Audio Device Definition Files

PurePath Wireless Configurator version 1.1.1 and later define all properties of the supported external audio devices using an XML based file format. This page describes how these XML files are structured, and provides the instructions for:

- Creating new audio device definition files for currently unsupported audio devices.
- Creating modified versions of the current audio device definition files.

The audio device definition file describes a large number of device properties:

- Information about the device (e.g. name and datasheet URL).
- Device capabilities (e.g. number of inputs/outputs and supported sample rates).
- A description of its pin interface for different hardware platforms, including up to 4 custom control pins for controlling reset, power etc.
- Default configuration sequences, including I2C, control pin and delay operations, that are executed when the audio device transitions between the OFF, SR-SWITCH, INACTIVE, LOW-POWER and ACTIVE states.
- Description of how volume control works, so that firmware can translate a volume setting specified in dBs into I2C write operations.

Read all information on this page carefully, and use caution when testing new or modified audio device definitions. Incorrect settings may cause random noise or very loud output volumes that can be very harmful to both ears and equipment.

File Handling

Audio device definition files provided by Texas Instruments are stored in "<Install directory>\audio_devices", in most cases "C:\Program Files\Texas Instruments\PurePath Wireless Configurator\audio_devices". User-created audio device definition files are stored in "<My Documents>\Texas Instruments\PurePath Wireless Configurator\audio_devices". At startup, the Configurator will load all audio device definition files (*.ppwadd) currently located in these directories.
NB: The Configurator does not detect changes to these files while running and needs to be restarted for any changes to take effect.

The Configurator requires and assumes that all audio device definition files are:

- Unique: Each audio device definition file must be identified uniquely by its **name** tag. Installer-provided audio device definition files are ignored if there are user-created files with the same **name** tag. This allows for modifications while keeping the original files.
- 100% correctly formatted: The Configurator will fail to start, crash or produce non-functional firmware images if
  - XML validation, based on "audio_device_definition.dtd", fails.
  - Formatting of XML tags, attributes, attribute values, and certain tag values is changed. This includes casing, extra spaces and use of special characters (e.g. the comma-separated lists must not contain spaces before or after the commas, and each item in the list must have correct casing).
  - The definition contains typographical errors or unsupported values.

Installed audio device definition files should not be modified. When customizing audio device definition files, it is recommended to work on a copy in the directory for user-created files. This copy will override the installed file if the **name** tag is unchanged.

When selecting audio device in the Configurator, user-generated audio device definitions are indicated as follows:

- **"+"** - Definition for new device, not included in the Configurator installation
- **"#"** - Definition for existing device, included in the Configurator installation. The user-generated definition overrides the installed definition. When the user-generated definition is deleted, the Configurator will (after restarting) resume using the installed definition.

File Format

The XML format contains the following **tags** (elements) and *attributes* listed below. For all examples below using quoting, the quoting itself should not to be included in the XML file.

- **audio_device**
  - **name** (string) - Unique name of the audio device, which should be kept short and descriptive (e.g. "AIC3101" for the Texas Instruments TLV320AIC3101). May only contain upper-case letters, numbers and dashes.
  - **type** (string) - Type of audio device, which should be kept short and descriptive (e.g. "Audio codec", "S/PDIF transmitter" or "Digital amplifier").
  - **desc** (string) - The audio device description displayed in the Audio Interface panel.
  - **input_count** (decimal) - Number of input channels supported by the audio device's serial audio interface (e.g. "2" for AIC3101).
  - **output_count** (decimal) - Number of output channels supported by the audio device's serial audio interface (e.g. "2" for AIC3101)
  - **datasheet *** - Optional tag that provides datasheet links on the start page. There can be multiple datasheet tags.
    - **path** (string) - Defines the audio device category and device name using the following format "<Category>//<Category>", (e.g. "Audio codec//TLV320AIC3101").
    - **link** (string) - Web link to the datasheet, preferably a permanent link (e.g. "http://www.ti.com/lit/gpn/tlv320aic3101").
  - **attr_spec +** - Specifies supported audio device attribute values, all of which are required. Each specification consists of a comma-separated list of the allowed attribute values.
    - *name* (string) - Name of the attribute.
    - *custom_setup* ("0 or 1") - 0 for default setup (i.e. settings supported by the default configuration sequences) and 1 for custom setup (i.e. settings supported by the audio device).
    - The following attributes must be specified ("(c)" denotes that the custom_setup attribute need to be present with values 0 and 1):
      - Hardware platform - Supported hardware platforms: PurePath Wireless EB,CC85XX Headset,Custom design
      - Interface format (c) - Supported serial audio interface formats: I2S,LJF,RJF,DSP
      - Resolution (c) - Supported serial audio interface sample resolutions: 16,24
      - Clock source (c) - Supported serial audio interface clock sources: External,Internal with MCLK,Internal without MCLK
      - BCLK multiplier (c) - Supported BCLK/WCLK ratios: 32,64,128
      - MCLK multiplier (c) - Supported MCLK/WCLK ratios: 128,256,384,256
      - Sample rate - Supported sample rates: 32000,40275,44100,48000
  - **adx_pin_spec +** - Specifies for each supported hardware platform which ADx pins may be used for input and output (one adx_pin_spec for each supported direction).
    - *dir* ("Input" or "Output") - Specifies the direction of the audio on the ADx pin, with CC85XX as reference.
    - **io_mapping** (hex-16) - Bit vector specifying which ADx pins are allowed for the given direction on the given platform (e.g. "0100" for AD1_GIO8 or "0280" for AD0_GIO7 and AD2_GIO9).
      - *hardware_platform* (string) - Name of the hardware platform, as specified by the attr_spec.
  - **i2c_interface ?** - If present, specifies that configuration is I2C-based.
    - **clock_rate** (decimal) - Specifies the I2C clock rate in kHz: 25, 100 or 400.
  - **ctrl_pin_spec *** - Specifies each control pin, which can be incorporated into the state configuration sequences. There may be 0 to 4 control pins.
    - *name* (string) - Name of the pin function, as used in the state configuration sequences. Should be kept very short and descriptive (e.g. "Reset"). May only contain letters and numbers.
    - **io_mapping_name** (string) - Name of the pin function, as displayed in the IO Mapping panel. Should be kept reasonably short and descriptive. May only contain letters, numbers, dashes and space.
    - **io_mapping +** (hex-16) - Bit vector specifying which GIOx pins (GIO1 to GIO15) are allowed on the given platform (e.g. "0002" for GIO1 or "3C04" for GIO2 + GIO10 through GIO13).
      - *hardware_platform* (string) - Name of the hardware platform, as specified by the attr_spec.
    - **initial_setting** (decimal) - The initial pin setting, which is applied shortly after CC85XX reset. Use 0 or 1 to drive low or high, or H or L to pull low or high.
    - **active_level** (decimal) - Active pin level, 0 for active low or 1 for active high.
  - **dig_input_valid_pin_spec ?** - Specifies the digital input valid pin, if supported by the audio device.
    - **io_mapping +** (hex-16) - Bit vector specifying which GIOx pin (GIO1 to GIO15) is allowed on the given platform (e.g. "1000" for GIO12 or "3FFE" for GIO1 through GIO13).
      - *hardware_platform* (string) - Name of the hardware platform, as specified by the attr_spec.
    - **active_level** (decimal) - Active pin level, 0 for active low or 1 for active high.
  - **state_cfg_sequence +** (string) - Specifies the default configuration sequence for the given audio device state transition.
    - *transition* (string) - Name of audio device state transition.
    - State configuration sequences must be specified for the following state transitions (refer to the CC85XX Family User's Guide for further details on the device states):
      - OFF to SR-SWITCH

        - SR-SWITCH 32000 to INACTIVE
        - SR-SWITCH 40275 to INACTIVE
        - SR-SWITCH 44100 to INACTIVE
        - SR-SWITCH 48000 to INACTIVE
        - INACTIVE to LOW-POWER
        - LOW-POWER to ACTIVE
        - ACTIVE to LOW-POWER
        - LOW-POWER to INACTIVE
        - INACTIVE to SR-SWITCH
        - SR-SWITCH to OFF
    - Use the following syntax:
        - Write operation - `w AA RR DD ...`, where `AA` is the I2C address, `RR` is the register address and `DD` is the first data byte"
        - Delay operation - `d D`, where `D` (1-65535) specifies the delay in units of 32 us
        - Pin control - `p N S`, where `N` is the control pin's "Function name", and `S` is the desired setting (0 or 1 to drive, L or H to pull)
        - Comment - `# Comment text`
    - **Important:** Observe the Guidelines for Volume Control in State Transitions (below).
- **vol_gain_range_spec +** (float,float) - Specifies the range of the controlled volume control registers in dBs for each supported audio direction, using the following format "<minimum above muting>,<maximum>", e.g. "-63.5,24.0" for the TLV320AIC3204.
    - *dir* ("Input" or "Output") - Specifies the associated audio direction, with the audio device as reference.
- **vol_ctrl_spec +** - Specifies how volume settings in the range specified by the vol_gain_range_spec are translated into a I2C register write operations. Each vol_ctrl_spec can specify a random number of fixed register write operations, followed by a single, patched write operation containing the variable field(s) that depend on the volume/mute setting.
    - *dir* ("Input" or "Output") - Specifies the associated audio direction, with the audio device as reference.
    - *type* (string) - Specifies the volume operation performed by this vol_ctrl_spec. "Mute", "Volume" and "Unmute" must be defined for all IO channels supported by the audio device.
        - Mute - Mutes the specified IO channels
        - Volume - Sets the volume for the specified IO channels. Use when volume and muting are controlled by different registers. Will always run before "Unmute"
        - Unmute - Unmutes the specified IO channels. Use when volume and muting are controlled by different registers.
        - Volume,Unmute - Sets the volume for and unmutes the specified IO channels. Use when volume and muting are controlled by the same register.
    - *io_channel_mask* (hex-8) - Specifies which IO channels the vol_ctrl_spec is associated with. There are independent IO channel masks for audio input and output. The IO channel mask contains one bit for each audio channel supported by the application role, and the bits are mapped by ADx pin index and then by I2S/DSP channel index, for example:
        - Using AD0 with 2 I2S channels: Bit 0 maps to AD0.LEFT, bit 1 maps to AD0.RIGHT.
        - Using AD1 with 4 DSP channels: Bit 0 maps to AD1.CH0, bit 1 maps to AD1.CH1, bit 2 maps to AD1.CH2, bit 3 maps to AD1.CH3.
        - Using AD0 and AD2 with 2 I2S channels each: Bit 0 maps to AD0.LEFT, bit 1 maps to AD0.RIGHT, bit 2 maps to AD2.LEFT, bit 3 maps to AD2.RIGHT.
    - **pre_patch_cfg_sequence ?** (string) - Optional configuration sequence performed before the patched operation. Refer to state_cfg_sequence for a description of the syntax.
    - **vol_precision ?** (decimal) - Volume setting: Number of bits after the decimal point, 0 to 3 bits (e.g. "2" for a precision of 0.25 dB)
    - **vol_negate ?** (decimal) - 1 to multiply the volume setting by -1, 0 otherwise. Set when the register value decreases with increasing volume setting.
    - **vol_add_value ?** (decimal) - Value added to the volume setting after optional negation, using the precision of the volume register setting. When the desired volume setting on the audio device is 0.0 dB, this is the value of the volume register field.
    - **vol_left_shift ?** (decimal) - Shift to perform to align the volume control field correctly. See "Volume Control Algorithm" for further details.
    - **vol_field_size ?** (decimal) - Number of bits in the register field containing the volume setting.
    - **patch_i2c_addr** (hex-8) - I2C address to be used in the patched operation (bits 7:1, bit 0 shall always be 0).
    - **patch_reg_addr_size** (decimal) - Number of bytes in the audio device register address in the patched operation (1 to 4).
    - **patch_reg_addr** (hex-8, hex-16, hex-24 or hex-32) - Audio device register address in the patched operation (hex-8 if 1 byte, hex-16 if 2 bytes and so on).

        - **patch_reg_data_size** (decimal) - Number of bytes in the register data field in the patched operation (1 to 4).
        - **patch_reg_data** (hex-8, hex-16, hex-24 or hex-32) Static register data value to be OR'ed in the patched operation (hex-8 if 1 byte, hex-16 if 2 bytes and so on).
        - **patch_read_mask** (hex-8, hex-16, hex-24 or hex-32) - Bit mask indicating the field values that need to be left unmodified by volume control register operations. If non-zero, the patched register value is read and masked with this before being written (hex-8 if 1 byte, hex-16 if 2 bytes and so on).
    - **ana_input_sw_mute_spec ?** - Enables software-based muting (by halting audio slice production and transmission) to eliminate or reduce "clicks" and "pops" when activating/deactivating and muting/unmuting analog inputs (e.g. for audio codecs and ADCs).
        - **post_mute_delay** (decimal) - Delay (0-65535 in units of 32 us) after software muting, before performing:
            - The "INACTIVE to LOW-POWER" state configuration sequence.
            - "Mute" volume control operations (see **vol_ctrl_spec**).
        - **pre_unmute_delay** (decimal) - Delay (0-65535 in units of 32 us) before software unmuting, after performing:
            - "Unmute" or "Volume,Unmute" volume control operations (see **vol_ctrl_spec**).

Volume Control in State Transitions

The CC85XX will only send input volume control updates to the audio device in the LOW-POWER and ACTIVE states, and output volume control updates in the ACTIVE state. When activating audio inputs and outputs, the initial volume control update happens some time after state transitions

- "INACTIVE to LOW-POWER" or "LOW-POWER to ACTIVE" for audio inputs
- "LOW-POWER to ACTIVE" for audio outputs

The delay between these transitions and the volume control initialization (to the desired volume setting) is typically in the order of 1 ms, but can be much more if the transitions end with a delay operation. Also, for audio inputs, the "LOW-POWER to ACTIVE" can delay the initial update. If not handled correctly, the audio device may start at an incorrect default volume setting (e.g. maximum output volume), or the volume setting used the last time the audio inputs or outputs were active. This can typically cause a loud "pop" at startup. Follow these guidelines to avoid such problems:

- The state configuration sequences shall never unmute or set other than minimum gain in the registers specified in the **vol_ctrl_spec** specifications.
- Before enabling audio inputs in the "INACTIVE to LOW-POWER" sequence, input muting must be enabled and minimum input gain must be set in the registers controlled in the **vol_ctrl_spec (dir="Input")** specifications. This muting initialization can be done in one of two ways:
    - In "OFF to SR-SWITCH" if needed (depending on register reset values), and in "LOW-POWER to INACTIVE".
    - In "INACTIVE to LOW-POWER" before enabling the audio inputs.
- Before enabling audio outputs in the "LOW-POWER to ACTIVE" sequence, output muting must be enabled and minimum output gain must be set in the registers controlled in the **vol_ctrl_spec (dir="Output")** specifications. This muting initialization can be done in one of two ways:
    - In "OFF to SR-SWITCH" if needed (depending on register reset values), and in "ACTIVE to LOW-POWER".
    - In "LOW-POWER to ACTIVE" before enabling the audio outputs.

Volume Control Algorithm

The following pseudo-code describes how the register value for the patched I2C write operation is generated for "Mute" and "Unmute" volume control operations:

- `regValue` is the register value to be written

```
// Set static part
regValue = patch_reg_data;

// Insert retained part, if any
if (patch_read_mask) {
    regValue |= "read register value" & patch_read_mask;
}
```

The following pseudo-code describes how the register value for the patched I2C write operation is generated for "Volume" and "Volume,Unmute" volume control operations.

- `"volume setting"` is the desired volume setting in the range specified by **vol_gain_range_spec** in 0.125 dB steps (e.g. -41 for -5.125 dB)

- `regValue` is the register value to be written

```
// Set static part
regValue = patch_reg_data;

// Insert retained part, if any
if (patch_read_mask) {
    regValue = regValue | ("read register value" & patch_read_mask);
}

// Volume calculation: Apply precision
volValue = "volume setting" >> (3 - vol_precision);

// Volume calculation: Negate?
if (vol_negate) {
    volValue = -volValue;
}

// Volume calculation: Apply add value
volValue = volValue + vol_add_value;

// Volume calculation: Mask out sign bits
volValue = volValue & ((1 << vol_field_size) - 1);

// Volume calculation: Align with register
volValue = volValue << (vol_left_shift + (8 * patch_reg_data_size) - 32);

// Insert the volume setting
regValue = regValue | volValue;
```