl set-property init.scope AllowedCPUs=0
ExecStartPre=/usr/bin/systemctl set-property system.slice AllowedCPUs=0
ExecStartPre=/usr/bin/systemctl set-property user.slice AllowedCPUs=0
ExecStart=/opt/dep/dante_package/dep.sh start
ExecStop=/opt/dep/dante_package/dep.sh stop
WorkingDirectory=/opt/dep/dante_package
PIDFile=/run/dante.pid
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

Once the host is rebooted the changes in CPU affinity should take place. Monitoring tools such as htop can be used to verify correct application of the policies in the service file.

sysvinit

On systems running sysvinit, services are typically started through shell scripts located in /etc/init.d/. Unlike systemd, there is no built-in support for CPU affinity. Instead, users must explicitly set the affinity inside the init script that launches the service.

Tools such as taskset or schedtool are commonly used to launch a process with a given CPU affinity, documentation and examples of their usage are available online.

Finally, users of sysvinit-based systems can also use cgroups to achieve CPU affinity. By placing the service's process in a dedicated cgroup and restricting that cgroup's access to a subset of CPUs (via the cpuset controller), one can enforce similar constraints as with AllowedCPUs in systemd.

The general steps involve:

- Creating a cgroup under the cpuset hierarchy.
- Configuring its cpuset.cpus file to specify the allowed CPU cores.
- Starting the service under that cgroup (either manually or by adapting the init script).

Users are encouraged to consult the documentation of their system or search online for step-by-step instructions.

Activating DEP

Activation

A DEP device is required to be activated before operation.

The activation process uses number of device unique identifiers to issue a device-specific license from Audinate's licensing server. Please note that any changes to the platform on which DEP is running after activation can affect validity of the license previously issued. If the device platform changes are part of customer's product lifecycle, please work with the Audinate support team to understand how these might impact the device licensing status.

When in the un-activated state, a DEP device will appear in Dante Controller as an unlicensed device with red text. The default name of an un-activated DEP device is `DEP-abcdef`, where `abcdef` are the last 3 bytes of the device's MAC address (primary network interface). The DEP prefix may be overwritten by the corresponding `dante.json` field. For the DEP EVK, the prefix is set to is `DEP-EVK-[abcdef]`.

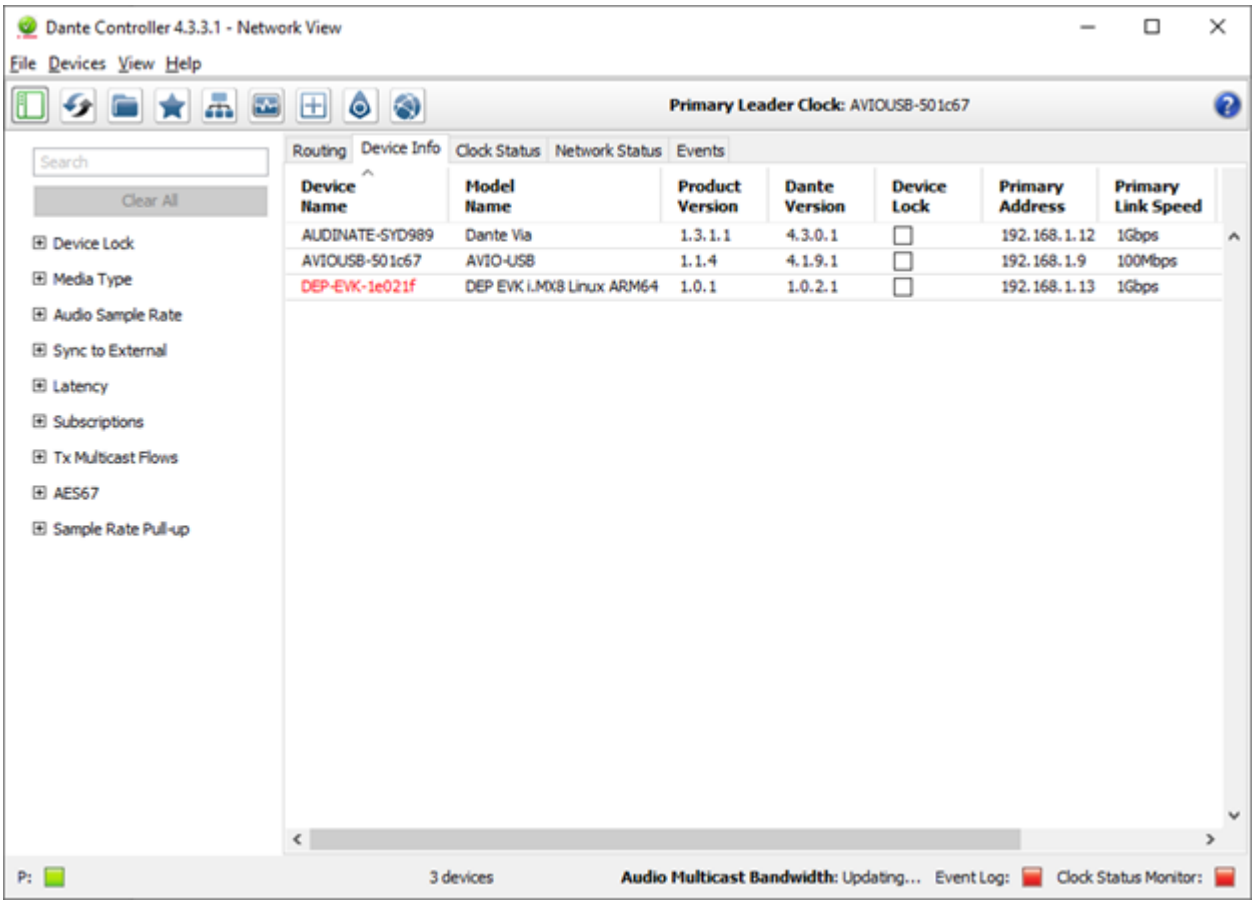


Figure 2 - DEP in Dante Controller, Device Info tab - un-activated state

In the Device View of Dante Controller, the device will appear as shown below.

Figure 3 - DEP in Dante Controller, Device View - un-activated state

Device Activation

DEP devices are activated using the DEP Activator program. Please refer to the DEP Activator User Guide for operating instructions.

If you have purchased the DEP EVK for iMX8, the supplied boards will have been pre-activated for you.

After activation, the activation files will be located in the DEP persistent data directory, i.e.

```
dante_package/dante_data/activation
```

It is expected that all files in the DEP persistent directory will be retained after a system upgrade or reset.

Trial Mode

DEP versions 1.1.1 and later can be configured to start in trial mode. When started in trial mode, DEP will provide all features without requiring activation - however, DEP will exit approximately 15 minutes after starting.

Please note that in trial mode DEP does not support Dante Media Encryption for devices enrolled to Dante Director managed domain with encryption policy enabled i.e. device will only support unencrypted media flows and enabling encryption in Dante Director for device channels when in trial mode will not be possible.

To configure DEP to run in trial mode, set `"trialMode" : true` at the root level in the Dante Configuration (`dante.json`) file (note: JSON field order does not matter). See Appendix 3 for more information.

Example Applications and Configuration

Applications

Several example applications with source code are provided to assist in developing your DEP product.

```
dep_examples
├── apps
│   ├── DepAlsaBridge
│   ├── DepLoopback
│   ├── DepTiming
│   ├── dinfo
│   ├── dplay
│   ├── drecord
│   ├── dsoundcard.so
│   └── dtest
├── CMakeLists.txt
├── config
│   ├── config.cgroup_v1.json
│   ├── config.cgroup_v2.json
│   ├── dante.alsa_asrc.json
│   ├── dante.base_stereo_redundant.json
│   ├── dante.base_stereo.json
│   ├── dante.cpu_share.json
│   ├── dante.hardware_clocking.json
│   ├── dante.json_schema.json
│   ├── dante.max.json
│   ├── dante.switched_network_redundant.json
│   ├── default_channel_names.json_schema.json
│   ├── README.md
│   └── stereo_channel_names.json
├── README.md
└── source
    ├── 3p_include
    ├── AlsaDevice
    ├── alsalib.cmake
    ├── CMakeLists.txt
    ├── dep_alsa_bridge
    ├── dep_audio_buffers
    ├── dep_info
    ├── dep_loopback
    ├── dep_play
    ├── dep_record
    ├── dep_soundcard
    ├── dep_test
    ├── dep_timing_trace
    ├── toolchain.cmake
    └── version

16 directories, 26 files
```

Figure 4 - dep_examples directory structure

Pre-compiled binaries have also been supplied ready to run on your chose platform. The example application binaries are in the `dep_examples/apps` directory.

- `DepLoopback`. A loopback software application that copies audio data from DEP receive (RX) channels back to the DEP transmit (TX) channels.
- `DepAlsaBridge`. An application that sends audio data from DEP receive to the output channels of an ALSA sound card, while simultaneously passing audio data from ALSA sound card input back to the DEP transmit channels. This application assumes that the ALSA clock is synchronized to PTP time.
- `dinfo`. A software application that prints information about the DEP shared audio buffers configuration.
- `dplay`. A playback software application that copies audio data from a flac, mp3 or wav file to the DEP transmit (TX) channels.

- drecord. A recording software application that copies audio data from DEP receive (RX) channels to a wav file.
- dsoundcard. This shared library 'dsoundcard.so ' is an ALSA plugin that allows ALSA compatible apps to play into and capture from DEP.
- dtest. A software application that plays test tones or pink noise to specified DEP transmit (TX) channels.

Refer to the README.md file in the dep_examples directory for compatibility requirements and installation instructions.

Source for the applications is found in the dep_examples/source folder. Please see the README.md file for information on how to build the examples.

Configurations

Example configurations have been provided for different DEP devices.

DEP Container Configurations

- config.cgroup_v1.json: default container configuration for Linux hosts using cgroup v1
- config.cgroup_v2.json: default container configuration for Linux hosts using cgroup v2

DEP Device Configurations

- dante.base_stereo.json: 2x2 channel DEP device with a single network configuration.
- dante.base_stereo_redundant.json: 2x2 channel DEP device with a redundant network configuration.
- dante.cpu_share.json: 2x2 channel DEP device with an 80% share of a single CPU core.
- dante.hardware_clocking.json: 2x2 channel DEP device with external hardware clocking support.
- dante.max.json: 64x64 channel DEP device with a single network configuration.
- dante.switched_network_redundant.json: 64x64 channel DEP device with a redundant network configuration and packets with DSA tags.

System and Configuration Checks

For DEP to function correctly and optimally the host system and DEP itself must be correctly configured. There is a script provided in the DEP release, dep/dante_package/development/dep_check.sh, which checks for common issues on the system and in the DEP configuration files which may result in DEP failing to start or failing to run optimally.

The script takes a single argument which is the path of the DEP root directory. The following checks are performed by the script:

- The host Linux kernel has been configured with the required settings as described in Appendix 1
- cgroups mounts have been set up as described in Appendix 1
- The system has enough random entropy
- The dante.json and config.json files exist
- The hardware timestamping configuration is correct. Hardware timestamping needs to be either disabled in dante.json, or if enabled, the network driver(s) must support the selected timestamping mode and a valid /dev/ptp device(s) must be correctly configured in config.json.

- The hardware clock configuration is correct. Hardware clocking needs to be either disabled in `dante.json`, or if enabled, a valid `/dev/extclkin` device must be correctly configured in `config.json`.
- The host system supports the CPU allocation configured in `dante.json` and `config.json`
- CPU frequency scaling is enabled, and if so that the 'performance' scaling governor is active on each CPU and set to use the maximum clock speed
- Interrupt coalescing on the Dante Ethernet interface(s) have been set to minimal values to minimize packet latency

PTP Timestamping Test Tool

DEP includes a standalone tool that can be used to test that the network driver(s) for the interface(s) in `dante.json` **fully** support the selected timestamping mode. Full support requires that a driver:

- Can not only be configured to use the chosen mode, but also
- Once configured thus, is successfully able to timestamp all PTP event packets in both directions

The tool can perform either a preliminary, configuration-only test or a full packet timestamping test. A preliminary test failure means that the tested timestamping mode **cannot** be selected in `dante.json` unless the driver(s) is updated and passes a subsequent retest.

`dep_check.sh` will run a preliminary check on not only the timestamping mode in `dante.json` but also the other modes, in order to provide the user with an initial recommended setting based on the configuration(s) supported by the driver(s). Following these preliminary checks, the script will ask the user to run a full test(s) for the mode in `dante.json` and/or the recommended mode (if different to the former).

It is strongly recommended that a full test(s) always be run before launching DEP for the first time on a platform, as any driver timestamping issues will result in DEP experiencing clock synchronisation problems. DEP should not be launched until a full test demonstrates that timestamping using the required mode reports no errors.

Please refer to the `README_ptp.txt` file under `dep/dante_package/development/tools` for details on how to run the tool.

Collecting Information for Audinate Support

Included in the DEP release is a script, `dep_support_collection.sh`, that you can run to collect logging and system information for analysis by Audinate technical support.

The script behaviour can be customised using its command line option as shown in the example below:

```
/dep/dante_package $ ./dep_support_collection.sh --help
[    INFO ] Usage: ./dep_support_collection.sh [OPTIONS]
[    INFO ]
[    INFO ] This tool collects diagnostic data to help debug issues with the DEP software.
[    INFO ]
[    INFO ] Options:
[    INFO ]   -c <path>    Specify the directory where DEP is installed.
[    INFO ]                Default is '/opt/dep'.
[    INFO ]   -l <path>    Specify the directory where DEP stores its log files.
[    INFO ]                Default is '/var/log'.
[    INFO ]   -o <path>    Specify the output directory for the final archive and any temporary
[    INFO ]                files or directories created in the process. This directory must be
[    INFO ]                writable by the user executing the script.
[    INFO ]                Default is the current directory, '/dep/dante_package'
[    INFO ]
[    INFO ] Examples:
[    INFO ]
[    INFO ]   ./dep_support_collection.sh -c /apps/dep -l /tmp/logs
[    INFO ]
[    INFO ]      Collects diagnostic data from a DEP installation in /apps/dep, DEP log files in
[    INFO ]      /tmp/logs, and stores the output in the current directory.
[    INFO ]
[    INFO ]   ./dep_support_collection.sh -c /apps/dep -l /tmp/logs -o /tmp/dep_diagnostics
[    INFO ]
[    INFO ]      Collects diagnostic data from a DEP installation in /apps/dep, DEP log files in
[    INFO ]      /tmp/logs, and stores the output in /tmp/dep_diagnostics.
[    INFO ]
[    INFO ]   ./dep_support_collection.sh -o /home/user/dep_diagnostics
[    INFO ]
[    INFO ]      Uses the default DEP installation and log file paths, and stores the output in
[    INFO ]      /home/user/dep_diagnostics.
```

After running the above script, you can supply the resulting archive file to Audinate Support to assist them in working on any issues you have encountered while running DEP.

DEP Performance

The performance of the platform that DEP is running on will in turn affect the following parameters of DEP performance:

- The maximum number of receive and transmit channels (and flows) that DEP can support
- The minimum Dante latency achievable

These parameters will also be affected by the chosen sample rate. For information on the Dante configuration parameters to control the channels, latency, sample rate and number of cores dedicated to DEP, please refer to Appendix 3: Dante Configuration. We also recommend that you configure the Linux kernel for low latency – see Appendix 1: Kernel and Platform Configuration for details.

This section assumes an understanding of the concept of Dante flows, which are a series of packets each containing up to 4 channels with a specific number of samples per channel. Unicast flows and packets assign slots for 4 channels of audio sent to a single destination, while multicast flows will only assign space for the number of channels they contain. More background is available at https://dev.audinate.com/GA/dante-controller/userguide/webhelp/content/routing_media.htm

The limiting factor for the number of channels that can be supported on a DEP device is the CPU time taken to process packets both by DEP and the protocol stack. This CPU time is proportional to the audio packets per second (PPS) sent and received by the device.

PPS is inversely proportional to the packet size, which is determined by the number of frames in a packet. One frame is defined as a single audio sample for each of the four channel slots in the packet (assuming it is a packet in a unicast flow). The latency and the number of frames per packet (FPP) are also related such that a smaller FPP value is required to achieve a lower latency. In summary, a smaller latency will result in higher CPU load and vice versa.

Dante devices with more than 2 channels create unicast flows containing four channels. Note also that the default maximum number of flows supported by DEP is either 2, or half the number of channels on the device, whichever is greater. A two channel DEP device supports 2 flows in each direction.

Maximum num supported flows = max(2, num channels / 2)

The maximum number of flows can be changed from the default values by setting "`audio.maxTxFlows`" and "`audio.maxRxFlows`" in the DEP device configuration. Please refer to Appendix 3: Dante Configuration for details of those config values. If increasing these values from the default, be mindful that it will require more processing power, and should only be done after qualifying that the platform can support the highest configured flow counts.

Note: For multicast flows, the FPP value is controlled by the sender. This means that even if your product uses 32 FPP in order to limit the PPS value and therefore support a higher channel count, you may still be receiving 16 FPP multicast flows. This may mean that your device may not be able to receive as many multicast channels as it can unicast channels. For example, when receiving or sending a flow from a hardware device such as Ultimo, it may force the flow to 16 FPP.

Un-encrypted Flow Testing

The following table illustrates the relationship between these parameters at 48 kHz sample rate, alongside example CPU usage values measured on an I.MX 8M Mini processor.

A few explanatory notes follow to aid in understanding the table:

- Audio conditions during the test: 48kHz sample rate, PCM24 encoding, no redundancy.
  - Doubling the sample rate or enabling redundancy on the device will double the PPS and approximately double the CPU load
- DEP set to run on the first 3 cores (numDepCores=3)
- DEP version tested: 1.1.1.1
- DEP running at 48kHz
- The main contributors to CPU usage are shown below. Note that DEP determines which CPU the DepApe threads run on but not the ethernet interrupt handling which will be system dependent.
  - CPU0: ethernet interrupt handling
  - CPU1: DepApe RX processing thread
  - CPU2: DepApe TX processing thread
- In addition to the DepApe process there are other DEP processes running. Those other processes are allowed to run on any of the three DEP cores. Their CPU usage is negligible compared to the DepApe.
- It can be observed that even with 0 flows DEP is loading CPU1 and CPU2 somewhat. This is because the DepApe wakes up periodically to check whether there are any audio packets to process.

Table 2 - DEP IMX8MM Performance (Unencrypted) - DEP 1.1.1.1

Channels	Unicast Flows (RX xTX)	Latency (ms)	FPP	CPU % usage		
				CPU0	CPU1	CPU2
Any	0	N/A	N/A	~1	~8	~11
4x4	1x1	1	16	~5	~7	~10
8x8	2x2	1	16	~10	~8	~13
16x16	4x4	1	16	~18	~11	~19
32x32	8x8	1	16	~30	~17	~29
4x4	1x1	2	16	~5	~7	~10
8x8	2x2	2	16	~9	~8	~12
16x16	4x4	2	16	~19	~11	~18
32x32	8x8	2	16	~31	~17	~29
64x64	16x16	2	16	~58	~34	~47

Dante device latency compensates for delay variation in the network, thereby enabling Dante to function with deterministic latency. In software Dante implementations, delay variation in packet processing in the device is also a factor. While you could run DEP processes scheduled alongside other processes running on the device, DEP performance may be more dependent on the CPU load profiles of these other processes, and so this would require setting a more conservative (higher) minimum latency setting to ensure reliability at that setting. As such it is recommended to dedicate up to 3 CPU cores to Dante in order to allow the lowest minimum latency setting possible for the given processor and platform. The number of cores you might want to reserve for Dante depends on the minimum latency value you wish to support as well as the number of channels on the device. The following table shows the relationship between channels, achievable latency and number of cores measured on an i.MX 8M Mini processor.

Table 3 - Channels, latency and number of cores, i.MX 8M Mini

Input / Output Channel Count	Latency	Number of cores
1 - 32	4ms	1
33 - 64	4ms	2
1 - 32	1-2ms	2
33 - 64	1-2ms	3

In order to validate your configuration, run a fully-subscribed device for an extended period. Monitor the CPU usage per core calculated from the change in /proc/stat over a period of time, and ensure that each core is not overloaded. Ensure that there are no late packets by observing the latency histogram in the Dante Controller.

DEP wakes up at regular intervals to process audio. At low flow counts or with no flows, DEP only wakes up for a short period of time, but the wake-up period may synchronise with the kernel sampling period. In this case, the CPU usage will be misrepresented above or below the correct value depending on the period alignment. At higher flow counts, this effect is minimal.

Note that you will need two other devices with an equal number of channels to utilize all the flows in your device. If possible, using hardware devices which require 16 FPP flows would be best, as it tests the worst-case scenario. If this is not possible you can estimate usage by extrapolating the CPU usage value, as this should increase linearly. You should ensure that there is sufficient headroom to allow for additional latency due to multiple switch hops in the network. This spreadsheet can be used to understand the contribution of network hops to the end-to-end latency of a Dante link: https://dev.audinate.com/documentation/evk/application-notes/system-latency/system-latency.xls

You may also encounter intermittent packet loss or audio issues even though the average CPU usage indicates the CPU is not overloaded. In this case you would need to tune your platform by identifying and fixing performance issues. For example, misbehaving device drivers may cause audio glitches by running for too long at high priority.

Encrypted Flow Testing

Dante Media Confidentiality introduces additional CPU overhead for encryption.

- DEP version tested 1.4.0
- DEP Tx channels set to “strict” encryption policy
- DEP running at 48kHz
- Encryption increases CPU usage in the DepApe processes on CPU1 and CPU2
- DEP set to run on the first 3 cores (numDepCores=3)
  - CPU0: ethernet interrupt handling
  - CPU1: DepApe RX processing thread
  - CPU2: DepApe TX processing thread

Table 4 - DEP IMX8MM Performance (Encrypted) - DEP 1.4.0

Channels	Unicast Flows (RXxTX)	Latency (ms)	FPP	CPU % usage		
				CPU0	CPU1	CPU2
4x4	1x1	1	16	~4	~6	~8
8x8	2x2	1	16	~5	~9	~12
16x16	4x4	1	16	~9	~13	~20
32x32	8x8	1	16	~17	~23	~35
4x4	1x1	2	16	~4	~6	~7
8x8	2x2	2	16	~5	~9	~12
16x16	4x4	2	16	~9	~14	~20
32x32	8x8	2	16	~18	~23	~35
64x64	16x16	2	16	~50	~43	~67

Networking

DEP supports using a single Linux network interface or two Linux network interfaces to enable redundancy. The interface names or indices are configured in `dante.json`. Best operation is achieved when the interfaces support hardware timestamping. The PTP devices corresponding to the configured network interfaces must be configured in config.json when hardware timestamping is enabled.

DEP supports connecting to a network directly via PHYs or via an onboard switch. The Marvell Link Street SOHO family of switch chips are supported. Non-redundant and redundant network configurations are supported. Dante redundancy can be achieved by using two PHYs or a switch configured with VLANs to support redundancy. If the DEP product uses a switch then products can also be connected in a daisy chain.

DEP switch configuration is performed using the Distributed Switch Architecture (DSA) subsystem in Linux. DSA can make external switch ports appear as virtual interfaces in Linux such that they can be configured using standard Linux network configuration. An example of kernel configuration for DSA can be found in the DEP-EVK-IMX8 platform configuration release package, along with a network configuration script, evk-network-config, which can configure the network interface for Dante redundant or daisy chain modes.

Note: `evk-network-configure` in DEP 1.0.0/1.0.1 EVK for i.MX8 should only be executed in the serial console of the EVK board, not from an SSH session to the board - execution in an SSH session will fail due to the network change caused by the script. This will be fixed in a future release.

Note: Network configuration using Dante Controller is currently not supported in DEP. Specifically, changing between switched and redundant mode, and changing between dynamic and static IP addressing using Dante Controller is not supported.

Example configuration files for different setups can also be found in the Example Applications and Configuration section of this document.

DSA Example: DEP-EVK-IMX8

The DEP-EVK-IMX8 platform uses a Marvell 88e6320 switch to support two external ethernet connections, both accessible through the single gigabit ethernet MAC on the i.MX8 family of SoCs.

The two PHYs are connected to switch ports 3 and 4, with port 5 in RGMII mode for the i.MX8. Using the DSA feature on Linux, the two PHYs are represented as virtual interfaces “lan3” and “lan4”.

The switch uses a header tag for packets to mark where packets came from, or where they should be routed to. The DSA driver system then uses these tags to forward packets from the built-in MAC (Freescale Ethernet Controller or FEC) to the correct virtual interface. Similarly, packets outbound from either lan3 or lan4 interfaces have a tag appended, before being passed through eth0 (FEC interface) and forwarded to the PHY by the switch.

Forwarding between switch ports is enabled with a bridge interface across the virtual interfaces corresponding to the desired switch ports. The figure below shows the data flow paths between different networking components on the DEP-EVK-IMX8 platform. Note that DSA requires a MDIO interface to the switch for configuration and status updates. The state of the virtual interfaces and packet statistics are therefore kept up to date with the physical ports.

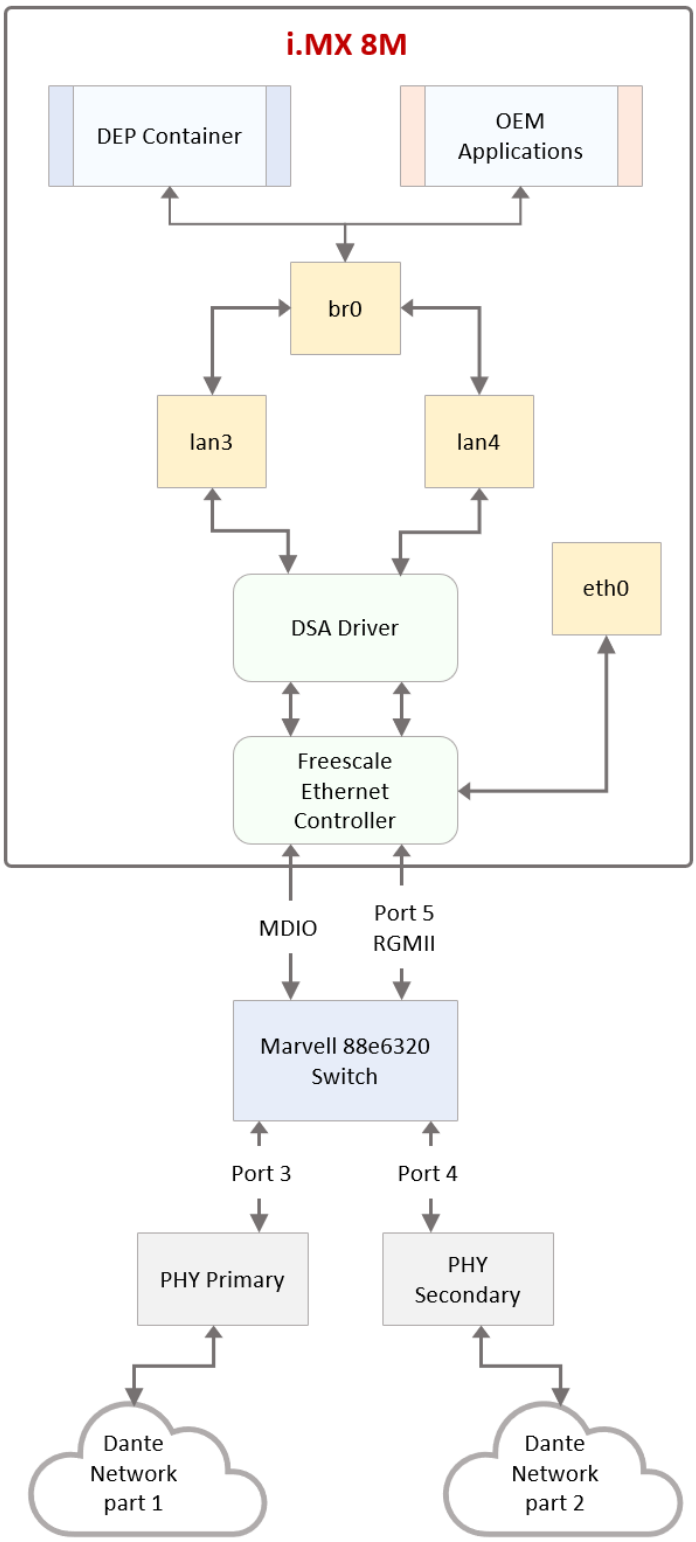


Figure 5 - DSA Example: DEP-EVK-IMX8

Similar arrangements can be used to enable various networking arrangements with DEP on a SoC. The next sections demonstrate different arrangements.

DSA with Packet Timestamping

When using DSA, the virtual interfaces can't directly make use of the IEEE1588 packet timestamping available through the FEC. DEP supports using an alternative "hardware" interface for timestamping when virtual interfaces are used. Similar to the `"interfaces"` field of `dante.json`, the `"hardwareInterfaces"` field takes an array of interface identifiers (names or indices). Each value in `hardwareInterfaces` must correspond to the same positioned value in the `"interfaces"` array. On the DEP-EVK-IMX8, eth0 is the hardware interface doing the timestamping for both lan3 and lan4. And so when in redundant mode, lan3 and lan4 are the interface values, whilst eth0 is the hardware interface value - as in the example below.

```
"hardwareInterfaces" : [ "eth0", "eth0" ]
```

When DSA is being used with a switch and the interface(s) selected for DEP use have DSA-tagged packets, the "`dsaTaggedPackets`" value should be set to true.

Dante Daisy Chain Mode on a Platform with a Switch

Figure 6 below shows the DEP-EVK-IMX8 at the top, and another DEP device without a switch on the bottom right. A bridge interface is created across the virtual interfaces corresponding to lan3 and lan4. DSA configures the switch to offload packet forwarding between PHY3 and PHY4, as indicated by the dotted line between them. Via this path, the Dante device on the bottom right is daisy chained by the Marvell switch through to the Dante network.

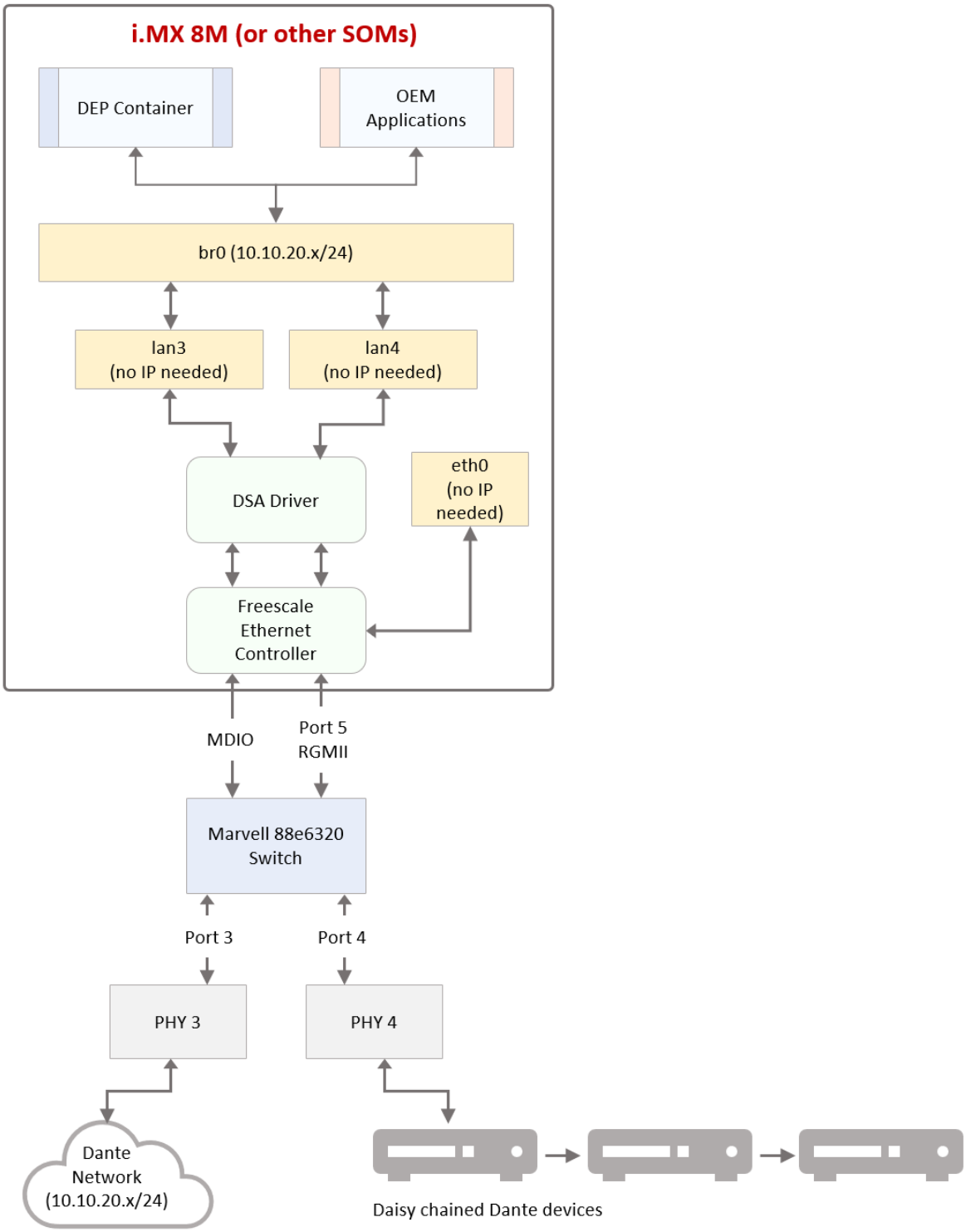


Figure 6 - Daisy Chain Mode on a Platform with a Switch

As shown in Figure 6 above, this configuration has 4 network interfaces - however DEP is only aware of the switch interface (br0). In this configuration:

1. Interface br0 is the effective network interface with an IP address assigned.
2. Interface lan3 represents the primary PHY, but it is not exposed to DEP and has no IP address assigned.

3. Interface lan4 represents the secondary PHY, but it is not exposed to DEP and has no IP address assigned.
4. Interface eth0 represents the Freescale MAC between the switch and CPU, but it is not exposed to DEP and has no IP address assigned.

An example of the “network” section in `dante.json` for the setup is as follows:

```
“network” :
{
    “interfaceMode” : “Switched”,
    “interfaces” :
        {
            “br0”
        }
},
```

Dante Redundancy on a Platform with a Switch

On a DEP platform with an integrated hardware switch, Linux DSA can also configure each port on the switch as an independent network port for Dante redundancy. As shown in Figure 7 below, this configuration creates 3 interfaces - of which lan3 and lan4 represent the switch ports to be used for DEP redundancy.

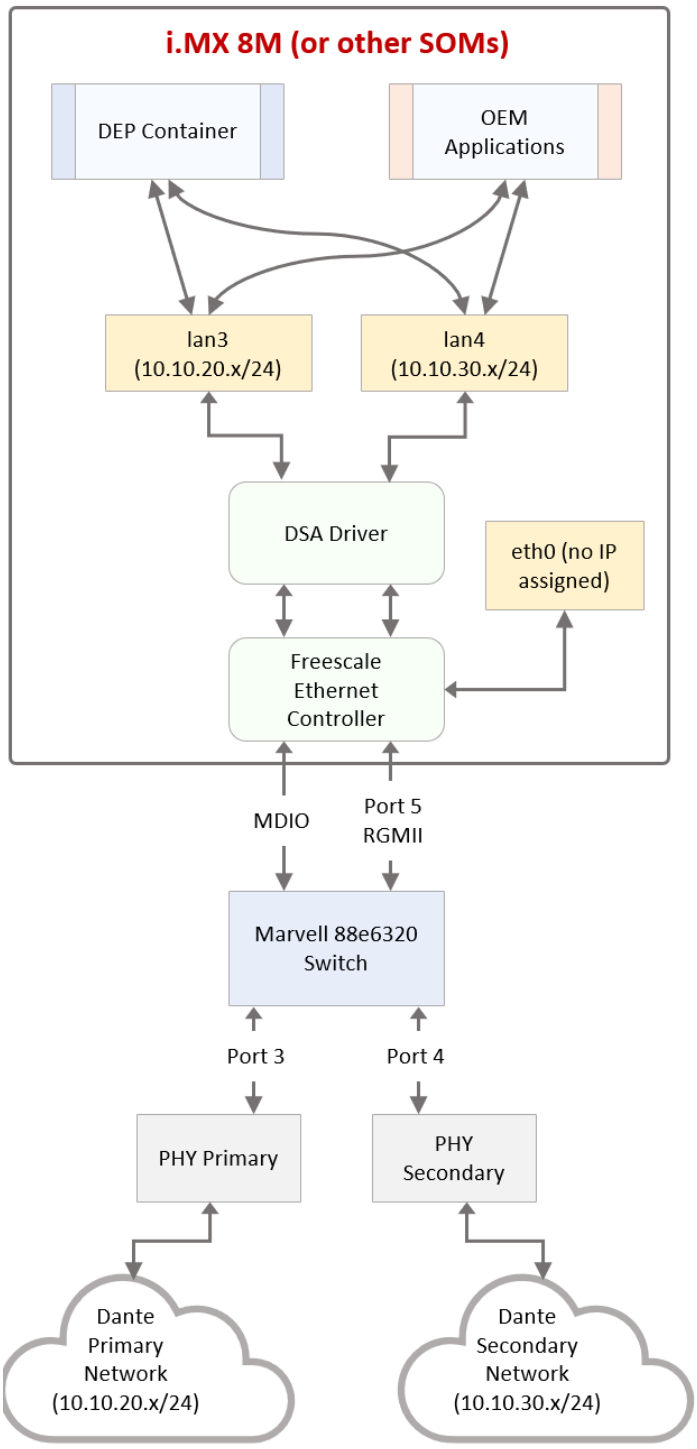


Figure 7 - Dante Redundancy on a Platform with a Switch

In this configuration:

1. Interface lan3 represents the primary PHY with IP of primary network assigned.
2. Interface lan4 represents the secondary PHY with IP of secondary network assigned.
3. Interface eth0 represents the Freescale MAC between hardware switch and CPU, but it is not exposed to DEP and has no IP address assigned.

An example of the “network” section in `dante.json` for the setup is as follows:

```
"network" :
{
    "interfaceMode" : "Switched",
    "interfaces" :
        {
            "lan3",
            "lan4"
        }
},
```

Platform with a Single PHY

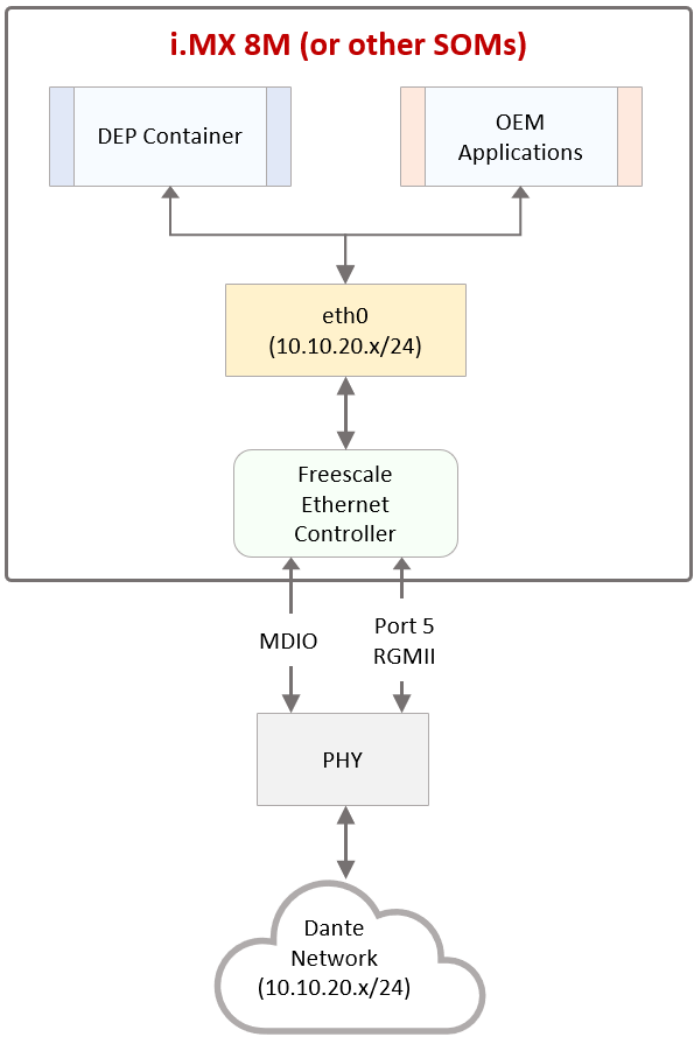


Figure 8 - Platform with a Single PHY

If there is only one PHY on the platform, DEP runs in non-redundant mode.

An example of the “network” section in `dante.json` for the setup is as following:

```
"network" :
{
    "interfaceMode" : "Direct",
    "interfaces" :
        {
            "eth0"
        }
},
```

DEP Redundancy on a Platform with Multiple PHYs

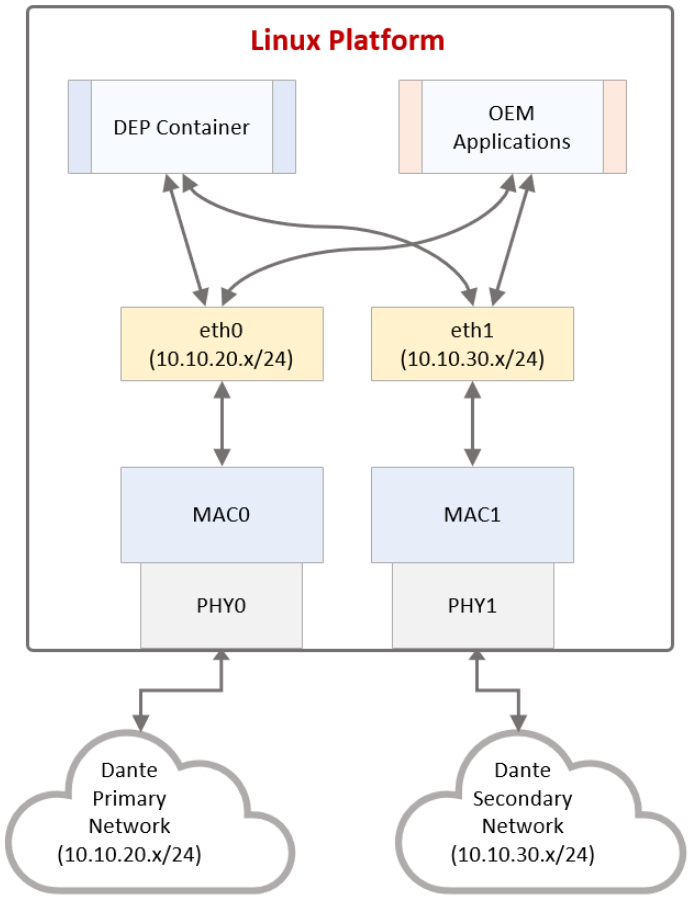


Figure 9 - DEP Redundancy on a Platform with Multiple PHYs

On a platform with multiple PHYs, Dante redundancy can be configured by including these interfaces in `dante.json`.

An example of the "network" section in `dante.json` for the setup is as follows:

```
"network" :
{
    "interfaceMode" : "Direct",
    "interfaces" :
        {
            "eth0",
            "eth1"
```

```
        }
},
```

DHCP Configuration

If the DEP host platform is configured to use DHCP, in order to interoperate with other Dante devices, the DHCP client used **must support both DHCP and link local addressing**. The dhcpcd client used on DEP-EVK-IMX8 meets this requirement. Where multiple ports are used (as in redundant DEP configuration), IP addresses should be assigned for each interface corresponding to an external interface. The interface connecting the SoC to the switch (e.g. eth0) should not have an IP address.

Take the aforementioned daisy chaining setup for example, as the only interface visible to DEP is “br0”, DHCP client is only required on this interface:

```
dhcpcd -b br0
```

DHCP client shouldn’t be started on any other interfaces, as they don’t require IP addresses.

Secondary Port Link Local Addressing

Dante requires that the secondary port of a platform supporting redundancy has a different link-local address space to the primary port. In the absence of a fixed address or a DHCP server, the secondary port should obtain a link-local address within the 172.31.x.x/16 subnet. The primary port should use the 169.254.x.x/16 subnet. Such a configuration may require a patch to DHCP clients. An example patch is supplied in Appendix 4: dhcpcd Patch for 172.31.x.x Link Local Addressing.

Firewall Configuration

This product requires specific ports to be available. If there is a firewall running on the device you can either (a) disable the firewall, or (b) add the following Dante ports to the firewall exceptions.

- External ports: 319, 320, 4440, 4455, 4321, 5004, 5353, 8000, 8002, 8700-8708, 8800, 9875, 14336-14591
- Internal ports: 4444, 8753, 8001, 8900

To license the evaluation x86_64 DEP you will need to be able to access the following URL: software-license-dep.audinate.com : port 443

Network Interrupt Coalescing

Network Interrupt coalescing reduces CPU usage by not interrupting the CPU on every packet that is received or transmitted. However, this comes at a cost of higher latency for packets traversing the protocol stack. Therefore, it is recommended that you configure these settings to minimise moderation. The following command will display the configuration options available for your Ethernet driver.

```
ethtool -c eth0
```

The following command ensures that each packet is received and transmitted without delay.

```
ethtool -C eth0 rx-usecs 1
ethtool -C eth0 tx-usecs 1
```

mDNS Visibility

DEP maintains mDNS records necessary to advertise DEP by its Dante Name. For example, given a device on the local network named `My-DEP-Device` with the hostname `DEP-EVK-XXXXXX`, on a machine running an mDNS client, both of the following should resolve:

```
ping My-DEP-Device.local
```

```
ping DEP-EVK-XXXXXX.local
```

We do not recommend changing the hostname of a device running DEP. If the hostname is changed, DEP must be restarted to pick up the change.

Other Configuration

- A valid network loopback interface is required in order to run DEP.
- Network drivers that have NAPI enabled may affect packet processing latency. For best performance, NAPI should be disabled or the `ksoftirqd` thread and should be set to a real-time priority.

Clocking

IEEE 1588 Precision Time Protocol (PTP)

All Dante-enabled devices use IEEE 1588 Precision Time Protocol (PTP) across the network to synchronise their local clocks to a leader clock, providing sample-accurate time alignment throughout the network.

One Dante device will be elected as the PTP leader clock for the network; all other Dante devices will synchronize their PTP clocks to the elected leader clock. Although many Dante devices may be capable of becoming PTP leader clock, only one device will win the election. Devices with clock inputs (e.g. word clock or AES3) will be preferred in the election process and a tie-breaker rule is used if several equivalent candidate leader clocks are available.

DEP, like other software Dante implementations, uses software algorithms to smoothly track “PTP time.” Accurate time tracking relies upon the timestamping of packets. Particularly when DEP’s hardware clocking feature is enabled, performance relies upon hardware packet timestamping support at the PHY or MAC layer. Hardware timestamping is recommended when the hardware audio clocking feature is used.

Hardware Timestamping

DEP has an optional feature to turn on Network Interface Card (NIC) level hardware timestamping of PTP packets, adjustable by setting the “`enableHwTimestamping`” option in `dante.json`. If the NIC that DEP is using supports hardware timestamping, this feature can be turned on, and will greatly increase the stability of PTP. By default, hardware timestamping is disabled, but for best performance we recommend enabling it, particularly if the hardware clocking feature is enabled. See the pre-qualification guide for details on hardware timestamping, including how to check whether your NIC supports it.

If hardware timestamping is enabled, DEP will be preferred as the leader clock over other software Dante devices such as Dante Via, or other instances of DEP without hardware timestamping.

Audio Muting

Audio will be muted if DEP’s local PTP time is not synchronised with the PTP leader. This will normally occur when DEP is first started and typically lasts for a few seconds.

When audio is muted, DEP will zero the samples in the audio packets.

Hardware Clocks

An optional feature of DEP is the synchronisation of hardware audio clocks to the rest of the Dante network clock.

Figure 10 depicts the different components involved in bridging the Dante clocking system and a hardware audio clocking system. The large, dotted rectangle encompasses the elements that are strictly prescribed by this feature. A control loop is used to bridge PTP time to the local hardware audio clocks, using rate and phase feedback to make small rate adjustments. A custom platform-specific driver enables the clock feedback path.

When the hardware clocking feature is used, all I2S connections should be externally clocked by the clocking circuit controlled by DEP. Note the MCLK, SCLK, and LRCLKs generated by the clocking circuit in Figure 10 drive both the I2S/TDM interface on the SOC and the CODEC.

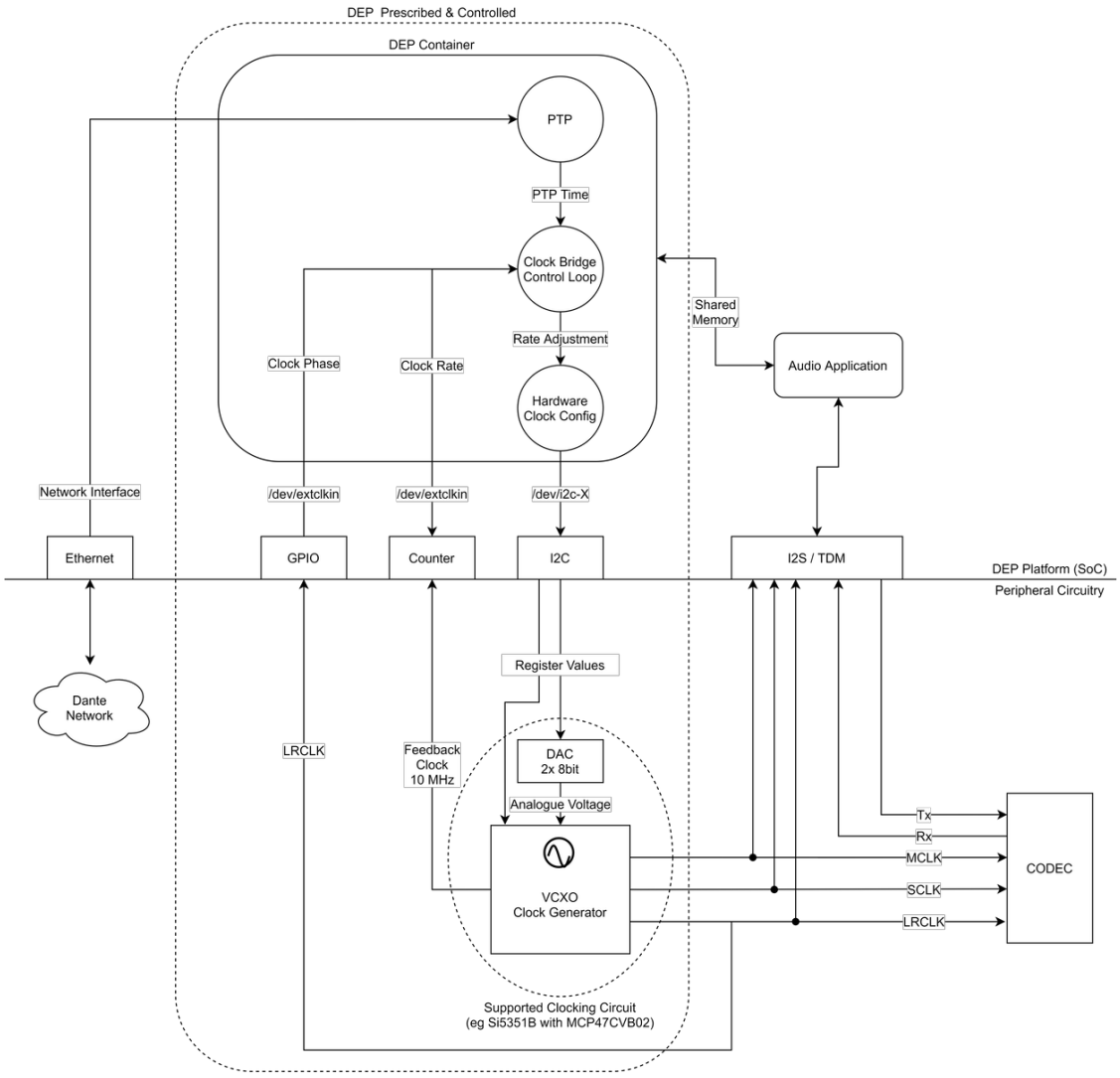


Figure 10 - Hardware clock system components and interconnections

Clock Feedback

The clock rate and phase information are accessed by the DEP container through platform-specific drivers. Example drivers are provided with the DEP-EVK-IMX8 and are known as `"extclkin-gpt"` and `"extclkin-gpio"`.

`"extclkin-gpt"` is providing the 10MHz clock feedback count through the i.MX general purpose timer (GPT) peripheral, which is an externally-clocked counter. The driver is loaded as a Linux kernel module and mounts a character device node at `/dev/extclkin`. The `"extClockInputDev"` value in `dante.json` can be used to specify an alternate path to the character device.

"extclkin-gpio" is determining the phase of the I2S LRCLK through a GPIO input pin. The driver is loaded as a Linux kernel module and mounts a character device node at /dev/extclkin-gpio. The "extLrclockInputDev" value in dante.json can be used to specify an alternate path to the character device.

Both devices and their device nodes must be mapped into the DEP container, and specified within the container settings (config.json) - see Appendix 2: Container Configuration.

The driver behaviour is specified by platform-specific examples. Future improvements to the DEP clocking system may rely on upgrades to the driver's behaviour and implementation. Examples are delivered as part of DEP-EVK-IMX8. Note the feedback clock frequency is not configurable, operating at 10 MHz +/- the adjustment in line with the I2S clocks.

I2S / TDM Clocks

The clocking circuit will generate three I2S clock outputs and a fourth feedback clock, used for rate control.

- Word clock, left-right clock or frame sync (LRCLK)
  - A 50% duty cycle clock with the same frequency as the sample rate. Used to synchronise samples.
- Bit clock, serial clock (SCLK)
  - An integer multiple of the word clock, with a pulse for each discrete bit of data.
- Master clock (MCLK)
  - A high-speed integer multiple of the word clock. Can be used to clock peripheral devices like CODECs and DSPs.
- Feedback clock (FBCLK)
  - A high-speed clock used to provide rate information to DEP through a hardware counter.

All four clocks are rate-adjusted on the order of parts per million, to ensure synchronous operation with other Dante devices on the network. The LRCLK is also fed back to a GPIO pin as a phase measurement.

The MCLK frequency is not configurable. For 48k and 96k sample rates, the MCLK runs at 24.576 MHz. At 44.1k and 88.2k sample rates, the MCLK runs at 22.5792 MHz.

Bit Clock Configuration

The integer ratio between the bit clock and the word clock is dependent on two factors - the number of TDM channels (N) and the bit depth of each sample (B). "SCLK"="LRCLK"×N×B

The "bitClocks" section in the dante.json configuration file permits configuring these two parameters by taking an array of bit clock objects. Each object must specify both the bit depth and the number of TDM channels. Configurations for specific sample rates are possible by adding a sample rate field to the object.

The valid values for these parameters are shown below:

- Number of TDM channels (N) must be one of: 2, 4, 8, 16, 32
- Bit depth (B) must be one of: 16, 24, 32

The resulting integer ratios supported are: 32, 48, 64, 96, 128, 192, 256, 384, 512, 768, 1024

Sample Rate Changes

All clocks are disabled whilst a sample rate change is occurring. The clocks are then re-enabled at the new sample rate.

Supported Clocking Circuits

This subsection lists the supported clocking circuits available in the accompanying release. These circuits are abstracted through the `"hardware clock config"` block in Figure 10 and require an I2C bus to operate. This block is configured through a standard set of configurable parameters.

Si5351B with MCP47CVB02

This circuit is a derivative of the circuit used on many existing Dante implementations. It is centred around the Si5351B clock generator, with a voltage-controlled oscillator (VCXO) providing rate adjustment to enable smooth synchronisation with PTP time. A 2-channel, 8-bit DAC is used to provide a fine-grained voltage input to the Si5351B. At the time of publication, the only DAC supported is the MCP47CVB02. Consult the reference design schematics for the precise circuit.

Configurable Parameters

- `"circuitName"` must always be `"Si5351B with MCP47CVB02"`. Other values are reserved for future use.
- `"circuitRevision"` must be `0`. Other numbers are reserved for future use.
- `"i2cBus"` must be the Linux I2C bus device on which both the Si5351B and the MCP47CVB02 are connected. For example, if the second I2C bus is being used, this is generally available as `/dev/i2c-1`
- `"i2cAddr"` is the I2C address of the MCP47CVB02. The Si5351B must always have an I2C address of `0x60` (default).

Configuration of Si5351B Output Clock Lines

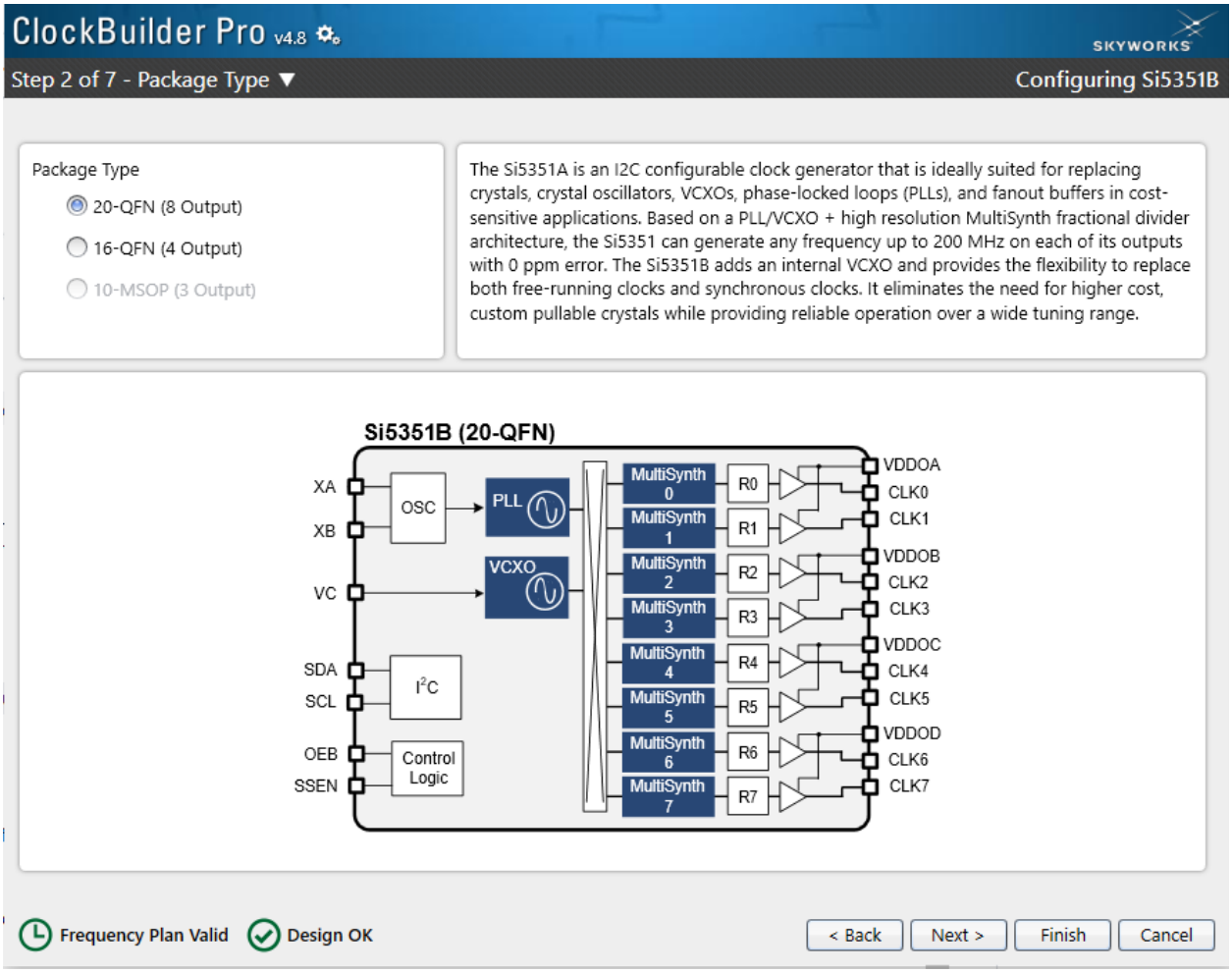
DEP makes use of the first four Si5351B clock outputs, and configures them as shown in the table below. DEP configures those clock outputs during startup and upon sample rate changes. Also, as described previously, these four clocks are subject to rate adjustment by DEP to maintain synchronisation with Dante time.

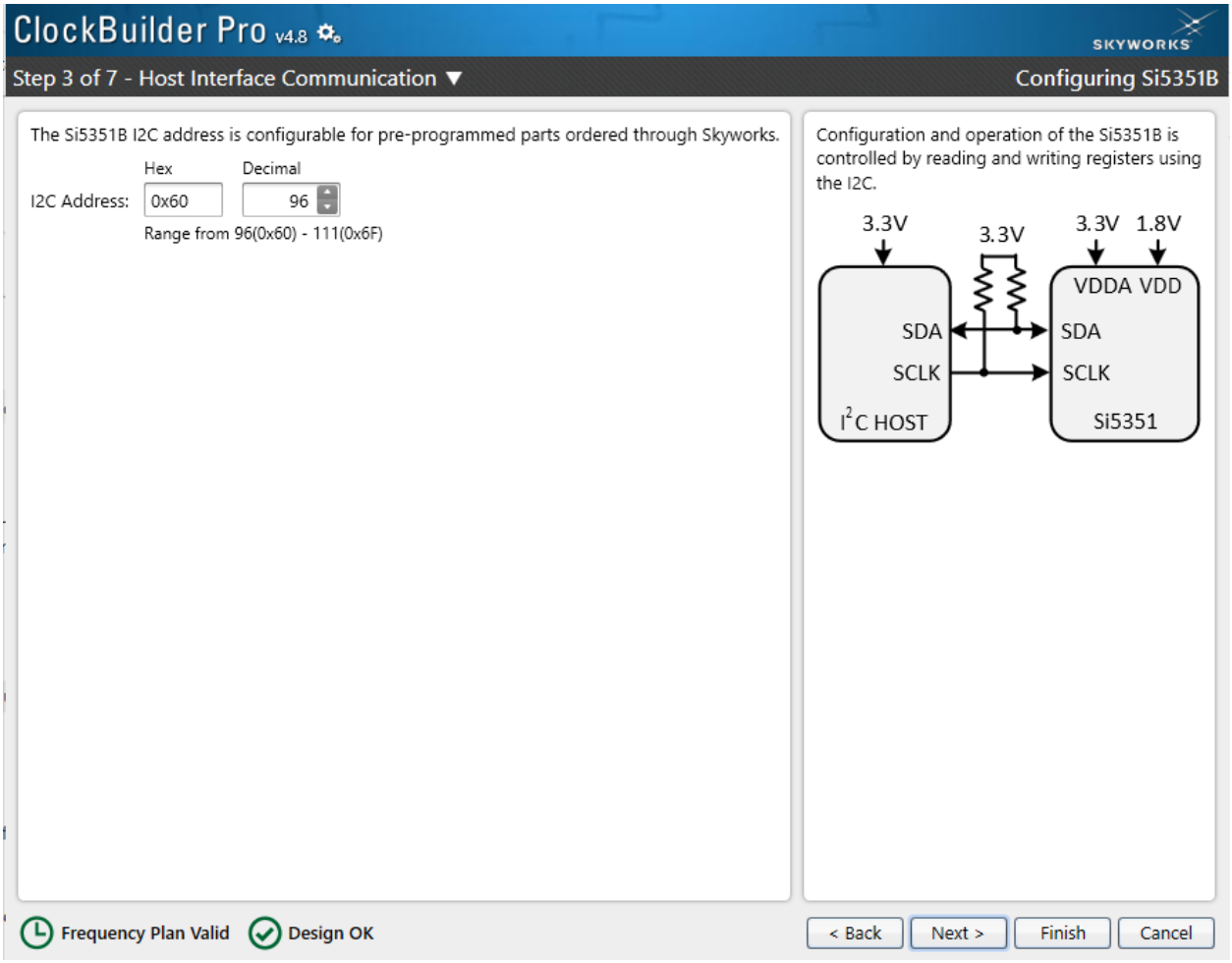
Clock Output	PLL	Function
OUT0	B (VCXO)	SCLK
OUT1	B (VCXO)	LRCLK
OUT2	B (VCXO)	MCLK
OUT3	B (VCXO)	FBCLK

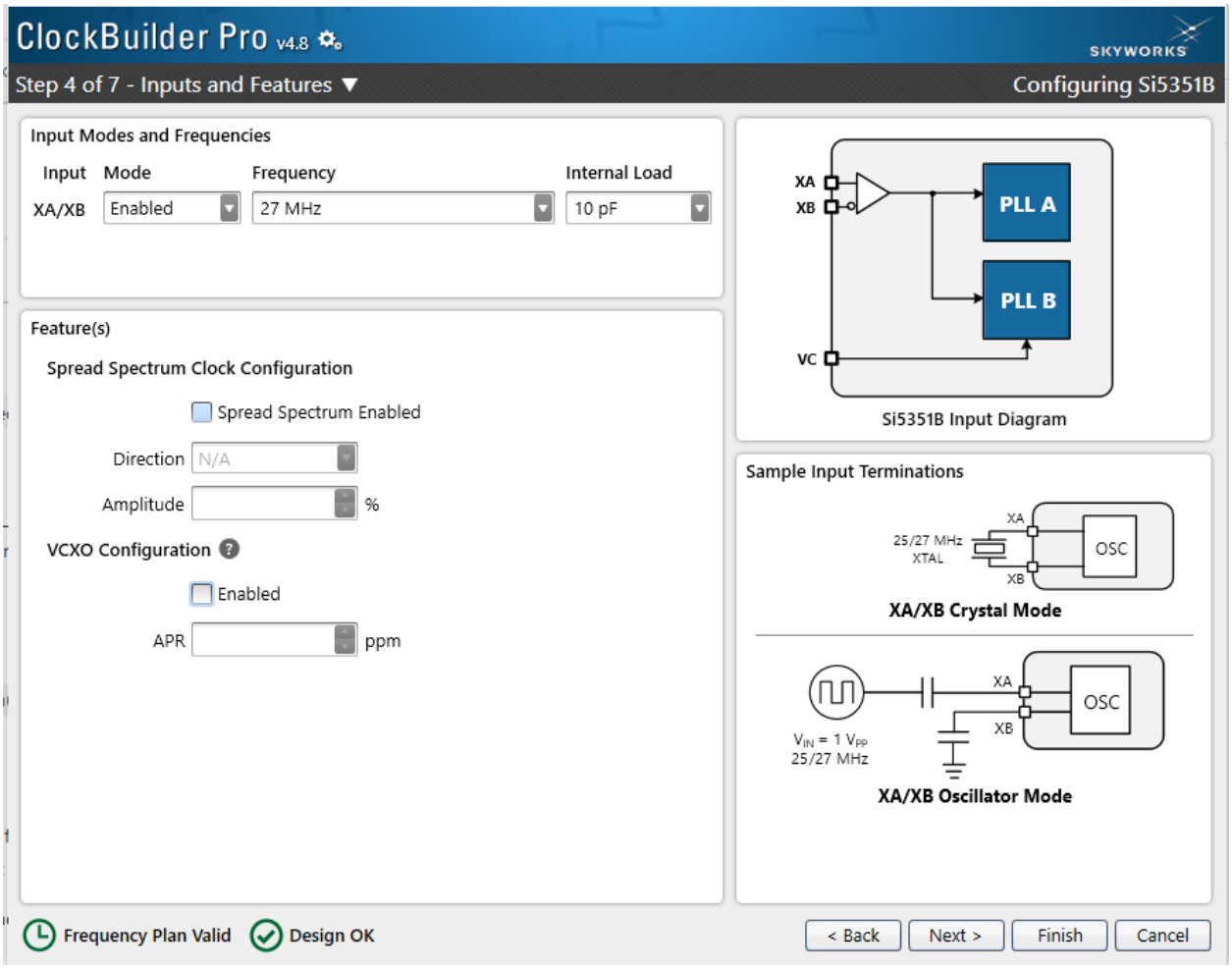
The remaining four clock outputs (OUT4-OUT7), hereto referred to as OEM clocks, are not used by DEP and can be configured by the OEM prior to starting DEP. The following restrictions must be adhered to when using the OEM clocks:

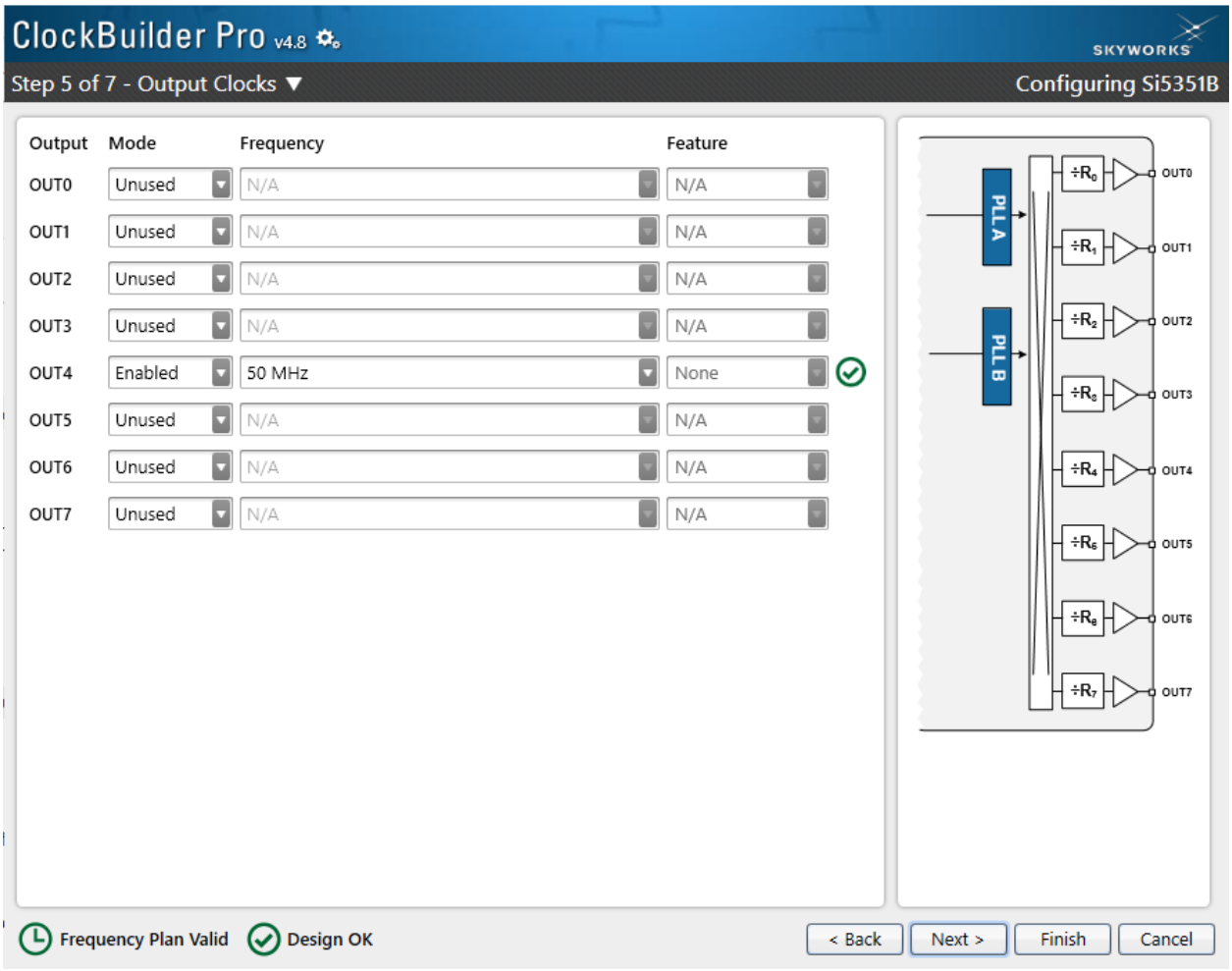
- Only PLLA can be used. That is, only non-VCXO based free-running outputs with fixed frequencies can be used. This is because the PLLB configuration is subject to change by DEP, and such changes may affect the final output frequency of any clock output that uses PLLB.
- OEM configuration must not change any of the PLLB, VCXO and OUT0-OUT3 related settings. Those are owned by DEP and will be overwritten when DEP start up.

For reference, the following SkyWorks ClockBuilder screens are shown as an example of a fixed 50Mhz clock configured on OUT4. In addition to the Output screen, the other screens are also shown with settings which are compatible with DEP.









Audio Interface

DEP Audio Interface

The DEP audio interface currently supports the following configuration:

Feature	Supported
Audio channels	128x128 (input x output; configurable)
Sample rate	44.1 kHz, 48 kHz, 88.2 kHz, 96 kHz
PCM access type (buffer configuration)	Non-interleaved
PCM sample format	32-bit, native endian

Note: The audio interface is currently locked to 32 bits per sample. On-the-wire PCM encoding can be either 16, 24 or 32 bits. When the on-the-wire-encoding is less than 32 bits, incoming (Rx) audio samples have their low bits zero padded and outgoing (Tx) audio samples have their low bits truncated as they are moved between the buffers and packets.

The DEP provides a shared memory interface to your audio application. Access to the shared memory is via an open-source API provided as part of DEP. OEMs can use the API to produce and consume audio from the DEP audio engine. Note that the shared memory layout is subject to change and should not be directly accessed by OEMs. The API includes automatic disconnection / reconnection and will notify the host application via a callback when the connection to the audio processing buffers is made or lost.

Audio Interface Timing

All Dante devices use a shared audio clock that is independent of any media transport. This audio clock is synchronized over the network using the IEEE 1588 clock synchronization protocol. The DEP Audio interface produces and consumes audio samples at the rate of the shared Dante clock. As Dante network time elapses, received blocks of samples are made available to the OEM application and transmit samples are consumed from the OEM application. In both cases the audio rate is independent of whether any audio is sent / received on the network, and how that audio might be packetized. It is also independent of any packetization or transmission delays on the network.

Audio Interface Metadata Block

In addition to the audio data buffers, the audio interface contains a metadata block. An API is provided to obtain a pointer to the metadata block. The metadata block contains information describing the audio data that may be useful to audio applications. The metadata includes:

- Sample rate
- PCM encoding in the buffers (currently this is always PCM32)
- Number of TX and RX channels

- Timing information. The provided APIs and examples show how the timing information can be used to synchronise the application with DEP.
- The epoch time indicates the PTP time at which clock synchronisation with the leader was achieved. The epoch seconds and samples are both zero when DEP is first started and are updated whenever synchronisation is gained. If synchronisation is temporarily lost, the epoch is not reset to zero. An audio application can use a non-zero epoch time as an indication that any hardware clocks have been synchronised with the local PTP time and are thus ready to use.

Sample Rate Change

If the Dante sample rate is changed, the application will be notified via an invocation of the registered reset callback. The new sample rate can then be retrieved from the audio interface metadata block.

Example Audio Applications

Each DEP release includes an examples package. Consult the `dep_examples/README.md` file in the examples package for details on how to build and run each example application.

DepLoopback

DepLoopback is a software application that copies audio data received by the DEP input back to the DEP output. below shows the relationship between DEP, the DEP audio interface and the DepLoopback application.

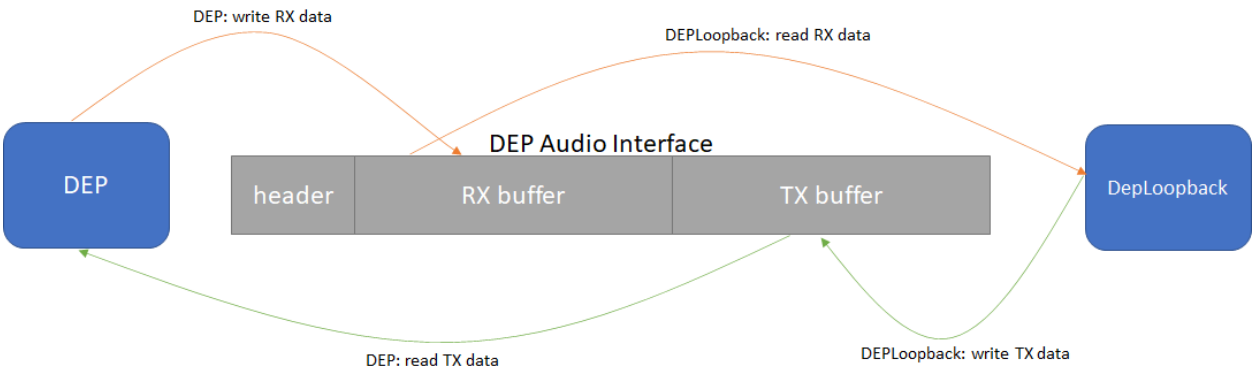


Figure 11 - DepLoopback architecture

The application illustrates the following concepts:

- Connecting to the DEP shared memory audio interface. A successful connection indicates that DEP has fully initialised and has started transferring audio data into and out of the audio data buffers in the shared memory.
- Timing. The example implementation contains a run loop that demonstrates how to wait for signalling from DEP to indicate the end of a timing period and how to use the timing information in the audio interface to calculate the number of audio samples contained in the timing period.
- RX/TX buffer management:
  - Get access to the RX and TX audio buffers for each channel.
  - Read from the RX channel buffers and copy to the corresponding TX channel buffers.

- Account for latency by not placing RX data in the exact corresponding TX frame but rather offset by a number of samples.

Note: The default TX latency of the DepLoopback application is 48 samples, which is 1ms at 48kHz sample rate. This is an optimized value for the DEP EVK for iMX8. Therefore, for proper functioning of DepLoopback in a dual or single core system, it may be necessary to set this latency to 192 samples, which corresponds to 4ms at 48kHz sample rate.

DepAlsaBridge

DepAlsaBridge is a software application that provides a bridge for audio data between DEP and an ALSA sound card. , assuming that the ALSA reference clock is synchronised to the Dante PTP time.

Audio data received by DEP is sent to the soundcard for playback and audio data captured from the soundcard is passed to DEP for transmission. below shows the relationship between DEP, the DEP audio interface, the DepAlsaBridge application and the ALSA soundcard.

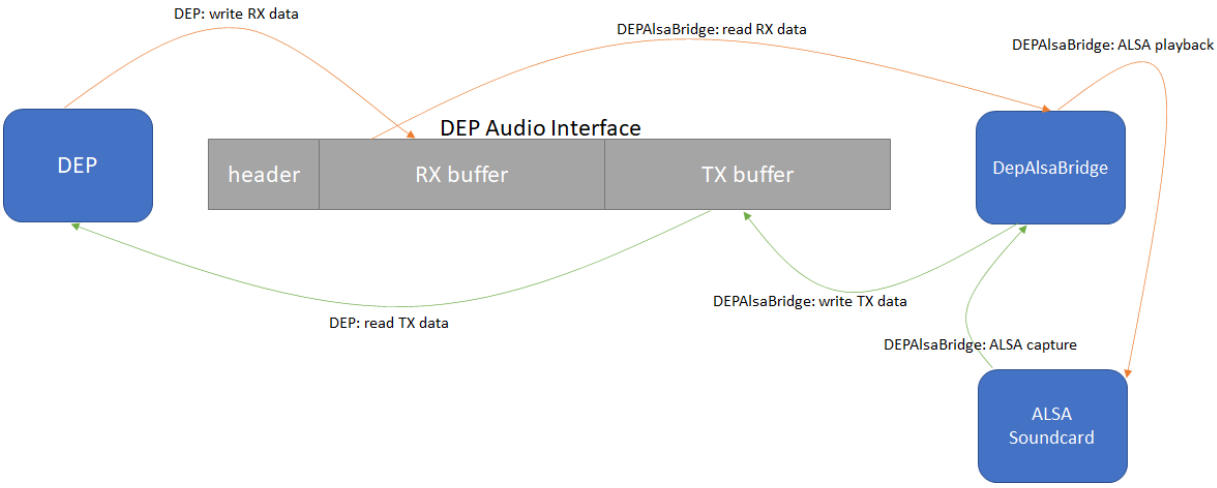


Figure 12 - DepAlsaBridge architecture

DepAlsaBridge includes the same buffer and timing aspects as the DepLoopback application. In addition, it also illustrates the following concepts:

- Initialise ALSA sound card. Some sound card parameters are derived from DEP values including number of TX/RX channels and sample rate.
- The application currently supports ALSA buffers in non-interleaved mode and with a 32 or 24 bit native endian format. It shows how to transfer audio data between the DEP audio interface buffers and the ALSA buffers. The DEP audio interface PCM format is 32 bit native endian, so the low bits in the data are truncated if the ALSA buffers are configured for 24 bit.
- The application exposes a few parameters that affect latency of the ALSA soundcard, and can be tuned for the required level of performance:
  - The number of ALSA frames in each buffer period
  - The total number of periods in the ALSA buffers
  - The number of samples to offset when copying ALSA captured data to the DEP TX buffers. The default is approximately 1ms at the configured sample rate.

Note that the approach used by this example assumes that the ALSA audio clock is synchronised to the PTP time. This could be done by enabling the Hardware Clocking feature and ALSA is clocked from the disciplined clock.

dinfo

dinfo is a software application that prints information about the DEP shared audio buffers configuration. On starting up, dinfo reads configuration information from the header and prints it to the terminal. The figure below shows the relationship between DEP, the DEP audio interface, the dinfo application and the user terminal.

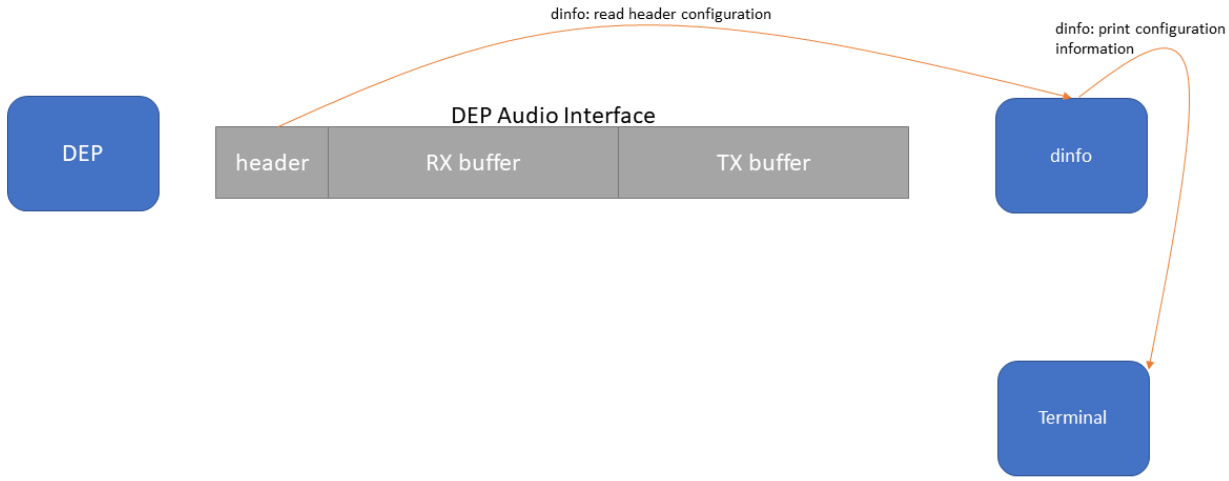


Figure 13 - dinfo architecture

dinfo illustrates the following concepts:

- Read DEP Audio Interface header information
- When to retrieve header information. The 'reset' callback is called by the DanteRunner before any audio data is requested or written by DEP, so this is correct place to configure settings for your application.

dplay

dplay is a playback software application that plays audio data from a file to the DEP output. When samples are requested by DEP, the application reads that number of samples from a flac/mp3/wav file and writes the audio data to DEP audio buffers for transmission. The figure below shows the relationship between DEP, the DEP audio interface, the dplay application and the audio file.

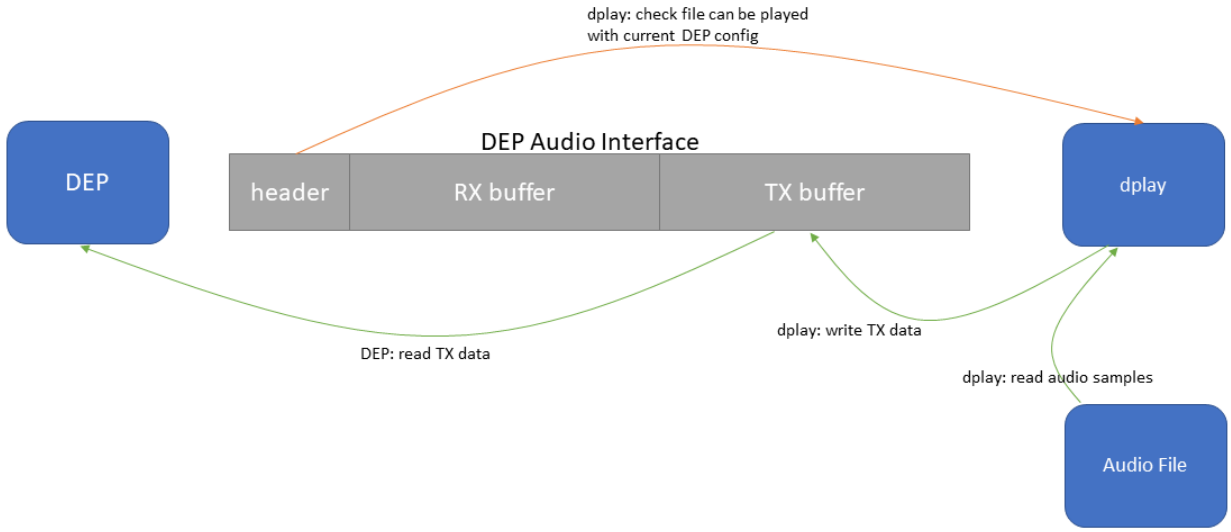


Figure 14 - dplay architecture

dplay includes the same buffer and timing aspects as the DepLoopback application. In addition, it also illustrates the following concepts:

- Reformatting of audio data for transmission by DEP. The supplied header files handle transformation conversion of the audio data to an interleaved PCM format, while the application illustrates the further conversion to a non-interleaved PCM format for DEP. Additionally, this example illustrates padding out the 16-bit samples decoded from mp3 files to be 32-bit samples.
- Checking audio files are compatible with DEP before playing them. The application compares the sample rates of the audio file and DEP before commencing playback. The application also checks that there are sufficient channels to play a file before commencing playback. If a configuration error is detected an error message is printed and the application exits.

drecord

drecord is a software application that records audio data received by the DEP input. The figure below shows the relationship between DEP, the DEP audio interface, the drecord application and the audio file.

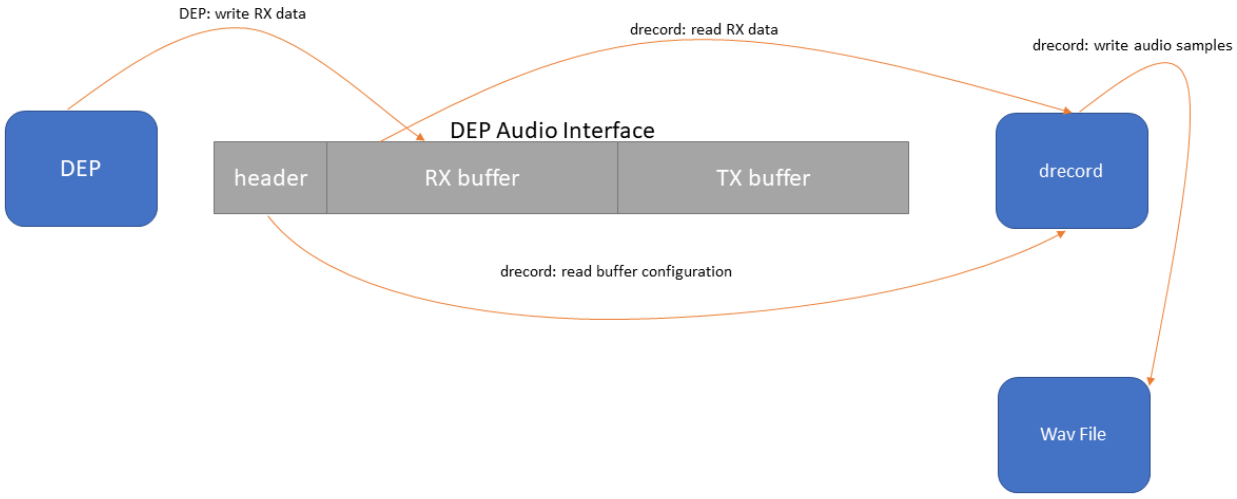


Figure 15 - drecord architecture

The application illustrates the following concepts:

- Converting from non-interleaved RX data to the interleaved PCM format used in wav files.
- Reading configuration information from the DEP Audio Interface headers for the purposes of writing file metadata.
- Keeping track of the total number of samples recorded.

dsoundcard

dsoundcard is an example ALSA plugin that allows playback to and capture from DEP by any application that supports ALSA. The figure below shows the relationship between DEP, the DEP audio interface, the dsoundcard plugin and ALSA.

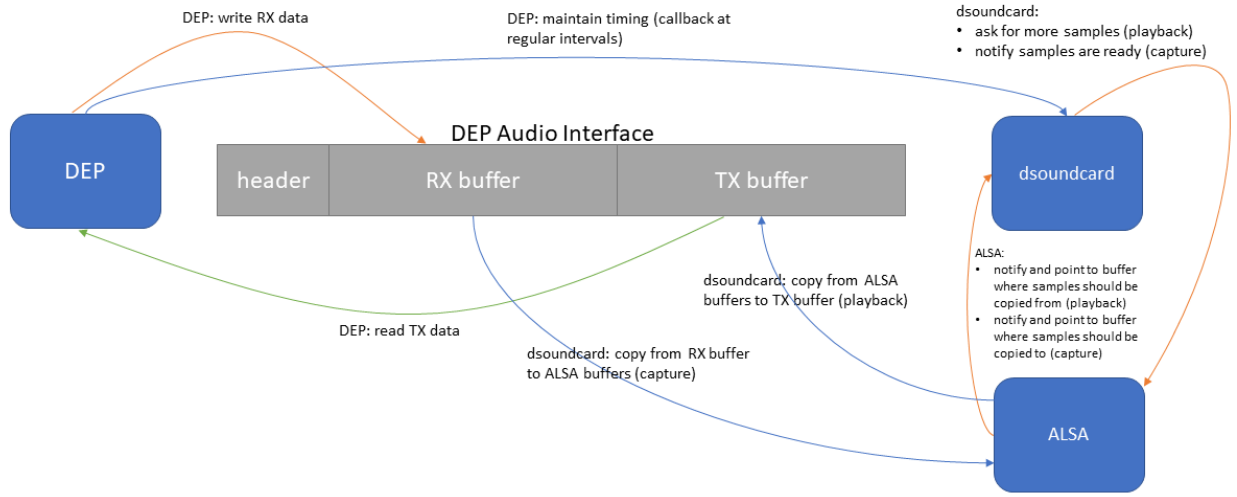


Figure 16 - dsoundcard architecture

The plugin illustrates the following concepts:

Interacting with a single ALSA device in playback or capture mode.

- Converting between the PCM formats and the DEP format.

The plugin can also be used as a PulseAudio input/output device. Please refer to the README file in the apps directory for details on how to accomplish this. To use DEP as a PulseAudio input and output simultaneously, you will have to execute the following steps:

1. Again, append the contents of `asoundrc` to the `~/.asoundrc` config file as with initial setup, but change all occurrences of 'dsoundcard' to a name of your choosing. (After this step you will need to reboot for the config changes to take place).
2. At the bottom of `alsa_plugin.cpp` file, change the two occurrences of symbol 'dsoundcard' to 'your_name'.
3. Recompile the soundcard and rename the output to '`your_name.so`'. Move this output to be in the same directory as '`dsoundcard.so`'.
4. Register the new soundcard plugin with PulseAudio by using the same commands as for dsoundcard, except replace each reference to dsoundcard with 'your_name'. Either plugin can act as either a sink or a source.

dtest

dtest is a software application that plays tones and pink noise into individual channels of the DEP output. The figure below shows the relationship between DEP, the DEP audio interface, the dtest application and the sample generator.

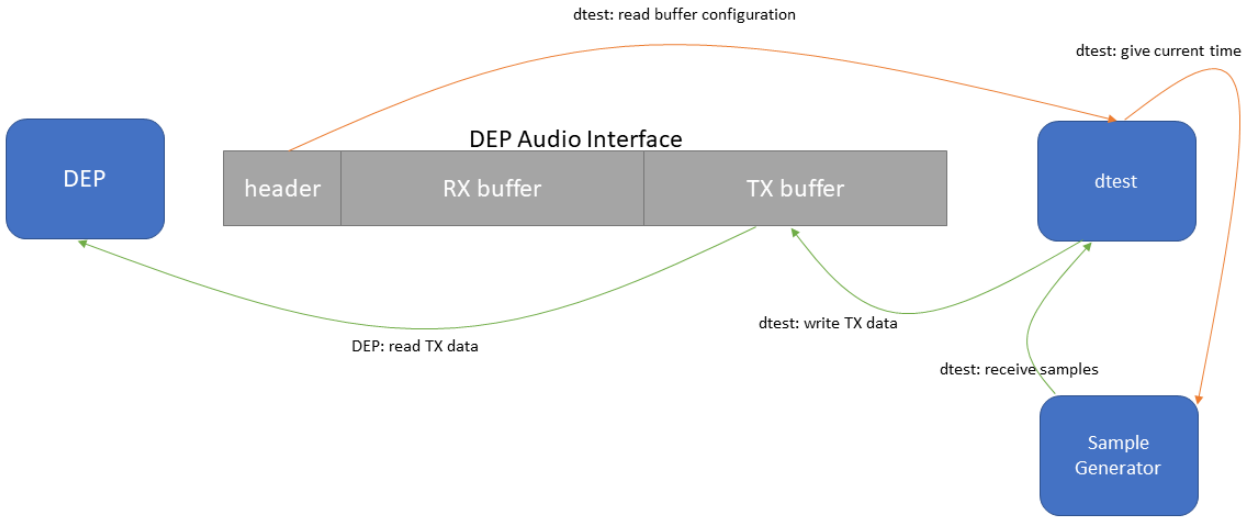


Figure 17 - dtest architecture

dtest illustrates many of the same concepts as the other applications. In addition, it also illustrates the following concepts:

- Playing audio samples into individual channels of DEP
- Keeping track of time for the purpose of generating a tone

ALSA ASRC

DEP is shipped with an optional Asynchronous Sample-Rate Converter (ASRC) which bridges audio data bidirectionally between the DEP Audio Interface and one or more ALSA devices. ASRC performs rate tracking on

the PTP time provided by DEP relative to the ALSA devices and is offered as an alternative to Hardware Clock synchronisation.

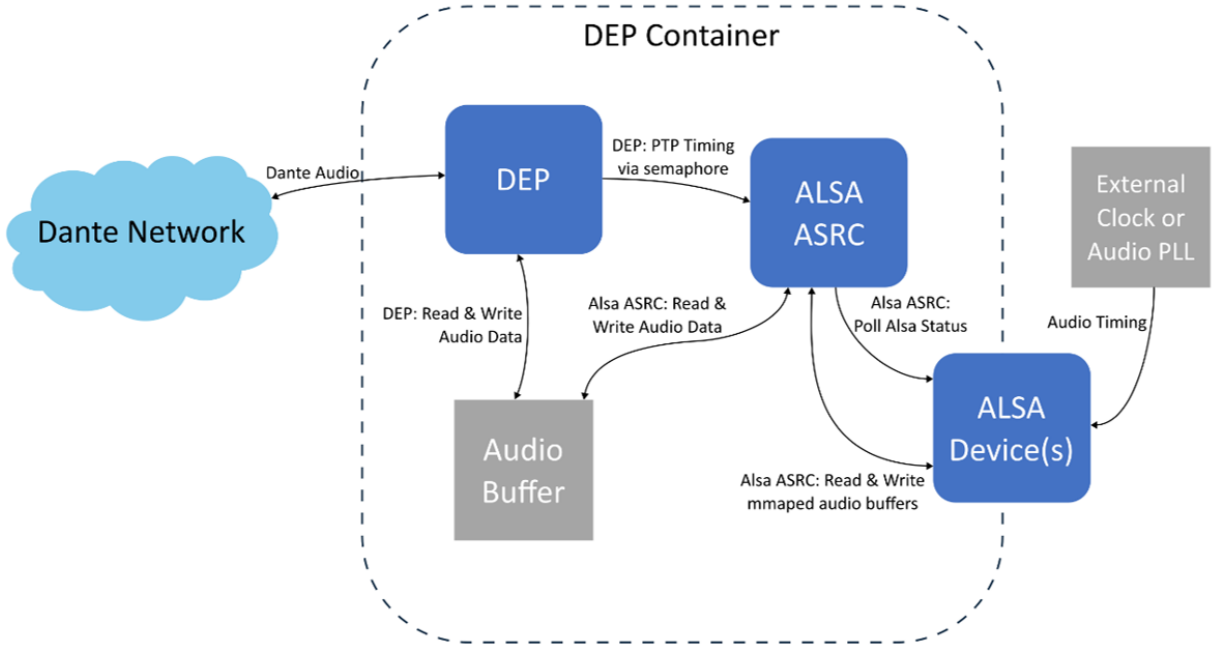


Figure 18 - ALSA ASRC architecture alongside DEP

ALSA ASRC accounts for small differences in PTP timing and audio device timing, but does not support macro sample-rate conversion, for example 44100Hz to 48000Hz. Instead, ASRC will open the ALSA devices at the same sample-rate as DEP, updating this rate if DEP's sample-rate is changed.

Running ALSA ASRC

ALSA ASRC runs inside the container and is managed by DEP. However, the exact configuration is dependent on the ALSA devices present on the target system. To run ALSA ASRC:

- The container configuration config.json should be adjusted to map any necessarythe ALSA devices into the container, along with the ALSA configuration file.
- Enumerate playback and capture audio device configurations in the dante.json file. This includes which Dante channels these devices should map to.

Once configured, when DEP next restarts it will initialise ALSA ASRC, which in turn opens the relevant ALSA devices and begins streaming audio data.

See the ALSA ASRC Integration Guide for detailed instructions on setting up and integrating ASRC on your platform.

Configuring the Container for ALSA

To use ALSA ASRC the necessary ALSA devices and configuration must be visible from within the container. You must provide the ALSA configuration file `alsa.conf` that is appropriate for your platform. This configuration file can then be bind mounted into the container at the path `/usr/share/alsa/alsa.conf` to be used by ALSA ASRC. For example, if your ALSA configuration file is present at `/path/to/alsa.conf` in your device, include the following in the `"mounts"` list of the config.json:

```
"mounts": [
         {
            "destination": "/usr/share/alsa/alsa.conf",
            "type": "bind",
            "source": "/path/to/alsa.conf",
            "options": [
                "bind",
                "ro"
            ]
        },
        ...
```

Furthermore, ALSA devices to be used need to be mapped into the container. To enable ALSA, add a device to the "linux : resources : devices" with the major device number 116. By not specifying the minor number, the container will be able to use any ALSA device. The below is an example demonstrating how to enable ALSA in the container:

```
"linux": {
        "resources": {
            "devices": [
                {
                    "allow": true,
                    "type": c,
                    "major": 116,
                    "access": "rw"
                },
                ...
```

Finally, the ALSA devices should be mounted into the container. This can be done by binding `/dev/snd` into the container in the `"mounts"` list. Note that this mount must be made after the item which mounts `/dev` into the container. See the below extract from a `config.json` file demonstrating mounting `/dev/snd` into the container:

```
"mounts": [
        {
            "destination": "/dev",
            "type": "tmpfs",
            "source": "tmpfs",
            "options": [
                "nosuid",
                "strictatime",
                "mode=755",
                "size=65536k"
            ]
        },
        {
            "destination": "/dev/snd",
            "type": "bind",
            "source": "/dev/snd",
            "options": [
                "bind",
                "rw"
            ]
        }
```

As an alternative, individual ALSA devices from `/dev/snd` can be mounted into the container in the `"linux : devices"` section, along with the corresponding control device(s) `/dev/snd/controlCX`.

Testing ALSA in the Container

Packaged in the container is a testing utility named `atest`. This utility can be used to test and interrogate ALSA devices from within the container. While the DEP container is running, `atest` can be run using `crun`, which is included in the DEP package at `/dep/dante_package/crun`. If DEP was started with super user permissions, `crun` must also be run with the same permissions. From the directory containing `crun`, run `atest` using the command:

```
sudo ./crun exec dante /dante/atest [args]
```

See the ALSA ASRC Integration Guide for complete usage, examples and procedures for using `atest`.

List ALSA Devices

`atest` can be used to list the ALSA devices visible in the container. To do so, use the `-L` argument. The following is an example partial output using this option:

```
# sudo ./crun exec dante /dante/atest -L
null                              Discard all samples (playback) or generate zero samples
(capture), , In+Out
Card hw:0 (hw:CARD=Loopback)
        Card 0, Name 'Loopback'
        Component: ''
        Driver: 'Loopback'
...
```

To see the full list of devices, ensure that the following options are “on” in your `alsa.conf` file:

```
defaults.namehint.showall on
defaults.namehint.extended on
```

Testing Device Audio

`atest` can play test tones out through playback devices, and displays VU meters for capture devices. Complete usage instructions for `atest` can be found Appendix 5: ALSA ASRC Configuration.

As an example, the DEP-EVK-IMX8, the following command opens ALSA device hw:1,0 in playback and capture mode, with bit-depth 32, sample-rate 96000, a buffer size of 384 samples and 4 channels in each direction, with a 440Hz sine wave playing over the playback channels using the following command:

```
/dep/dante_package/crun exec dante /dante/atest -d 'hw:1,0 play b32 R96000 B384 c4' -d 'hw:1,0 capt
b32 R96000 B384 c4' -D -S 440
```

Configuring ALSA ASRC in DEP

When enabled and configured via the `dante.json` file, DEP will start and manage the ALSA ASRC process. ALSA ASRC is enabled via the `alsaAsrc : enableAlsaAsrc` option, and must be configured with at least one device in the "deviceConfigurations" list.

Provided ALSA ASRC is correctly configured, ASRC will take a few seconds to initialise and the configured ALSA devices will play/capture audio to/from DEP. ALSA devices opened in playback mode will play Dante RX audio DEP receives, and capture device audio will be sent over DEP’s Dante TX subscriptions.

Latency

ALSA ASRC adds several sources of configurable and non-configurable latency.

- Similarly to the example applications, when writing to DEP's TX buffers, ASRC offsets the writes by a fixed latency. This is configurable using the `"txLatencySamples"` option.
- ALSA creates a configurable buffer per device of audio captured or to be played. ASRC aims to keep this buffer filled to the halfway point, adding a latency of half the buffer size. The size of the buffer is configured with the `"bufferSize"` option in each device configuration. If desired, the target point within the buffer can be adjusted by setting the `"latency"` option in the device configuration also.
- The ALSA drivers for your devices will add latency also from various sources, such as an internal buffer for USB bus transfers. This is not accounted for in the above settings and is not configurable or measured from within ALSA ASRC.

Real Time Considerations

ALSA ASRC is highly sensitive to scheduling and performance. By default, ASRC waits on and consumes the DEP semaphore, and runs at a high real time thread priority. It is recommended to allocate an exclusive core for ASRC.

- If it is necessary for the DEP semaphore to be available for another application to use, it can be disabled by turning on the `"pollMode"` setting.
- By setting the `"cpuAffinity"` setting, ASRC will set its core affinity separately from the cores specified by `"numDepCores"`. When combined with core isolation this allows ASRC to use an exclusive core.
- By default, DEP starts ASRC at real time thread priority 70. This can be configured using the `"schedulingPriority"` setting. It is recommended that this be set to 100 once the system has been designed and is stable.

ALSA Device Configuration

ALSA ASRC can open multiple ALSA devices in playback and/or capture mode, with most of the same options as in `atest` device configuration, see the appendix for a complete list of options.

The primary configuration options are setting the ALSA buffer size, as described in the Latency section, the audio format, and mapping ALSA channels to DEP channels. The audio format can be selected either using the `"bitDepth"` or `"alsaFormat"` options. Similarly to `atest`, setting a bit depth will select the first Alsa format with that bit depth supported by the device.

ALSA ASRC does not support sample-rate conversion beyond small timing differences between PTP time and audio time. ASRC will open each ALSA device at DEP's Dante sample-rate. When DEP's sample-rate changes, ALSA ASRC will re-open all the configured ALSA devices at the new sample-rate.

Each capture device maps to a block of DEP TX channels and each playback device maps to a block of RX channels. Channels are mapped from ALSA to DEP via the options `"numOpenChannels"`, `"alsaChannelRange"` and `"danteChannelRange"`. In combination, an ALSA device can be opened with any number of channels, and then a specific block of those channels are mapped to a equivalently sized block of DEP channels. For example, to open the device `hw:0,0` capture device with 8 channels, and have the audio from channels 4, 5, 6, and 7 send audio to DEP channels 16, 17, 18 and 19, the following should be in the ALSA configuration:

```
"alsaAsrc": {
        "enableAlsaAsrc": true,
        "deviceConfigurations": [
            {
                "deviceIdentifier": "hw:0,0",
                "direction": "capture",
```

```
                "numOpenChannels": 8,
                "alsaChannelRange": 4-7,
                "danteChannelRange": 16-19
            }
        ]
}
```

By default, ALSA ASRC will map all the ALSA channels opened to the first `"numOpenChannels"` DEP channels.

DEP On-Device Monitoring and Control

Overview

There are currently two SDKs that can be used to write applications that run on the same processor as DEP to monitor and control DEP: Embedded Dante Application Programming Interface (eDAPI) and Dante Device Protocol (DDP).

Both are available as separate packages to DEP. They are not specific to DEP, as there are other Dante devices that support one or both of those programming interfaces. This section only gives a brief overview of the two SDKs and highlights specific points that are of particular note in the context of DEP. For complete documentation of eDAPI please consult the Dante API Programmer's Guide, the Dante API SDK Installation Guide (Linux) and the examples included in the eDAPI package. For DDP please consult the Host CPU SDK Programmer's Guide" and the examples included in the Host CPU SDK package.

The table provides contrasting points between eDAPI and DDP. In deciding which to use, the key items are likely to be the preferred programming interface mode, the available APIs and possibly any existing applications that have been written for other Dante devices that could be reused. In general, DDP has a simpler programming interface, and eDAPI has a richer set of APIs. For the exact set of supported APIs, please consult the relevant programming manuals as references above.

	eDAPI	DDP
Programming interface mode	Asynchronous	Synchronous
Monitor/Control reach	Local and remote Dante devices	Local Dante device only
Dante device browsing/discovery	Supported	Not supported
Dante Conmon monitoring/control	High coverage	Subset coverage
Dante routing control/status	Local and remote Dante devices	Local Dante device only

Host System Setup

Common

As described throughout this document, DEP runs within a container. The recommended configuration is to run any eDAPI/DDP application outside the DEP container.

eDAPI/DDP and DEP communicate via local Unix sockets. eDAPI also uses some local IP ports. Thus, it is critical to the functioning of eDAPI /DDP that these communication channels be accessible both within the DEP container and outside the container in the native host environment.

Specifically, DEP and eDAPI/DDP will create Unix sockets within the `/var/run/dante` directory, and thus this host directory needs to be bind-mounted into the DEP container. The example DEP `config.json` files show how that can be configured.

The relevant snippet of `config.json` is shown below. It shows how to mount `/var/run` as a tmpfs inside the DEP container and overlay a bind mount of the host `/var/run/dante` directory into the container.

```
"mounts": [
                {
                        "destination": "/var/run",
                        "type": "tmpfs",
                        "source": "tmpfs",
                        "options": [
                                "nosuid",
                                "strictatime",
                                "mode=755",
                                "size=65536k"
                        ]
                },
                {
                        "destination": "/var/run/dante",
                        "type": "bind",
                        "source": "/var/run/dante",
                        "options": [
                                "bind",
                                "rw"
                        ]
                },
```

eDAPI

eDAPI is always enabled in DEP and no further on device host configuration is required.

DDP

DDP is disabled by default in DEP. DDP can be enabled by setting the following values in the DEP `dante.json` configuration file.

```
"hostcpu": {
    "enableDdp": true
},
```

Dante Device-Host Interface

Dante Device-Host Interface (DDHI) is a protocol for communication between Dante stacks and their local host environment. It allows the Dante stack to interact with the host to perform operations that require host side support. Examples of this are device reboot, device IP interface configuration and device upgrade. DEP supports DDHI as of version 1.5.0.

The following table lists the DDHI features supported by DEP and the DEP version that support was first added.

Feature	First DEP Version
reboot	1.5.0
network configuration (static/dynamic IP)	1.5.0

To fully support a DDHI feature an OEM needs to perform the following integration steps.

1. Enable DDHI support in DEP. This is the default state, so no extra configuration is needed, but for completeness the `dante.json` field for this is `"ddhi/enable"`.
2. List the DDHI RPCs for each supported feature in the `"ddhi/clientRpcs"` list. The RPCs for each feature are listed in the “Dante Device-Host Interface SDK User Manual”.
3. Write a DDHI client application that runs on the host to handle the list of client RPC messages for each supported DDHI feature. The “Dante Device-Host Interface SDK” source code package and User Manual are the required resources for writing a complete DDHI client application. The SDK also includes an example application which can either be used as a reference or starting point for writing a customised DDHI client application. The sample application handles all DDHI request messages and then invokes a shell script to handle each request. If the sample application is used then this step is simplified to include just running the example `"ddhi_service"` and implementing the shell script for each of the supported features (e.g. `"reboot.sh"` and `"network.sh"` for the reboot and network configuration feature respectively). Consult the DDHI Client SDK User Manual for a description of the how each script is called for each feature and what the script needs to do.

The following is an example walkthrough of how to enable and test DDHI based device reboot.

1. Update dante.json to include the following “ddhi” configuration listing the RPC needed for reboot.

```
"ddhi": {
    "clientRpcs": [ "ddhi::reboot" ]
}
```

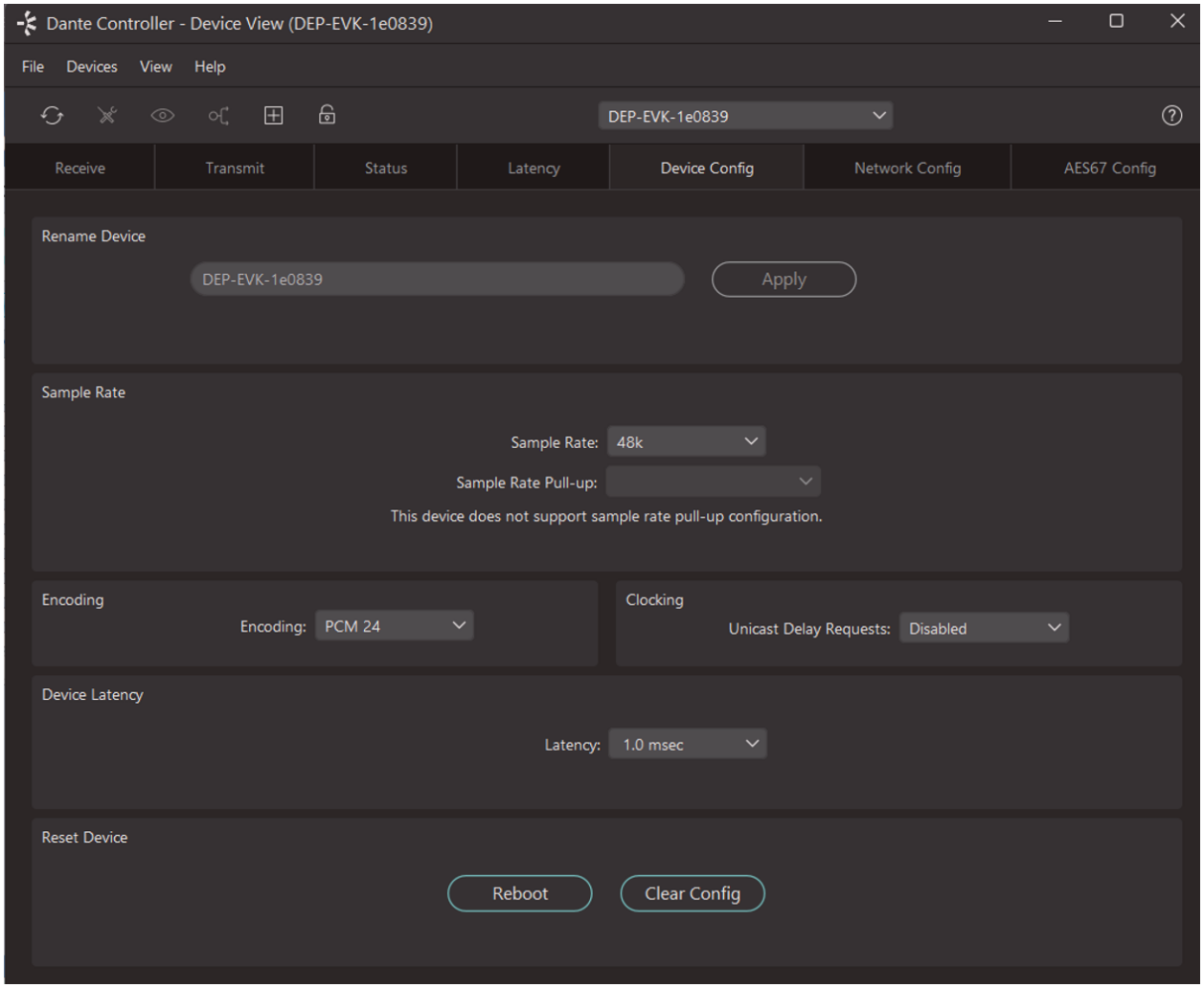
2. Download and build the “Dante Device-Host Interface SDK”. The example application binary will be built as `"ddhi_service"`.
3. 1. Install `"ddhi_service"` onto the target and configure the system (e.g. via system or sysvinit) to run the application on start up. An example command line is shown below. The `"–transport"` argument is required, as that is the DEP DDHI server connection path. The `--auto-reconnect` argument is recommended as it will re-connect to the DEP DDHI server if the connection is lost (e.g. as a result of a DEP restart).

```
ddhi_service --transport=ddhi+unix+seqpacket:/var/run/dante/ddhi --auto-reconnect
```

4. Implement a "`reboot.sh`" which `ddhi_service` will invoke whenever a reboot is requested via Dante (e.g. via Audinate Dante Controller or any other third-party controller). Place the script in the same directory as `ddhi_service`. This is an example reboot.sh which is applicable to most Linux systems:

```
#!bin/sh
Reboot
```

5. Start or re-start DEP.
6. To test, start up Audinate Dante Controller and open the "Device Config" tab in the "Device View" for the device. If the integration is successful, then the "Reboot" button will now be enabled and when clicked will reboot the device.



Appendices

Appendix 1: Kernel and Platform Configuration

Enable Cgroup Support

The following Linux kernel config options must be enabled to support cgroups. Additional cgroup config options may be enabled as required.

- CONFIG_CGROUPS
- CONFIG_CGROUP_DEVICE
- CONFIG_CPUSETS
- CONFIG_CGROUP_SCHED
- CONFIG_MEMCG
- CONFIG_CGROUP_FREEZER

Cgroups Versions

Note: This sub-section is only relevant for DEP versions prior to 1.5.0. From DEP 1.5.0 onward Cgroups V2 is supported.

There are two versions of Cgroups - v1 and v2. While DEP prior to 1.5.0 only supports Cgroups v1, some Linux distributions may use Cgroups v2 by default. If that is the case, then DEP will fail to start unless `numDepCores` in `dante.json` is set to `0` to disable DEP Cgroups configuration.

In general, it is not recommended to disable DEP Cgroups as that may lead to audio quality issues. So that should only be done as a short-term workaround.

One of the main differences between Cgroups v1 and v2 are the mount points. As described above, Cgroups v1 uses per controller mounts. In contrast, Cgroups v2 uses a single unified mount point.

One way to tell whether Cgroups v2 is active on a given Linux system is to run the mount command and check which mounts are in place. If there are any mounts with a “cgroup2” filesystem type, that indicates Cgroups v2 is active and needs to be disabled before starting DEP.

Example output showing a system with active Cgroups v2 configuration:

```
# mount | grep cgroup2
cgroup2 on /sys/fs/cgroup type cgroup2 (rw,no-
suid,nodev,noexec,relatime,seclabel,nsdelegate)
```

The exact procedure for disabling Cgroups v2 is system/distribution specific so it may be necessary to consult relevant system documentation. For Debian and Ubuntu systems that use systemd as init system, the following procedure has been found to be effective for disabling Cgroups v2:

1. Edit /etc/default/grub to add the following line:

   ```
   GRUB_CMDLINE_LINUX="systemd.unified_cgroup_hierarchy=0 systemd.legacy_systemd_
   cgroup_controller=0"
   ```

2. Run `update-grub`
3. Reboot

In addition to meeting the above requirements, at least the following cgroup subsystems must be mounted at `/sys/fs/cgroup/`. An example mount output snippet is shown below.

```
cgroup on /sys/fs/cgroup type tmpfs (rw,relatime,mode=755)
cgroup on /sys/fs/cgroup/cpuset type cgroup (rw,relatime,cpuset)
cgroup on /sys/fs/cgroup/cpu type cgroup (rw,relatime,cpu)
cgroup on /sys/fs/cgroup/memory type cgroup (rw,relatime,memory)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,relatime,devices)
```

This can be done by enabling a package such as cgroupfs-mount (https://github.com/tianon/cgroupfs-mount).

Disable Fine-Grained Real-Time Scheduling

DEP doesn't support fine-grained real-time scheduling per cgroup yet, so the following Linux kernel configuration should be disabled:

- CONFIG_RT_GROUP_SCHED

Enable Namespace Support

The following Linux kernel config options must be enabled to support Linux namespaces.

- CONFIG_NAMESPACES
- CONFIG_UTS_NS
- CONFIG_IPC_NS
- CONFIG_USER_NS
- CONFIG_PID_NS
- CONFIG_NET_NS

No further action is needed to enable Linux namespaces on system start-up.

Enable SquashFS Support

The following Linux kernel config options must be enabled to support SquashFS in Linux. Additional config options related to SquashFS may be enabled as required.

- CONFIG_SQUASHFS
- CONFIG_SQUASHFS_ZLIB

Enable Loop Device Support

The following Linux kernel config options must be enabled to support mounting of SquashFS files.

- CONFIG_BLK_DEV_LOOP

Low Latency Configuration

To get consistent performance from a platform running DEP, we recommend that you configure the following options in the kernel.

CPU frequency scaling options

CPU frequency scaling should be disabled, either by not setting the option below at all or setting it to no:

```
CONFIG_CPU_FREQ
```

Or, if this option is set to yes, the following options should also be set:

```
CONFIG_CPU_FREQ_GOV_PERFORMANCE = yes
CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE = yes
```

Setting the default scaling governor as above is ideal but not strictly required, as the default can later be changed via sysfs.

Enable real time optimizations by enabling the following kernel config option

```
CONFIG_PREEMPT = yes
```

If you are using a standard x86_64 distribution, it is recommended that you use a minimal server version rather than a desktop version, to minimize the possibility of interfering drivers and processes.

Network Configuration

The following network related kernel config option is needed to enable DEP function properly:

```
CONFIG_IP_MULTICAST
```

Hardware RNG

Hardware RNG must be present and enabled on the board.

Using DEP-EVK-IMX8 as an example, hardware RNG is enabled with following kernel config option:

```
CONFIG_HW_RANDOM
CONFIG_CRYPTO_HW
CONFIG_CRYPTO_DEV_FSL_CAAM
```

`rngd daemon` from the `rng-tools` package must be started before starting DEP.

i.MX EVK Network Driver Configuration

There are several network issues identified in the 4.14.98 kernel used for DEP-EVK-IMX8 requiring the application of the following patches.

Freescale Ethernet Driver Checksum Issue

There is a silicon bug in the i.MX Ethernet that drops packets less than the minimum size when protocol layer hardware checksums are enabled. One way to fix the issue is to disable protocol layer hardware checksums while retaining IP layer hardware checksums. The patch of the change is available at:

```
DEP-EVK-IMX8.sources.tgz/DEP-EVK-IMX8/board/audinate/dep-evk-pico-
imx8mm/patches/linux/0001-fec-chksum-offload.patch
```

Alternatively, add the following command to your init scripts to disable receiver hardware checksums completely.

```
ethtool -K eth0 rx off
```

Note: If you are configuring multiple ethtool options ensure that they are invoked in the following order:

1. `ethtool -K eth0 rx off`
2. `ethtool -C eth0 rx-usecs 1`
3. `ethtool -C eth0 tx-usecs 1`

Freescale Ethernet Driver Coalesce Parameter Reset Issue

Network coalesce parameters configured for low latency are reset by the Freescale network interface driver between link down and up. This consequently affects DEP's performance. The patch for fixing the issue is available at:

`DEP-EVK-IMX8.sources.tgz/DEP-EVK-IMX8/board/audinate/dep-evk-pico-imx8mm/patches/linux/0002-fec-coalesce-reset.patch`

DSA Duplicate Multicast Data Issue

When daisy chaining, the multicast packets from daisy chained device are duplicated on the output port of the switch. The patch for fixing the issue is available at:

`DEP-EVK-IMX8.sources.tgz/DEP-EVK-IMX8/board/audinate/dep-evk-pico-imx8mm/patches/linux/0300-dsa-offload-forward-mark.patch`

Daisy Chaining on Network Without IGMP Querier

When daisy chaining, if an IGMP querier isn't present in the network, the multicast group database on the built-in switch may not be correctly updated. To work with this kind of network, switch ports on DEP-EVK-IMX8 are configured to be in multicast flooding mode and IGMP snooping is also turned off. The patch for this change is available at:

`DEP-EVK-IMX8.sources.tgz/DEP-EVK-IMX8/board/audinate/dep-evk-pico-imx8mm/patches/linux/ 0500-dsa-switch-port-multicast-flood.patch`

Dante AV-H

The following kernel parameters are needed for Dante AV-H products. These enable the container to apply traffic shaping to the AV-H video flow to improve performance, and analyse the video flow to report the status.

These kernel settings are required from v1.0.7 of the AV-H container and need to be enabled on AV-H encoders and decoders. If these kernel settings are not enabled, it will not be possible to subscribe the AV-H video channels.

```
CONFIG_NET_SCHED
CONFIG_NET_SCH_HTB
CONFIG_NET_SCH_FIFO
CONFIG_NET_CLS_U32
CONFIG_NET_CLS_ACT
CONFIG_PACKET
CONFIG_NETFILTER
```

Appendix 2: Container Configuration

DEP runs within a standards compliant OCI container. OCI makes use of a container configuration file to specify parameters for running a container. DEP expects the container configuration file to be located at:

`dante_package/dante_data/capability/config.json`

The syntax and format of the container configuration file are described as part of the OCI run time specification, which can be found at this URL: https://github.com/opencontainers/runtime-spec/blob/master/config.md.

See Example Applications and Configuration for a description of how to make use of the example container configuration files provided as part of the DEP examples release package. The following sections highlight key settings in the container configuration which are relevant to the correct functioning of DEP. Some sections may need customisations to reflect specific DEP platform installation properties, and those are pointed out below where relevant.

hooks

This section specifies custom actions that are run at certain points in the container lifecycle. The `depconfig` command included in the DEP container release needs to be run at the poststart and poststop stages of the container. The configuration files in the examples package provide the path and args needed to run `depconfig`. The path value may need to be updated to reflect the actual DEP installation location.

Selecting the CPU cores used by DEP

For DEP Versions Prior to 1.5.0

When DEP is started, `depconfig` creates two cgroups for the container:

- A group for DEP's processes
- A group for all other tasks

By default:

- The DEP cgroup is assigned a contiguous set of cores on the system starting from core 0. The number of cores assigned is `numDepCores` (specified in `dante.json`). For example, a `numDepCores` value of 2 will result in cores 0 and 1 being assigned to the DEP cgroup.
- The other cgroup is either assigned the remaining system cores, or all of them, depending on whether DEP is using its cores exclusively (as specified in `dante.json`)

If desired, different sets of cores for either or both of these cgroups can be allocated by supplying additional arguments to `depconfig`. These are placed immediately after the "`depconfig`" command in the "`poststart`" section and have the following format:

- `"--dep-cpu-cores", "<cores>"` for the DEP cgroup
- `"--other-cpu-cores", "<cores>"` for the other cgroup

`<cores>` consists of comma separated values, where:

- Each value can be a single core or a range of contiguous cores specified using a dash, e.g.1-3
- The order of the range specifiers does not matter, and so the aforementioned example could be written as 3-1

When only one set of cores is supplied:

- If only `--dep-cpu-cores` is specified, the other cgroup will default to the remaining cores or all cores, depending on exclusivity
- Otherwise if only `--other-cpu-cores` is specified, the DEP cgroup will default to the first `numDepCores` cores starting from 0

Examples

In a system with five or more cores, to configure DEP to use cores 1 and 3 and place other container tasks onto core 4:

```
"args": ["depconfig", "--dep-cpu-cores", "1,3", "--other-cpu-cores", "4", "setup", …
]
```

To set DEP to use cores 2, 3 and 4 and use default assignment for the other tasks:

```
"args": ["depconfig", "--dep-cpu-cores", "2-4", "setup", … ]
```

Error conditions

DEP will fail to start if any of the following are true:

- The number of unique cores in `dep-cpu-cores` is less than `numDepCores`
- With unavailable (e.g. offline, already in use by a service) cores excluded, the number of cores earmarked for DEP is less than `numDepCores`
- There are no cores available for the other cgroup because they are all either offline or being used exclusively by DEP

Any cgroup configuration errors or issues can be picked up by running the `dep_check.sh` script.

For DEP Version 1.5.0 and Newer

When Cgroups V2 is enabled, `depconfig` is no longer run. Instead the following configuration replaces what `depconfig` previously attempted to do:

1. Core isolation: The platform should be configured to isolate the cores that DEP will run on. See the "Achieving CPU Isolation" chapter for guidance on how to achieve core isolation at the platform level.
2. CPU affinity: DEP will set the CPU affinity of the DEP processes based on `"numDepCores"` field in `dante.json`. If `"numDepCores"` is a single value, say N, then DEP will set its process affinities using cores 0 to N-1, If `"numDepCores"` is a list of values then DEP will use that as the exact cores for its process affinities.

The cores configured for isolation and the effective cores for affinity in `"numDepCores"` must match and it is the OEM's responsibility to ensure that is the case.

process

This section specifies what process to run when the container is started as well as various parameters of the process. For DEP, the `dep_manager` is the process to be started and the other parameters are provided in the example configuration file.

mounts

This section specifies the mounts of the host file system into the container file system. For DEP there are several important mounts. The target mount points already exist in the read only root filesystem that is the base mount inside the container.

Host/Source mount point	Container/target mount point	Description
`$DEP_INSTALL_DIR/dep/dante_package/dante_data`	`/dante_data`	The writeable directory tree where DEP will store all persistent data, such as user configuration and activation data
`/tmp`	`/tmp`	Temporary directory
`/var/log`	`/var/log`	Host systems that have a separate (non-symlink) log directory can create this mount

Host/Source mount point	Container/target mount point	Description
`/dev/shm`	`/dev/shm`	Shared memory device. Needed by the DEP Audio Interface
`/var/run/dante`	`/var/run/dante`	Dante creates local Unix sockets in this directory. It needs to be visible to both the host and to DEP running in the container if embedded DAPI is to be used.
`/dev/snd`	`/dev/snd`	ALSA sound devices. Necessary if using ALSA ASRC.
Device's `alsa.conf` file	`/usr/share/alsa/alsa.conf`	ALSA configuration file. Necessary if using ALSA ASRC.

linux : devices

This section specifies the host devices that are made available inside the container. It should be modified to correspond to the platform that DEP is installed on. Of note:

- If hardware timestamping is enabled, `/dev/ptpX` must be provided and correspond to the correct PTP device on the host system. For systems with secondary Dante network interfaces, both corresponding PTP devices must be provided. The presence of hardware timestamping support on an interface and *PTP Hardware Clock index* (X number in ptpX) can be determined by running ethtool with the name of the interface - for example: `ethtool -T eno2`
- I2C buses and hardware clock feedback devices (`extclkin`) should be added if DEP's hardware audio clocking feature is to be available and used for the platform

linux : resources : devices

This section specifies cgroup device limits/permissions. The list of devices in this section should include all the devices in the `"linux : devices"` section described above.

Appendix 3: Dante Configuration

The DEP configuration file is in the JSON format. DEP expects the file to be located at:

```
dante_package/dante_data/capability/dante.json
```

For ease of configuration, the configuration file is broken up into several sections, as follows. The order of the sections and the order of the fields in each section are not significant. That is, the sections and fields in each section can be listed in any order.

Platform

This section specifies general platform behaviours.

Logging

DEP will produce logs as it runs. There are separate log files for each DEP process.

`logDirectory` represents a path inside the container filesystem, and must specify a writable directory. The base container root filesystem is mounted as read-only - however the container `config.json` file specifies read-write bind mounts that overlay this base with writable directories on the host system.

Thus, `logDirectory` must be set to either:

- A read-write bind mounted directory, or
- A file which is symbolically linked to such a directory

The default `dante.json` has `logDirectory` set to `/var/log`, which depending on the host system will either be directly bind mounted or linked to a bind mount.

DEP provides a few mechanisms for balancing the disk space used by logs against the need to retain as much logging as possible for troubleshooting purposes.

1. The maximum size of each log file can be set by the `maxLogSize` field which defaults to `102400` bytes.
2. Once a log file reaches its maximum size it will be rotated. Rotation means that the current log file will be copied to a file of the same name but with an extra "`.X`" suffix where "X" is an incrementing number starting from 1. The maximum number of log files per process is controlled by the `maxNumLogs` field which defaults to 6.
3. Log verbosity can be set via the `logLevel` field. Higher log levels will generate more logs and thus fill up the log files faster. The valid log levels in increasing verbosity are: Error, Warning, Notice, Info, Debug. The default value is Warning.

Audio Options

This section allows you to configure audio parameters such as latency and number of channels.

Network Settings

This section allows you to specify network interface(s), interface mode and preferred link speed for DEP.

Note: Only wired Ethernet interfaces are supported for Dante audio - i.e. wireless and virtual interfaces cannot be used.

Note: The Primary and Secondary (when in redundant mode) network link speeds as shown in Dante Controller are obtained from the physical network interfaces specified in the `network.interfaces` JSON configuration, or their member interfaces in the case of bridge interfaces. If the interface speed is not obtainable from the specified interface, DEP will show a speed of "Unavailable" in Dante Controller. During development of your device you can check if the specified interface speed will be reported correctly by running `ethtool` on the specified interface on your device. If no speed is reported by the `ethtool` program then DEP will also be unlikely to be able to report it to Dante Controller.

For more information about network interface speeds please reach out to Audinate Support.

Clock

This section is used for configuration of the PTP clocking system.

Hardware Clock

This section is used for configuration of the hardware clocking subsystem, see Clocking for more information. If not present, the hardware clocking feature is disabled and left unconfigured.

hostcpu

This section configures values related to the Host CPU interface. Currently this is only used for enabling/disabling Dante Device Protocol (DDP).

ALSA ASRC

This section configures the optional ALSA ASRC application.

Product Information

This section configures how your product identifies itself to the Dante network. You are required to set both the manufacturer ID (as provided by Audinate) and model information of the device. This information is presented to the user in Dante Controller.

Miscellaneous

This section configures miscellaneous parameters that don't belong to any of the other sections.

Example Configuration File

```
{
       "platform":
       {
           "logDirectory": "/tmp",
           "maxNumLogs": 6
       },
       "audio":
       {
           "txChannels": 2,
           "rxChannels": 2,
           "sampleRate": 96000,
           "availableSampleRates": [ 44100, 48000, 88200, 96000 ],
           "samplesPerPeriod": 16,
           "periodsPerBuffer": 3000,
           "networkLatencyMinMs": 1,
           "networkLatencyDefaultMs": 4,
           "numDepCores": [1, 2, 3],
           "percentCpuShare": 100,
           "silenceHeadDelayMs": 20,
           "supportedEncodings": ["PCM24", "PCM16", "PCM32"],
```

```
            "aes67Supported": false
        },
        "network":
        {
            "interfaceMode": "Switched",
            "interfaces": [ "lan3", "lan4" ],
            "preferredLinkSpeed": "LINK_SPEED_1G",
            "webSocketPort": 5000
        },
       "mdns"
        {
            "restrictInterfaces": true
        },
        "clock":
        {
             "enableHwTimestamping": true,
             "dsaTaggedPackets": true,
             "hardwareInterfaces": [ "eth0", "eth0" ]
        },
        "hardwareClock":
        {
            "useHwClock": true,
            "circuitName": "Si5351B with MCP47CVB02",
            "circuitRevision": 0,
            "i2cBus": "/dev/i2c-1",
            "i2cAddr": "0x62",
            "extClockInputDev": "/dev/extclkin",
            "bitClocks":
            [
                {
                    "sampleRate": 48000,
                    "tdmChannels": 16,
                    "bitDepth": 32
                },
                {
                    "tdmChannels": 8,
                    "bitDepth": 32
                }
            ]
        },
        "hostcpu":
        {
            "enableDdp": true
        },
        "product":
        {
            "manfId": "Audinate",
            "manfName": "Audinate Pty Ltd",
            "modelId": "DEPEVK1",
            "modelName": "DEP EVKi.MX8 Linux ARM64",
            "modelVersion":
            {
                "major": 1,
                "minor": 0,
                "bugfix": 0
            },
            "devicePrefix": "DEP-EVK"
        },
        "misc":
        {
            "enableIdentify": true
        },
        "trialMode": false
}
```

JSON Field Descriptions

The following bullet list describes all the possible fields in the `dante.json` file. Each indented sub-list indicates that those are child fields of the parent section. The section and field ordering are not significant, and the actual `dante.json` file does not need to exactly follow the ordering shown in the following list.

- `"platform"`
  - `"logDirectory"`: Directory to write DEP log files. The directory must be available inside the container, DEP must have read/write access to the directory and the directory must exist before the DEP container is started.
  - `"logLevel"`: This sets the logging verbosity for all logs. Higher log levels will generate more logs. The valid log levels in increasing verbosity are: `"Error"`, `"Warning"`, `"Notice"`, `"Info"`, `"Debug"`. The default value is `"Warning"`.
  - `"maxNumLogs"`: Maximum number of versions of each DEP log file to keep. Default is 6.
  - `"maxLogSize"`: Maximum number of bytes of each DEP log file. When a log file reaches the max size it will be rotated to a different file name, and the original log file will be restarted with zero size. The default value is 102400 (100K) bytes.
  - `"cgroupVersion"`: Tells DEP which version of cgroups the system is configured for. Setting this value allows DEP to optimise the system more appropriately. If this value is not set at all then DEP will still function but possibly in a less optimal way. Valid values are 1 and 2 with 1 being the default.
- `"audio"`: configuration of audio subsystem.
  - `"txChannels"`: The number of transmit channels (from the perspective of Dante Controller). This is a mandatory field.
  - `"rxChannels"`: The number of receive channels (from the perspective of Dante Controller). This is a mandatory field.
  - `"maxRxFlows"`: Maximum receive flows that DEP will allow. This field is optional and the default is max (2, (rxChannels+1)/2) if the field is not present or set to 0.
  - `"maxTxFlows"`: Maximum transmit flows that DEP will allow. This field is optional and the default is max(2, (txChannels+1)/2) if the field is not present or set to 0.
  - `"sampleRate"` : The sample rate. This field is optional and the sample rate will default to 48000 if the field is not present.
  - `"availableSampleRates"`: A list of sample rates that can be selected. The list must contain at least 48000.
  - `"samplesPerPeriod"` : The number of samples per channel between audio period events (ticks). This field is optional and the default value is 16.
  - `"periodsPerBuffer"`: The number of periods in the buffer. This field is optional and the default value is 3000.
  - `"networkLatencyMinMs"`: The minimum latency in milliseconds that this device can support. Valid values are 1,2,3,4,5,10. This field is optional and the default of 2ms will be used if the field is not present.
  - `"networkLatencyDefaultMs"`: Network latency in milliseconds, which is applied when a user clears the configuration. The field is optional and the default value is 4 if the field is not present. The range of the default latency value is between 1ms and 40ms (inclusive). If the value is out of this range, the DEP software will set it to 4ms. The default latency value also needs to be equal or larger than "`networkLatencyMinMs`". If the default latency is set lower than the minimum, DEP will report an error and will not start.
  - `"numDepCores"`: CPU affinity for DEP. This is a mandatory field. Notes:

    - If set to zero, CPU resource allocation and management is not performed by DEP and will be performed by the host platform. Note that DEP requires resource allocation to ensure good performance.

- Must be between 0 and the total number of cores.

- As of DEP 1.0.x, assigning greater than 3 cores to DEP will not scale performance linearly higher.

- As of DEP 1.5, assigning a list of CPU cores can be used for detailed CPU affinity in DEP. All unsigned integers specified in the list must correspond to existing CPU cores, which are 0-indexed.

- DEP should never have exclusive allocation of all the cores on a system.

- `"percentCpuShare"`: **DEPRECATED** – but still supported when upgrading to DEP 1.5+ on cgroups v1 systems. This option has been deprecated for the following reasons:

  - The original intention of `"percentCpuShare"` was to allow customers to run other tasks alongside DEP ones on the same CPU cores where Dante audio is being processed. Recent testing and development has proved that DEP tasks need to run without any kind of system interruptions for glitch-free audio to be guaranteed, regardless of CPU capabilities and performance. Sharing CPU time with other tasks goes the opposite direction.

  - Audinate recommends that customers run DEP on Real-Time kernels and tune their systems to fully isolate CPU cores dedicated to DEP processes: `cpu.shares` (the cgroups v1 parameter `percentCpuShare` translates to) and other resource distribution mechanisms that depend on scheduling patterns all become completely irrelevant on such setups, as fully isolated CPUs (eg via isolcpus) don't participate in CFS load balancing.

  - Customers who still want to experiment with CPU load balancing on DEP dedicated cores can still do so by following the cgroups v1 and v2 online manuals.

- `"defaultEncoding"`: **DEPRECATED** – see `supportedEncodings` for the new recommended way to set this value. For backwards compatibility this field can still be specified for now. The default encoding is what DEP will use on the very first startup, and after a clear of device configuration. Valid values are "PCM16", "PCM24" or "PCM32" as a string. If `supportedEncodings` is specified, it must contain this default value.

- `"supportedEncodings"`: a list of the encodings which can be selected in the Device Config Encoding drop-down list in Dante Controller. These are the encoding(s) applications accessing the DEP Audio interface can use natively, and thus prefer to send/receive in. Valid values are "PCM16", "PCM24" and "PCM32", as strings. The first value in the list will be used as the default encoding, unless the `defaultEncoding` field is set, in which case that is used as the default encoding value. This field is optional. If it is not set but `defaultEncoding` is, the latter becomes the sole supported encoding. If both are not set, "PCM24" will be the sole encoding.

- `"defaultChannelNamesFile"`: the full path to a separate JSON file (details in the Assigning Default Channel Names section below) that specifies custom default channel names. The file must be visible to the container running DEP, and so one place to put it is the same directory as `dante.json`. In this case, if the separate JSON file were named `defaultChannels.json` this field would be set to `/dante_data/capability/defaultChannels.json`. This field is optional.

- `"channelGroupsFile"`: the full path to a separate JSON file (details in the Creating Channel Groups section below) that specifies logical channel groupings. These groups will appear in Dante Controller. As with the default channel names file, this JSON file needs to be visible to the container. This field is optional.

- `"perChannelEncodingsFile"`: the full path to a separate JSON file (details in the Specifying Per-Channel Encodings section below) that specifies native encodings at the channel level. If this field is present, the supportedEncodings and defaultEncoding fields are unused as device-wide encodings are no longer applicable. As with the preceding two fields, this JSON file needs to be visible to the container. This field is optional.

- `"aes67Supported"`: whether this device supports AES67. If this is set to "true" the AES67 tab will appear in DC when looking at this device, allowing end-users to turn the feature on or off. This field is optional, and by default AES67 is not supported. In a production device, this capability should not be changed at runtime or across restarts of DEP, as it may result in inconsistent device state. During development this capability can be changed, but the device configuration should be cleared after such a

change (e.g. via the Dante Controller device “Clear Config” operation) to ensure the device state remains consistent.

- `"enableSelfSubscription"`: whether this device supports self-subscriptions. If this is set to “true”, it will be possible to subscribe this device to receive audio from its own TX channels. This field is optional and set to “true” by default.
- `"silenceHeadDelayMs"`: the delay after which DEP erases audio in the RX and TX buffers, measured in milliseconds. Increasing this value can allow systems with more scheduling variance to process audio reliably. Defaults to 20ms.

- `"network"`: network configuration
  - `"interfaceMode"`: String value of either “`Switched`” or “`Direct`”. Optional, will default to “`Direct`”. Direct - means connected to the network via a PHY, Switched means connected to the network via a switch. When the interface is a switch, the switch/interfaces can be configured to be either in “switched” (daisy chain) or “redundant” mode for DEP. For details, refer to 'DEP Device Configurations' in Example Applications and Configuration, and Networking.
  - `"interfaces"`: List with names or indexes of the network interfaces your product will use to connect to the Dante network, e.g. `["eth0"]`, `["eno1"]`, `["eth0","eth1"]`, etc. The first entry in the list is used as the primary interface and the second entry, if present, is used as the secondary interface. This field is mandatory. NOTE: Only wired Ethernet interfaces are supported for Dante audio. i.e. wireless and virtual (virtual machine) interfaces cannot be used.
  - `"preferredLinkSpeed"`: The preferred link speed of the “interfaces” used by DEP. It is normally decided by the speed of the underlying Network Interface Card (NIC). Valid values are “LINK_SPEED_100M”, “LINK_SPEED_1G”, “LINK_SPEED_10G” as strings. This field is optional and will default to “LINK_SPEED_1G”. If the actual link speed is lower than the preferred link speed, DEP will appear red in the “Network Status” window of Dante Controller, as an indication that the DEP device’s performance might not meet expectations.
  - `"webSocketPort"`: The websocket port used by DEP. Valid port numbers are in range 1024 to 65535. If not set an ephemeral port is used. Useful when an ephemeral port is non-ideal e.g. firewall rules.
- `"mdns"`: mDNS configuration
  - `"restrictInterfaces"`: Whether to restrict mDNS advertisements to just the configured Dante interfaces. Optional boolean value, defaulting to true.
- `"clock"`: configuration of PTP clocking subsystem
  - `"enableHwTimestamping"`: Whether to use hardware packet timestamping at the Network Interface Card (NIC) level. This field is optional, and can be either a boolean value or a string, with a boolean value of false being the default. A full timestamping test using the PTP timestamping test tool (see PTP Timestamping Test Tool) should always be run to ensure that PTP will be operational with the configured value:

    - If the value is a boolean, hardware timestamping will be used if the value is true, otherwise software timestamping (which is less accurate than hardware) is used. Regardless of the value being true or false, both PTPv1 and PTPv2 packets will be timestamped. Domain/site unicast clocking is supported, and `"aes67Supported"` can be set to true.

    - The single supported string value is `"v1"`. If set to this value, DEP will hardware timestamp PTPv1 packets only. PTPv2 will be disabled, meaning that domain/site unicast clocking is not supported and `"aes67Supported"` cannot be set to true. `"v1"` can be used if a value of true fails a timestamping test and the aforementioned PTPv2-dependent features are not required.
  - `"dsaTaggedPackets"`: Whether packets read from the network interface have a DSA tag attached. Optional boolean value, defaulting to false.
  - `"hardwareInterfaces"`: The hardware network interfaces to use for packet timestamping, where different from the `"interfaces"` to be used for packet transmission and reception. This field is mandatory when `"enableHwTimestamping"` is set to either true or `"v1"`, and `"dsaTaggedPackets"` is set to true. It follows the same conventions as the `"interfaces"` field. The entries in this field cor-

respond to the entries in the "interfaces" field with a 1-1 mapping. See Networking for more information.

- "followerOnly": Sets the device to operate in PTP follower-only mode. With this set to true, the device cannot become a clock leader. This setting is optional, and defaults to false. **Note**: a DEP device running in follower-only mode cannot be used as a boundary clock. For this reason:

  - If there are third-party AES67 devices on the network, one other Dante device must be acting as a boundary clock when a DEP device operating in follower-only mode has AES67 enabled.

  - Domain/site unicast clocking is disabled for a follower only DEP.

- "hardwareClock": configuration of the hardware clocking subsystem
  - "useHwClock": A boolean field to enable the use of the clocking hardware. This field is optional. The default is false.
  - "circuitName": The name of the clock generator and adjustment circuitry. This field must be one of the supported strings in a DEP release. This field is required for this block.
  - "circuitRevision": An integer representing the circuit revision to use. This field must correspond to a supported revision and circuit in a DEP release. If not present, a value of 0 is used.
  - "i2cBus": The I2C bus device to use to communicate with the clock circuitry. If not present, the first I2C bus device "/dev/i2c-0" is used.
  - "i2cAddr": The I2C address configurable for a circuit. If not present, the default addresses for the circuit are used.
  - "extClockInputDev": The device path to the external clock input driver used in the clock feedback algorithm. If not present, this field defaults to "/dev/extclkin".
  - "extLrclockInputDev": The device path to the external GPIO clock input driver used in the hardware clock for clock phase alignment.
  - "bitClocks": An array of mappings between the sample rate and bit clock configurations. Each mapping must contain a "tdmChannels" field and a "bitDepth" field. A "sampleRate" field sets the mapping for use only at a particular sample rate. A single mapping without a "sampleRate" field is permitted and applies to all supported sample rates without an existing mapping.
  - "loadCapacitance": Optional field. Integer value for the internal load capacitance in pf to set for the clock circuit. If not set or set to a negative number the circuit's default will be used. The default and set of valid values are clock circuit specific. For DEP supported si5351b based clock circuits the default load capacitance is 10pF and the set of valid values for this field are 6, 8 and 10.
- "hostcpu": Host CPU configuration
  - "enableDdp": Set to true to enable DDP and false to disable. Default if not specified is false/disabled.
- "alsaAsrc": ALSA ASRC configuration.
  - "enableAlsaAsrc": Set to true to enable ALSA ASRC and false to disable. Default if not specified is false/disabled.
  - "txLatencySamples": Offset used by ASRC when writing audio to the DEP TX buffer measured in samples. Defaults to 48
  - "pollMode": If true, ALSA ASRC will not wait on the DEP shared memory semaphore and will instead poll the memory to determine when more data is available. Boolean defaulting to false.
  - "schedulingPriority": The application will set its scheduling priority to this. Integer in [0-100] with 100 being the highest priority. Defaults to 70
  - "cpuAffinity": The CPU affinity of ALSA ASRC will be set to this CPU. This is a zero-indexed core number. If unset, ASRC will be executed in CPUs according to the scheduler.
  - "deviceConfigurations": An array of device configurations. One for each direction on each ALSA device. Each device configuration is an object containing the following properties:

    - "deviceIdentifier": The ALSA device identifier for this device, e.g. "hw:1,0" or "hw:CARD=sofhdadsp,DEV=0". Required for each device.

- "direction": The direction to open the ALSA device in. Must be "capture" or "playback". Required for each device.

- "bitDepth": The PCM bit depth to open the ALSA device with. The device will be opened with the first format it claims to support which is that depth. Typically this maps 8 to S8, 16 to S16_LE, 24 to S24_LE and 32 to S32_LE. Integer defaulting to 24

- "bitWidthOverride": The number of bits each sample is packed into. For example, "bitDepth": 24, "bitWidthOverride": 32 is equivalent to S24_LE, so the application writes samples aligned to 4 bytes, in the form xxxxxxxx xxxxxxxx xxxxxxxx 00000000. This is useful for ALSA devices which report their supported PCM formats incorrectly, for example claiming to support S24_LE but actually writing samples aligned to 3 bytes; in such a case this could be resolved by setting "bitDepth": 24, "bitWidthOverride": 24. If unspecified, the application uses the hardware bits from the selected PCM format.

- "alsaFormat": The specific ALSA format name to open the device with. Alternative to bitDepth and bitWidthOverride.

- "numOpenChannels": The number of channels to open on the ALSA device. Integer defaulting to 2

- "alsaChannelRange": The block of ALSA channels to use. Can only be provided if numOpenChannels is specified. String of the form "X-Y" where X and Y are zero indexed channel numbers, specifying the block [X,Y] inclusive. Defaults to 0-(numOpenChannels - 1).

- "danteChannelRange": The block of DEP channels this device will read from or write to. Can only be provided if numOpenChannels is specified. String of the form "X-Y" where X and Y are zero indexed channel numbers, specifying the block [X,Y] inclusive. Defaults to 0-(numOpenChannels-1).

- "gain": Positive or negative gain in dB to apply to the audio for this device. Integer defaulting to 0

- "bufferSize": Size of the ALSA buffer to request the device to open with. Integer defaulting to 64

- "samplesPerPeriod": Samples per period to request the ALSA device to open with. Integer defaulting to 8. Note that the exact numbers for bufferSize and samplesPerPeriod are merely a request, and individual ALSA drivers are entitled to find other nearby valid values, if necessary. Many sound cards / codecs have limitations on how many periods can fit into the buffer and will adjust one or both values to make it fit.

- "latency": By default, ASRC maintains the ALSA buffer at its halfway point - which corresponds to the insertion latency of ASRC. This key overrides this behaviour, specifying the target buffer point in samples. Integer value defaulting to bufferSize / 2

- "readWriteiBuffer": For drivers that don't support MMAP (memory-mapped) buffer operations, the application can emulate the memory mapping internally by inserting an additional buffer and services that through ALSA R/W calls. If this value is >0, it specifies the size of this additional buffer. Integer defaulting to 0.

- "forceArtificialAudioTime": This setting provides an override for drivers which don't provide correct audio timestamps. If this is set to true, ASRC overrides the audio time with an artificial one calculated from sample counts.

- "product": information about the product
  - "manfId": Your company's manufacturer ID. It is assigned and provided by Audinate to you when you sign up as a DEP licensee. It is important to enter this value correctly as it may be used as part of the activation process. The manufacturer ID is provided as a 64-bit value (16 hexadecimal digits). The value for this manfId JSON field needs to be the ASCII string representation of the provided manufacturer ID hex value, where each pair of hex digits forms one ASCII character. Do not enter the trailing NUL characters ('\0'), if any, to pad the string out to the full eight character length. Some examples to

illustrate:

Audinate Provided Hex Value	ASCII Character Equivalent	dante.json 'manfId' Value
417564696E617465	'A', 'u', 'd', 'i', 'n', 'a', 't', 'e'	"Audinate"
41636D65496E6300	'A', 'c', 'm', 'e', 'I', 'n', 'c', '\0'	"AcmeInc"
4F656D3100000000	'O', 'e', 'm', '1', '\0', '\0', '\0', '\0'	"Oem1"

- `"manfName"`: Human-readable name that users will see in Dante Controller. Can be no longer than 31 characters. This is a mandatory field.
- `"modelId"`: Your product's model ID, which you must assign. Up to 8 characters long and unique for each product type. This is a mandatory field.

  The `manfId` and `modelID` are used by Dante to uniquely identify the product in the field. This is used by Dante Ready to offer Dante upgrades, and by Dante Updater to identify the firmware updates to provide to the user. In general, if the model name is different, the `modelID` should also be different, to ensure the correct Dante Ready and firmware updates can be offered for the product.
- `"modelName"`: Human-readable name that users will see in Dante Controller. Can be no longer than 31 characters. This is a mandatory field.
- `"modelVersion"`

  - `"major"`: Product version major number. Must be an integer. This is a mandatory field.

  - `"minor"`: Product version minor number. Must be an integer. This is a mandatory field.

  - `"bugfix"`: Product version bugfix number. Must be an integer. This is a mandatory field.
- `"modelVersionString"`: An arbitrary string that overrides the `"modelVersion"`. Can be no longer than 31 characters. This field is optional. If not set the `"modelVersion"` fields will be used to construct a model version string.
- `"devicePrefix"`: The Dante device name prefix used to construct a device name. The device name in Dante Controller will be `devicePrefix-nnnnnn` where `nnnnnn` is the last 3 bytes of the device mac address. Can be no longer than 24 characters, and should contain no spaces (spaces will be removed if used). This field is optional, and the prefix of "DEP" will be used if not set. Legal characters are `A-Z`, `a-z`, `0-9`, and '`-`' (dash or hyphen). Device names cannot begin with a hyphen.

- `"trialMode"`: Set to true to start the container in trial mode. If excluded or false, DEP will require activation.
- `"video"`: configuration of video subsystem.
  - `"defaultChannelNamesFile"`: the full path to a separate JSON file (details in the 'Assigning Default Channel Names' section below) that specifies custom default channel names. The file must be visible to the container running DEP, and so one place to put it is the same directory as `dante.json`. In this case, if the separate JSON file were named `defaultVideoChannels.json` this field would be set to `/dante_data/capability/defaultVideoChannels.json`. This field is optional.
- `"misc"`: Miscellaneous configuration parameters
  - `"enableIdentify"`: Set to true to enable the device Identify function and false to disable. This field is optional and the default if not specified is false/disabled.
- `"ddhi"`: Dante Device Host Interface configuration parameters:
  - `"enable"`: Set to true to enable Dante Device Host Interface (DDHI) and false to disable. All other properties in the ddhi object are only used if this value is true. This field is optional and the default if not specified is true (enabled).
  - `"clientRpcs"`: List of DDHI RPCs supported by the platform DDHI client(s).

Assigning Default Channel Names

By default, channel names for a DEP and AV-H device will use the standard numbering scheme, where each name is identical to the channel number. By creating and using an additional JSON file, default names can be customized for any or all channels.

To use this feature, the audio channel names JSON file should be created and the file name set in `"audio.defaultChannelNamesFile"` as described in the 'JSON Field Descriptions' section.

Below is an example JSON file:

```
{
    "txChannelDefaultNames": [
            {
                "channel": 1,
                "defaultName": "Line Out"
            },
            {
                "channel": 2,
                "defaultName": "Mic Out"
            }
    ],
    "rxChannelDefaultNames": [
            {
                "channel": 1,
                "defaultName": "Mic In"
            },
            {
                "channel": 2,
                "defaultName": "Line In"
            },
            {
                "channel": 4,
                "defaultName": "Aux In"
            }
    ]
}
```

The following rules apply when using this JSON file. In the event of any errors, an appropriate error message will be logged to `dante_container.log` and the file contents will be ignored:

- Either or both the "`txChannelDefaultNames`" or "`rxChannelDefaultNames`" arrays can be specified as required
- "`channel`": cannot be less than 1. Values greater than the configured Tx/Rx channel count are ignored.
- "`defaultName`": cannot be empty
- Within each array, no two channels can have the same default name
- Naming every channel is not required - unspecified channels keep their standard name
- The same applies to video channels: the file name for default video channel names needs to be specified in `"video.defaultChannelNamesFile"`

Creating Channel Groups

By default, when channel groups are enabled, channels in Dante Controller will be grouped in blocks of 16 (or up to the number of configured channels if less than 16) which are shown with the start and end channels. With the use of an additional JSON file, the displayed groups can be shown with custom names and channel groupings.

To use this feature, the channel groups JSON file should be created and the file name set in "audio.channelGroupsFile" as described in the 'JSON Field Descriptions' section.

Below is an example JSON file:

```
{
      "groups": [
              {
                    "name": "Analogue Audio",
                    "tx": [ 1, 2 ],
                    "rx": [ 1, 2 ]
              },
              {
                    "name": "Misc Tx",
                    "tx": [ 4, 7, 8 ]
              },
              {
                    "name": "Misc Combined",
                    "tx": [ 5 ],
                    "rx": [ 4, 5, 6 ]
              }
        ]
 }
```

The following rules apply when using this JSON file. In the event of any errors the file contents will be ignored:

- The group name can be up to 31 characters and must not contain the characters ".", "@" or "="
- The group must specify either a list of "tx" channels, "rx" channels or both
- An array of channels cannot be empty or contain duplicate entries
- Channel numbers cannot be less than 1 or greater than 512
- Channel numbers greater than the configured channel count are ignored
- Placing every channel in a group is not required – unspecified channels will be placed into default groups
- Currently, channels can (but should not) be duplicated in two or more Tx arrays or Rx arrays. If a channel appears in more than one array of the same type, it will be placed in the group under which it last appears.

For more information about Channel Groups in Dante Controller please see the Dante Controller help – section "Channel Groups".

Specifying Per-Channel Encodings

By default, DEP allows users to specify a set of device-level native encodings, one of which is in use at any time via selection in Dante Controller.

In some deployments, it may be useful to be able to set different native encodings for different channels. To do this, a JSON file specifying per-channel encodings can be created and the file name set in "audio.perChannelEncodingsFile" as described in the 'JSON Field Descriptions' section above.

If a JSON file is supplied:

- Encodings for each channel will be set exactly as specified in that file
- The device-level encodings contained in "audio.defaultEncoding" and "audio.supportedEncodings" will not be used (but must still contain valid entries)
- DEP will exit with an error if the file is not found or is invalid (see further below). DEP will not automatically switch to using device-level encodings in this case.
- Dante Controller will not show a device-level encoding for this device, and encoding changes will be disabled

- The encoding for a channel cannot be changed while DEP is running, as only one native encoding can be specified per channel. In order to make any changes, the JSON file must be modified as required and DEP restarted.
- To switch back to using device-level encodings, `"audio.perChannelEncodingsFile"` must be removed and DEP restarted

The following is an example JSON file:

```
{
        “defaultEncoding”: “PCM32”,
        "txChannelEncodings": [
              {
                    "channels": “1-4,7”,
                    "encoding": “PCM16”
              },
              {
                    "channels": "5",
                    "encoding": “PCM24”
              },
              {
                    "channels": "10-12”,
                    "encoding": “PCM16”,
              }
        ],
        "rxChannelEncodings": [
              {
                    "channels": “2,3,8”,
                    "encoding": “PCM24”
              }
        ]
}
```

The following rules apply when using this JSON file. In the event of any errors, an appropriate error message will be logged to `dante_container.log` and DEP will fail to start:

- `"defaultEncoding"` specifies the default encoding for any Tx/Rx channels not explicitly assigned their own encoding. This is a required field, and must be one of PCM16, PCM24 or PCM32.
- The `“txChannelEncodings”` and `“rxChannelEncodings”` arrays are both optional
- `"channels"` is a string consisting of one or more values separated by commas (spaces are not permitted). Each value can be a single channel, or a dash-separated channel range where the second value must be greater than the first. This is a required field for an array entry.
- `"encoding"` is a required field for an array entry, and must be one of PCM16, PCM24 or PCM32
- Channel numbers greater than the configured channel count are ignored
- Channels cannot overlap or be duplicated inside a `"channels"` value, or between encoding entries inside either the `“txChannelEncodings”` or `“rxChannelEncodings”` arrays
- On the other hand, the same encoding can be specified for more than one entry in either of those arrays. Of course, those entries could be combined into one by changing the assigned channels - however, keeping them separate enables easy switching of encoding for a specific group of channels between restarts of DEP.

Validating DEP JSON Files Using a JSON Schema

In the DEP examples release package, in the capabilities JSON section you will find a JSON schema for the various DEP JSON files. A JSON schema can be used by you to validate your JSON configuration file before running DEP. It can also assist you in early development when creating your initial JSON files.

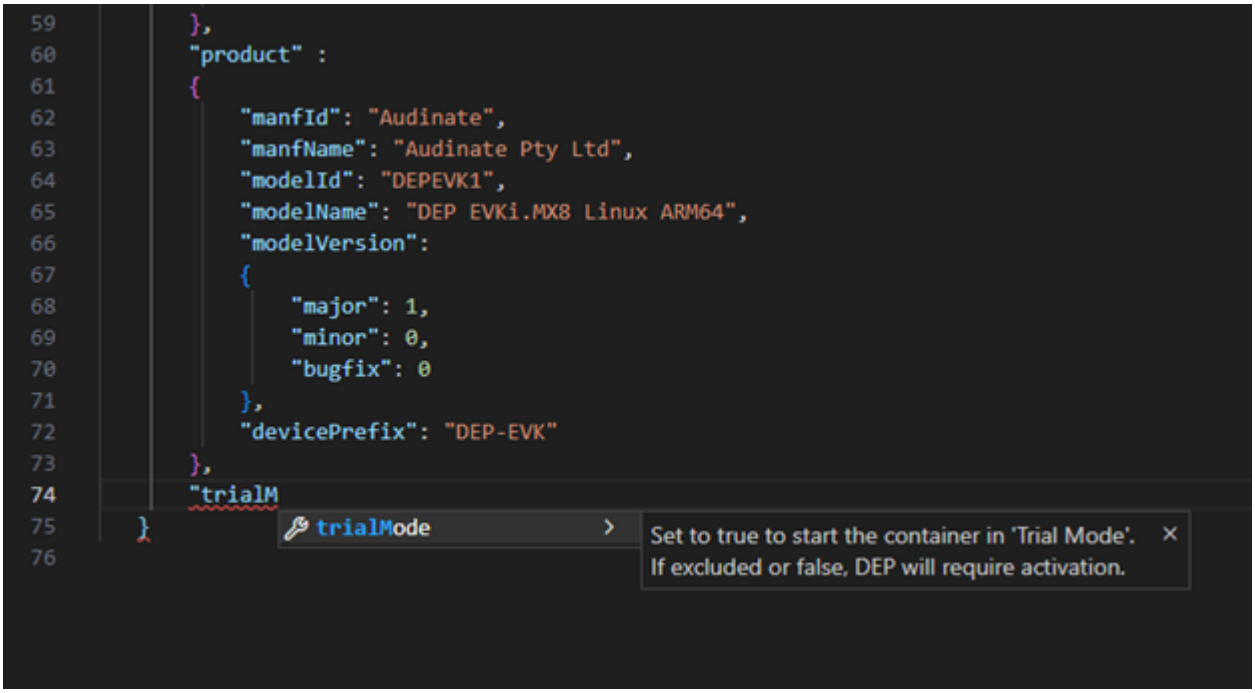
To add validation to your JSON file add a `$schema` entry to the file you intend to validate. The schema property is typically the first field of the JSON file, for example:

```
{
"$schema": "./dante.json_schema.json",
       "platform": …
```

Validating in an IDE

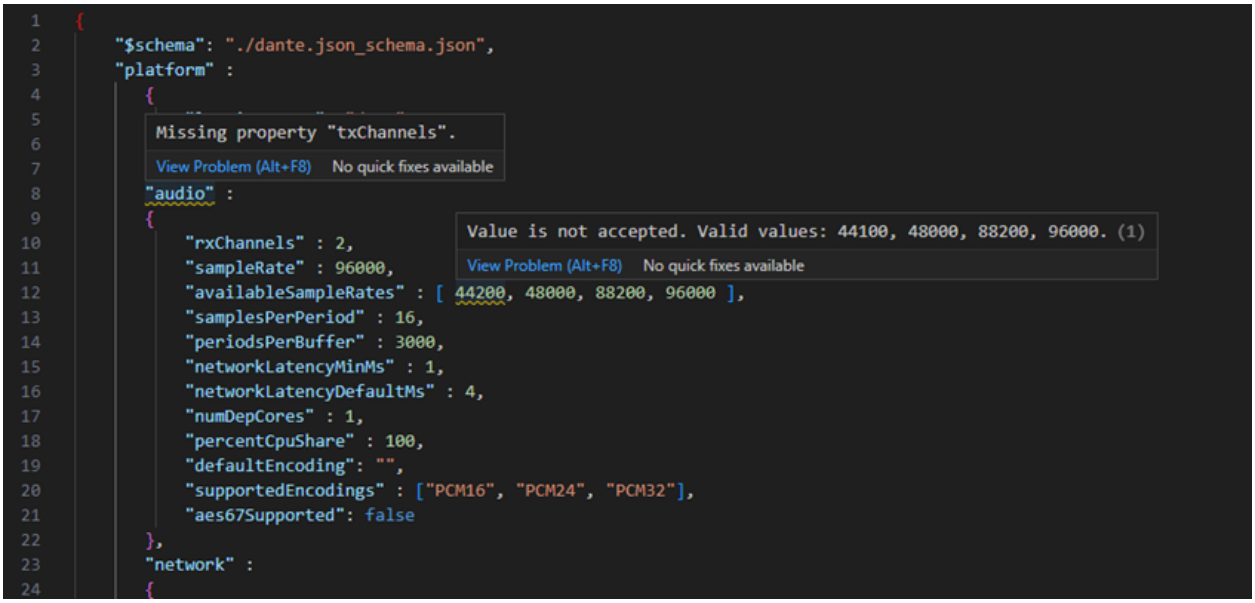
JSON Schema validation can be done in an IDE. Visual Studio Code (VS Code) is a good example of an IDE that has native schema validation and assistance.

Load the JSON file and schema into your VS Code workspace with the appropriate local path from your JSON file to the schema file. VS Code will now provide built-in field description, auto-completion of JSON fields and validation of your JSON data. for example:



If invalid values are provided, or required values are missing in the JSON, VS Code will highlight the file in the workspace view, and the field or data item in the text, with a warning.

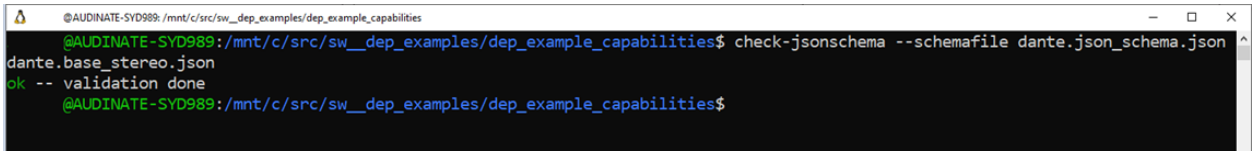
In the example below we have set an invalid sample rate of 44200Hz and have not set the required `txChannels` count property:



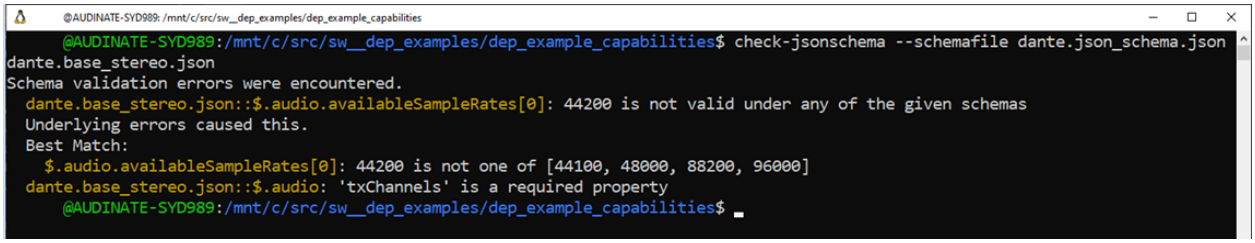
Validating in the Command Line

There are a number of command line schema validators available. In the examples below we have used `check-jsonschema` (https://github.com/python-jsonschema/check-jsonschema), a Python-based validator that is often used for command line validation.

The example below shows a JSON file with no issues:



In this example we apply the command line validator tool to the same JSON file shown in the above VS Code example:



The two problems identified with the JSON are output as errors - 44200 is not a valid sample rate, and `txChannels` is a required property.

Appendix 4: dhcpcd Patch for 172.31.x.x Link Local Addressing

The "`dhcpcd-1-ipv4llprivate.patch`" provided with the DEP release adds a `` `ipv4_ll_private` `` option which can be specified for an interface to use the private 172.31.x.x/16 address space for IPv4 link-local address assignment. This should be set on the secondary port on platforms supporting redundancy. This patch is available at:

`DEP-EVK-IMX8.sources.tgz/DEP-EVK-IMX8/board/audinate/dep-evk-pico-imx8mm/patches/dhcpcd/`

Appendix 5: ALSA ASRC Integration

Integrating ALSA ASRC may require some trial and error with the `atest` tool to determine the capabilities of your ALSA devices, and some minor tuning of ALSA ASRC alongside DEP.

ALSA Diagnostic Tool

The DEP container comes with a ALSA testing and diagnostic tool. The primary purpose of this tool is to verify that the audio path from inside the container to your ALSA devices is functioning, without DEP in the loop. The tool is called `atest` and can be used while the DEP container is started up, while ALSA ASRC is turned off and nothing else is using any ALSA devices. To see the usage for atest, while the container is running, run:

```
sudo ./crun exec dante /dante/atest -h
```

atest Usage

`atest` can play test tones out through playback devices, and displays VU meters for capture devices. To open one or more devices and view basic performance statistics, the usage of `atest` is as follows:

```
sudo ./crun exec dante /dante/atest -d '<device config>' [-d '<device config>' ...] [-D] [-C CPU] [-S FREQ | -T TEXT]
```

These options do the following:

- Each instance of the `-d '<device config>'` option opens an ALSA device as defined in the device configuration string. The syntax for this string is detailed below.
- The `-D` option causes `atest` to output a status line which updates approximately 20 times per second. This includes processing time for each ALSA poll, samples processed per poll and rate tracking statistics of ALSA audio time against the system clock.
- The `-C` option causes `atest` to set its CPU affinity to a specific core.
- The `-S` and `-T` options create test signals on all playback devices. The `-S` option creates a sine wave tone with the given frequency, and the `-T` option outputs audio which appears as the given text in a spectrogram.

Note that the Text option is cpu-intensive and should be used with at most one device and maximum 2 channels.

Each device configuration string specifies a single ALSA device and direction to open, along with various configuration options. The string has the following syntax:

```
DEVICE_NAME DIRECTION [options]
```

Where:

- `DEVICE_NAME` is the ALSA name of the device. For example `hw:0,0` or `hw:CARD=card_name,DEV=device_name`
- `DIRECTION` is either `play` for opening the device as a playback device, or `capt` for opening the device as a capture device.
- Options is any combination of the following options:

  Basic Options

  - `RN` - Open the ALSA device at sample-rate N. Defaults to selecting the nearest sample-rate to 48000
  - `bX` - Set the ALSA format/bit depth to use. X can be a specific format name such as `S24_LE` or `FLOAT_LE`, or can be a number. If given a number, atest uses the first format supported with that bit depth. For example, `b24` will select the first format the device supports between `S24_LE`, `S24_3LE`, `S24`. Defaults to `b24`.
  - `BN` - Request a buffer size of N samples when opening the ALSA device. Defaults to `B64`.
  - `cN` - Open N channels on the ALSA device. Defaults to `c2`.
  - `coX` and `ccY` - These options together define a block of channels to use, within the set of channels opened by `cN`. The block of channels starts at X (zero indexed) and is a block of Y channels. For example, `c16 co8 cc8` opens 16 channels, but causes `atest` to only read or write from the last 8.
  - `gN` - Apply a gain of N decibels to the audio, where N is an integer, and can be negative. Defaults to `g0`.
  - `PN` - Request a period size of N when opening the ALSA device. Defaults to `P8`.

- **Workaround Options**

  - Occasionally ALSA drivers are incompatible with basic device configuration. Several options are provided to work around these issues. If using atest shows that these options are necessary, they can be applied similarly in ALSA ASRC, see Configuring ALSA ASRC in DEP.
  - `hbN` - Override the alignment of the chosen ALSA format, instead align the data to N bits. For example, occasionally ALSA drivers report that they support `S24_LE` (24 bits of data aligned to 32-bit steps), but then output 24-bit samples aligned to 24-bit steps. In this case, the options `b24 hb24` are required. Defaults to unused.
  - `rwN` - ALSA ASRC and `atest` use the mmap mode for reading and writing audio to ALSA. If the ALSA driver does not support mmap mode, setting this option to non-zero switches to using interleaved read and write calls (`readi` and `writei`). N is the size of an internal buffer which should be 0 (to use mmap mode) or the size of the ALSA buffer to use `readi/writei`. Only in certain exceptional cases will this buffer need to a different size to the ALSA buffer. Defaults to `rw0`.
  - `fN` - In the case where an ALSA driver does not provide correct audio timestamps, ALSA ASRC and `atest` can infer the audio time from samples consumed/produced by ALSA. If N is not zero, `atest` switches to this mode. Defaults to `f0`.

For example, on the DEP-EVK-IMX8, the following command opens ALSA device hw:1,0 in playback and capture mode, with bit-depth 32, sample-rate 96000, a buffer size of 384 samples and 4 channels in each direction, with a 440Hz sine wave playing over the playback channels using the following command:

```
/dep/dante_package/crun exec dante /dante/atest -d 'hw:1,0 play b32 R96000 B384 c4' -d 'hw:1,0 capt
b32 R96000 B384 c4' -D -S 440
```

Interpreting atest Output

When opening one or more ALSA devices, atest will output detailed information about the ALSA driver's reported parameter space, as well as the exact parameters selected. If the `-D` option is included, it will also output an updating status line of the current ASRC performance.

ALSA Parameter Space

For each ALSA device opened, `atest` outputs a parameter list structured like below:

```
Setting up playback on 'hw:1,0', using p48000Hz / 32bits / 16ch.
Unconstrained hw params
ACCESS:  MMAP_INTERLEAVED RW_INTERLEAVED
FORMAT:  S16_LE S24_LE S32_LE
        <----- Other parameters omitted ----->
hw:1,0p Buffer(480)=480|480, Period(8)=8|8, (req)=ret|get
Constrained hw params
ACCESS:  MMAP_INTERLEAVED
   <----- Parameters and diagnostics omitted ----->
hw:1,0p type=play, rate=48000, width=32, hwbits=32, endian=LE, fp=0, period=8, buffer=480,
latency=240
hw:1,0p Actual audio format is fmt=10, S32_LE{32/32}, 48000Hz, 16ch
```

This specifies both what values the ALSA driver claims to support, under `Unconstrained hw params`, what parameters should be set based on the settings, under `Constrained hw params`, and what values were selected after applying the ALSA settings. For example, in the above example the driver claims to support both `MMAP_INTERLEAVED` buffer access and `RW_INTERLEAVED` read/write access. The default used by `atest` and ALSA ASRC is `MMAP_INTERLEAVED`, so this is selected in the constrained parameters. Note that often these values will be reported as a range, where the driver or device only support specific values within that range. For example, `RATE: [44100 192000]` may mean the device supports 44100, 48000, 88200, 96000 and 192000 sample rates.

Of note is the line:

```
hw:1,0p Buffer(480)=480|480, Period(8)=8|8, (req)=ret|get
```

This line indicates the requested ALSA buffer size and requested ALSA period in samples in the brackets. The ALSA driver selects the nearest valid buffer and period size. This actual size selected by the ALSA driver is shown to the right of the equals sign.

Status Line

When the `-D` option is included, `atest` will output an updating status line to the terminal for each ALSA device and direction opened. The line updates approximately 20 times per second and has the form:

```
hw:1,0p 658.403s, sch=[0.25|0.31|0.40]ms, sam=[27.6| 34.2|156.6]us, fil=[1.5,2.0,52.1]us, cpu= 1,
FFM=0, ofs=-1.48Hz (-30.8ppm)
```

- `hw:1,0p` - The device and direction this status line is reporting on. The direction is indicated by the character, either 'p' or 'c' suffixing the device name.
- `658.403s` - The time, in seconds, since sampling started.
- `sch=[0.25|0.31|0.40]ms` - The minimum, latest and maximum (respectively) scheduling interval observed between the current and previous wake-up time. The software attempts to schedule at a nominal rate, as set by a poll loop, at 0.3ms. The variance in these values can be used as an indication for tuning the system and scheduling priorities. The ALSA buffer size should be twice as large as the worst case scheduling period.
- `sam=[27.6| 34.2|156.6]us` - The minimum, latest and maximum (respectively) sample processing time, in microseconds, for a single poll. The ratio between these values and the scheduling interval is a measure of the CPU load/memory bandwidth imposed by writing/reading samples to or from ALSA. This is approximately indicative of the load for ALSA ASRC; `atest` does not perform interpolation, but interpolation only increases this load by 10-15%. An excessively large maximum processing time is an indicator that the atest process was pre-empted, so its real-time priority may be insufficient.
- `fil=[1.5,2.0,52.1]us` - The minimum, latest and maximum (respectively) processing time by the rate tracking element. `atest` performs rate-tracking of the ALSA clock against `CLOCK_MONOTONIC_RAW`, this number should be much smaller than the sample processing times. Again, an excessively large value may indicate that the software has been pre-empted.
- `cpu= 1` - The CPU ID the process is currently running on. On multi-processor systems, it often changes, as the task is migrated between cores. This can be detrimental to cache efficiency, so the `atest` process may be locked to specific core via command line option. The cpu affinity of ALSA ASRC can similarly be set using the `useDedicatedCore` option.
- `ofs=-1.48Hz (-30.8ppm)` - The measured clock frequency offset between the ALSA clock and CLOCK_MONOTONIC_RAW, in Hz and parts-per-million. Positive values indicate that the source is faster than the destination. So in the case of this playback device, CLOCK_MONOTONIC_RAW is running 30.8 parts-per-million slower than the ALSA clock.

Tuning ALSA ASRC

ALSA ASRC itself needs very little tuning, but the platform and ALSA devices may require tuning. The primary tools for tuning ALSA ASRC itself are the ALSA buffer size (bufferSize) and the ALSA period size (`samplesPerPeriod`).

Generally, a larger ALSA buffer will cover-up scheduling delays and provide a larger allowance for unstable ALSA clock rates, in exchange for increasing the latency introduced by ALSA ASRC. For devices with very stable rate-tracking and scheduling, the minimum buffer size is two ALSA/DEP periods, whichever is larger.

The period size can impact the performance of ASRC. A smaller ALSA period, down to DEP's period size, will usually improve the stability of ALSA ASRC. Some ALSA drivers will limit the number of periods in the buffer, meaning a large buffer can force a larger ALSA period, reducing the stability of ALSA ASRC.

System Tuning

Naturally, preventing or reducing scheduling delays is preferable to increasing the buffer size. To this end, it is recommended to set the `schedulingPriority` parameter high, and/or use core isolation along with setting the `cpuAffinity` setting.

The `txLatencySamples` setting determines how far ahead of the current DEP period TX audio is written into the buffer. Choosing the value for this setting is dependent on the system's scheduling latency and stability, as well as the load on ASRC. The selected number must be at least the worst-case scheduling delay plus the amount of time taken to process one DEP period of samples. This can be approximated by looking at the worst variance from the

0.3ms target scheduling in atest, and adding the DEP period (assuming ASRC is under maximum load), but must be validated with a long-term glitchtest using ALSA ASRC. For example, on the DEP-IMX8-EVK:

- At 48000 sample rate and period-size 16, ALSA ASRC runs stably with `txLatencySamples` of 24.
- At 96000 sample rate and period size 32, ALSA ASRC runs stably with `txLatencySamples` of 48.

ALSA Device Tuning

ALSA devices can be tested and tuned using `atest`, and then any settings or work-around necessary in atest can be applied to ALSA ASRC. Along with the basic configuration `dante.json` options, there are workaround options corresponding to those in atest:

- `"bitWidthOverride"` corresponds to the hbN option
- `"readWriteiBuffer"` corresponds to the rwN option, and
- `"forceArtificialAudioTime"` corresponds to the fN option.

Index

A

Activation 31-32
ALSA 61
ALSA ASRC 65, 83
ALSA ASRC Integration 96
ALSA Diagnostic Tool 96
Alternatives to Modifying the Kernel Parameters 29
API 59
Applications 32
Assigning Default Channel Names 91
atest Usage 96
Audio Interface 59
Audio Interface Metadata 59
Audio Interface Timing 59
Audio Muting 51
Audio Options 82
AV-H 78

B

Bit clock 53
Bit Clock Configuration 53
Building a Product 14

C

Cgroup Support 75
cgroup v2 25
Cgroups Versions 75
Channel Groups 91
Clock 83
Clock Feedback 52
Clocking Circuits 54
Clocks 53
Coalesce Parameter Reset Issue 78
Collecting Information for Audinate Support 35
config.json 17
Configuration File 83
Configuration Files Setup 17
Configuration of Si5351B Output Clock Lines 54
Configurations 34
Configuring ALSA ASRC 68
Configuring the Container for ALSA 66
Container Configuration 78
Container Configurations 34
Control 71
CPU cores 38
CPU Isolation 28
CPU load 37
CPU usage 49
Creating Channel Groups 91

D

DAC 54
Daisy Chain Mode 43
Daisy Chaining 78
Dante AV-H 78
Dante Configuration 81
Dante Device-Host Interface 73
dante.json 17, 53
DEP Field Updatable Firmware 22
DEP v1.5 25
dep_support_collection 35
DepAlsaBridge 33, 61
DEP-EVK-IMX8 41
DepLoopback 33, 60
Device Activation 32
Device Configurations 34
Device-Host Interface 73
DHCP Configuration 49
dhcpcd Patch 96
dinfo 33, 62
Disk Space Usage 27
dplay 33, 62
drecord 34, 63
DSA 41, 43
DSA Duplicate Multicast Data Issue 78
DSA Example 41
dsoundcard 34, 64
dtest 34, 65
Duplicate Multicast Data Issue 78

E

Encodings 92
Encrypted Flow Testing 39
Ethernet driver checksum issue 77
Ethernet Driver Coalesce Parameter Reset Issue 78
Example Applications 32

Example Audio Applications 60
Example Configuration File 83

F

Familiarise Yourself with DEP 16
Feedback clock 53
Firewall Configuration 49
Flow Testing 37, 39
Flows 37
FPP 37
Frames per packet 37
Freescale Ethernet driver checksum issue 77
Freescale Ethernet Driver Coalesce Parameter Reset Issue 78

G

GPIO 53

H

Hardware Clock 83
Hardware Clocks 51
Hardware RNG 77
Hardware Timestamping 51
hostcpu 83

I

i.MX EVK Network Driver 77
I2S 53
IDE 94
IGMP Querier 78
init 25-26
Installing DEP 19
Interpreting atest Output 98
Introduction 14
isolcpus 28

J

JSON Field Descriptions 85
JSON Files 93
JSON Schema 93

K

Kernel and Platform Configuration 75
Kernel Parameters for CPU Isolation 28

L

Latency 37, 69
leader clock 51
Link Local Addressing 96
List ALSA Devices 68
logDirectory 82
Low Latency Configuration 76
LRCLK 53

M

Master clock 53
MCLK 53
MCP47CVB02 54
mDNS Visibility 50
Metadata 59
Miscellaneous 83
Monitoring 71
Multiple PHYs 48

N

Namespace Support 76
Network Configuration 77
Network Driver 77
Network Interrupt Coalescing 49
Network Settings 82
Networking 41
nohz_full 28

O

OCI container 78
OCI run time 78
On-Device Monitoring and Control 71

P

Packet Timestamping 43
Packets per second 37
PCM encoding 59
Per-Channel Encodings 92
Performance 37
Persistent Data 24
PHYs 41
Platform 81
Platform Configuration 75
Ports 49
PPS 37
Precision Time Protocol 51
Prerequisites 19
Product Information 83
PTP 51, 54, 83
PTP Timestamping Test Tool 35

R

rcu_nocbs 29
Real Time Considerations 69
Reducing Disk Space Usage 27
Redundancy 45, 48
RNG 77
Running ALSA ASRC 66

S

Sample Rate Change 60
Sample Rate Changes 53
SCLK 53
Si5351B 54
Si5351B Output 54
Single PHY 47
Specifying Per-Channel Encodings 92
SquashFS Support 76
Starting and Stopping DEP 18
Steps 19
Switch 41, 43, 45
Switch configuration 41
System and Configuration Checks 34
systemd 26, 29
sysvinit 25, 30

T

TDM 53
TDM channels 53
Testing ALSA 68
Testing Device Audio 68
Timestamping Test Tool 35
Tuning ALSA ASRC 99

U

Un-encrypted Flow Testing 37
Updating 22
Updating the Host Platform 24

V

Validating DEP JSON Files Using a JSON Schema 93
Validating in an IDE 94
Validating in the Command Line 95
VCXO 54
VLANs 41

W

Word clock 53