Application Report
SLOU408 – June 2015

PurePath™ Console 3 User Manual

Web Technologies

ABSTRACT

This document provides an overview of the functions available in the PurePath™ Console 3 (PPC3) Graphical User Interface (GUI) application.

Table 1. Document History

Version	Date	Author	Notes
1.0	June 2015		First release

WARNING: EXPORT NOTICE

Recipient agrees to not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S., EU, and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S. or other applicable laws, without obtaining prior authorization from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.
According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this technology is classified as follows:

US ECCN: 3E991
EU ECCN: EAR99

And may require export or re-export license for shipping it in compliance with the applicable regulations of certain countries.

Contents

Figures

Tables

1 Overview

PurePath™ Console 3 (PPC3) is a software platform for Texas Instruments' audio devices. Many of these audio devices can be configured, tuned, and validated. This platform has the ability to support a number of TI's audio devices with features that make the audio tuning experience more intuitive and exciting.

The features of PPC3 are explained in this document.

2 App Center

The App Center is the home page to manage EVM Apps in the PPC3 App GUI. The PPC3 App GUI displays all the installed and available EVM Apps as shown in Figure 1. Sign in using a TI account to view and install the available apps. This section explains the following in detail:

- Sign in button
- Installed EVM apps
- Available EVM apps
- App update
- Platform update

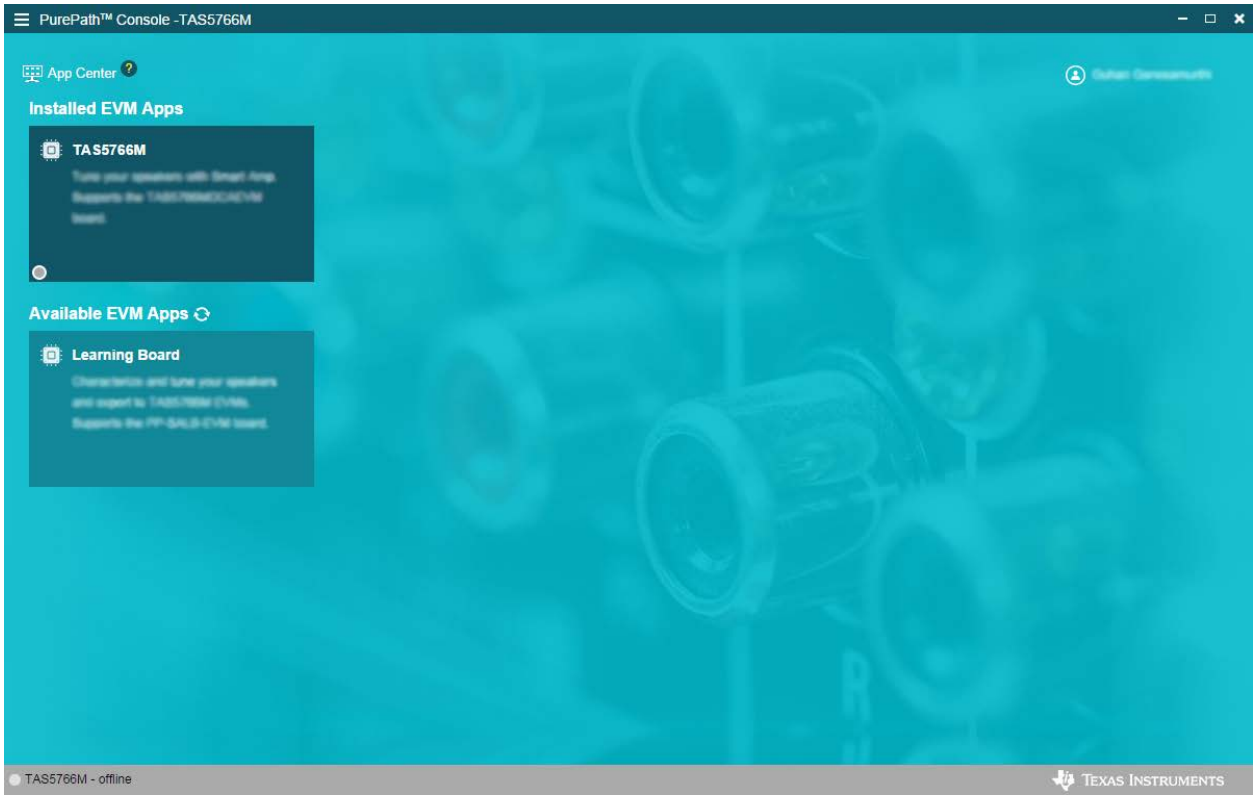

Figure 1. App Center Overview

2.1 Sign in Button

Sign into the PPC3 App GUI by selecting the ‘Sign In’ button in the App center page and entering TI account details in the resulting pop up, as shown in Figure 2.

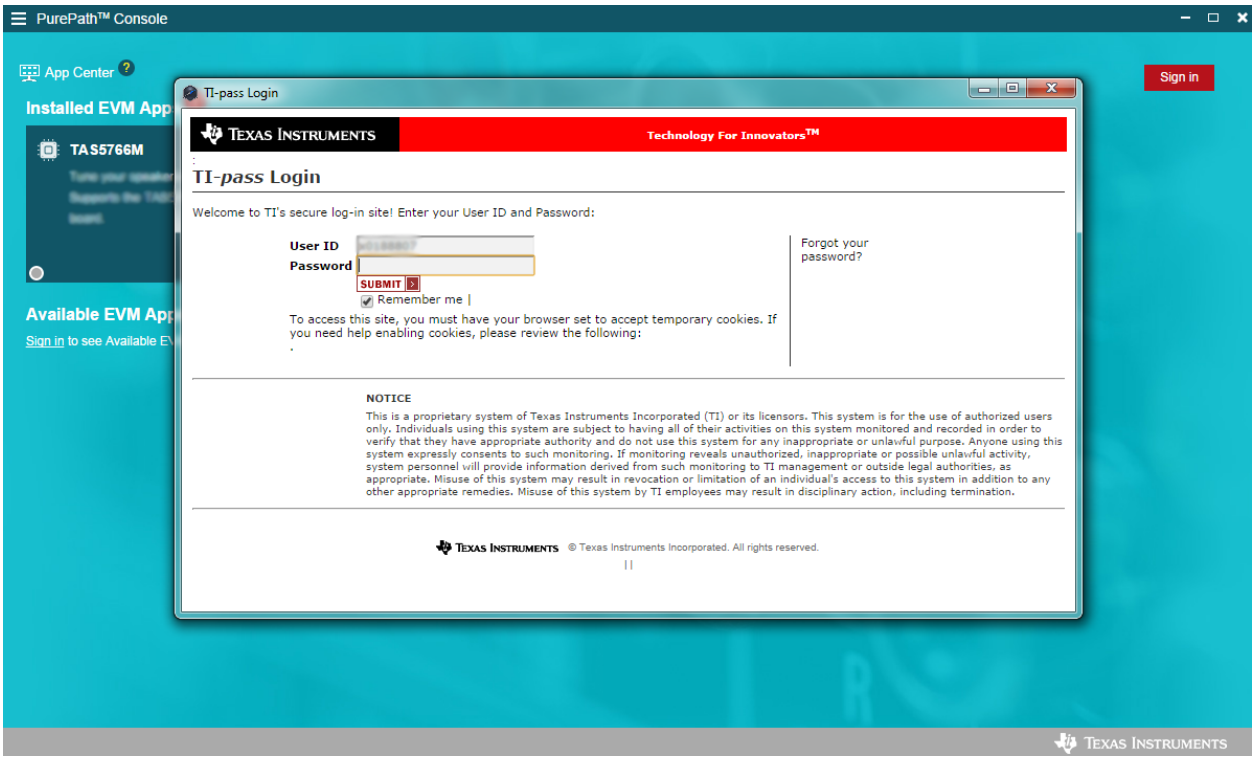

Figure 2. Sign in

2.2 Installed EVM Apps

The **Installed EVM Apps** section displays the list of EVM Apps installed in the PC. Clicking on an EVM App in this section directs to the EVM *Home Page*. If a device is auto-detected by PPC3, the circle on the bottom left of the detected EVM App is filled with green color. Clicking on the 'x' button on the bottom right corner of the App, which appears while hovering over the EVM App (as shown in Figure 3), launches a pop up asking for 'uninstall app' confirmation. Upon confirmation, that EVM App is uninstalled.

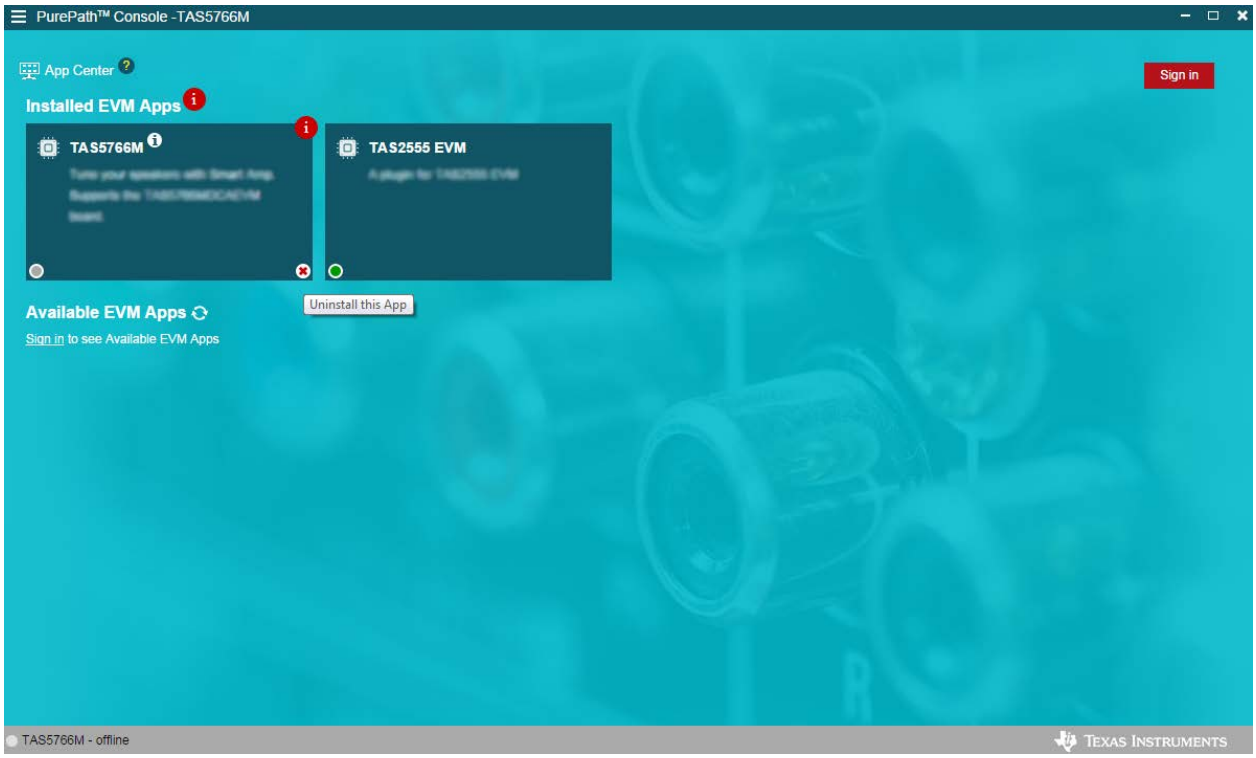

Figure 3. Uninstall App from App Center

2.3 Available EVM Apps

The available EVM Apps section has the list of available EVM Apps. Clicking on the EVM App in this section launches the install EVM App popup. Selecting 'Install' in the pop up installs the selected app, as shown in Figure 4.

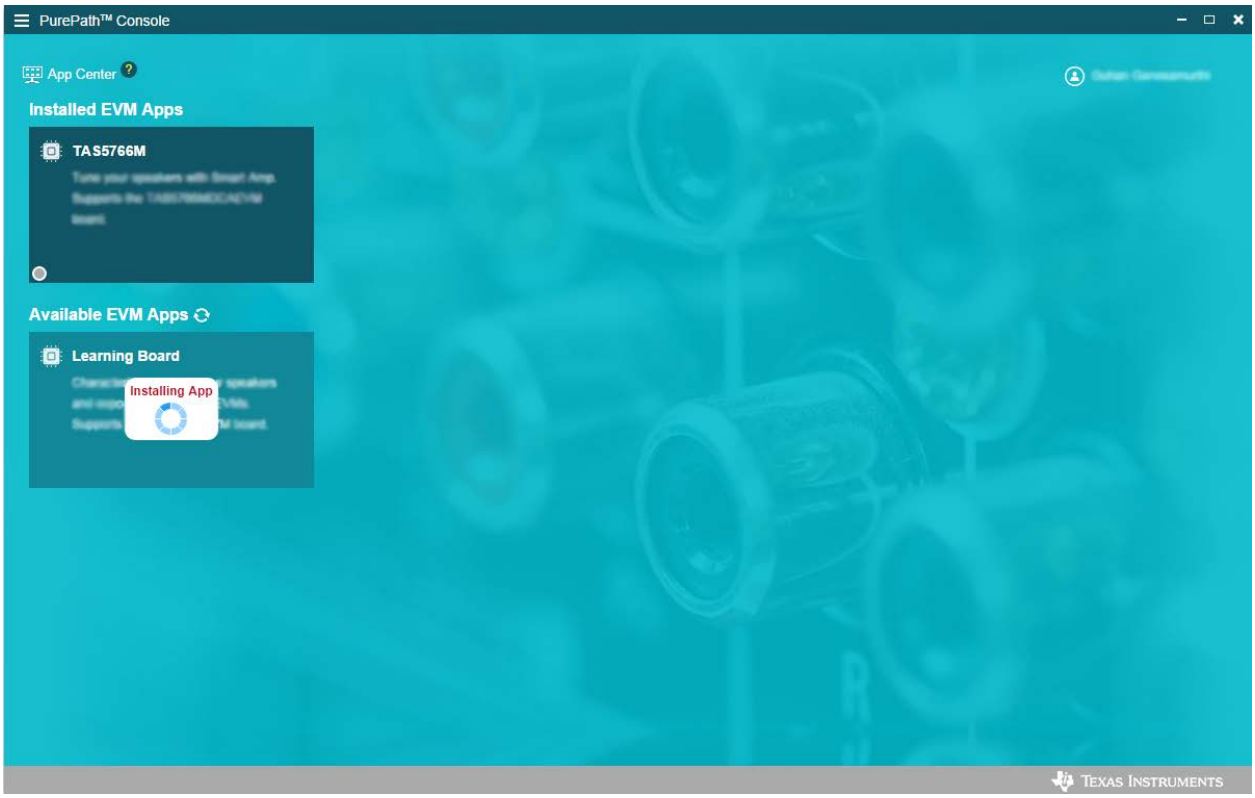

Figure 4. Available EVM Apps

2.4 App Update

Installed EVM Apps can be updated if a new version is available. Update availability is indicated by the information icon (‘i’) on the top right corner of each installed EVM app, as shown in Figure 5. When the icon is clicked, a pop up is launched asking for confirmation. Upon confirmation, the app will be updated to the latest version.

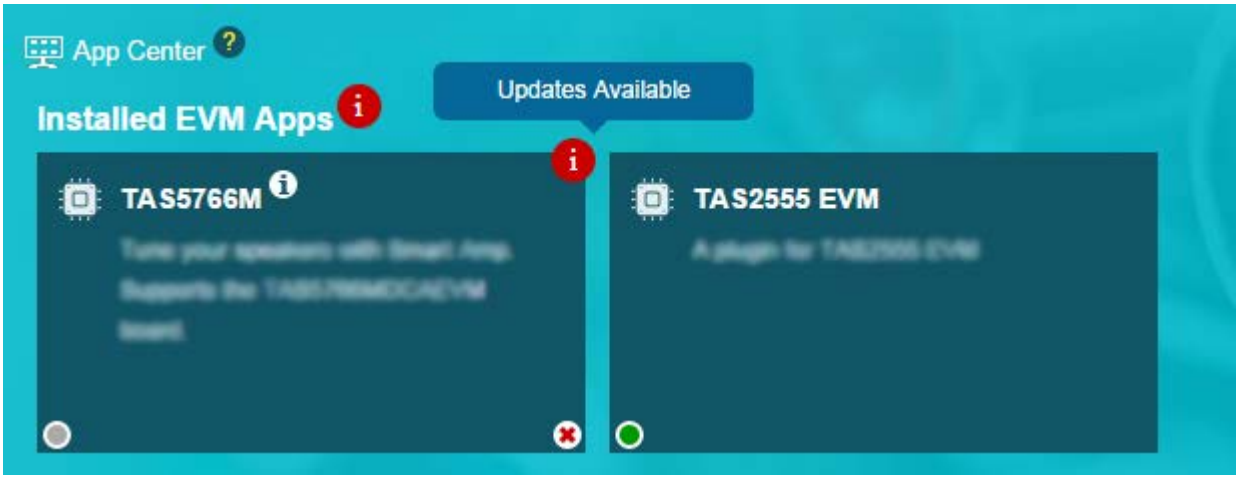

Figure 5. App Update

2.5 Platform Update

The PPC3 Platform can be updated if a new version is available. Update availability is indicated by the information icon (‘i’) next to Installed EVM Apps as shown in Figure 6. When the icon is clicked, the ‘Platform Update’ pop up requests confirmation to download the update. Figure 7 shows the download status. Once the download is completed, the user is prompted to save the current work and restart the app.

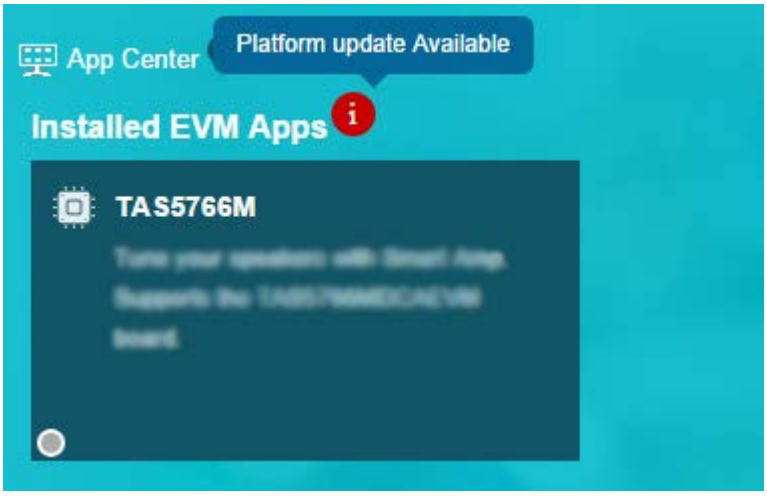

Figure 6. Platform Update Availability

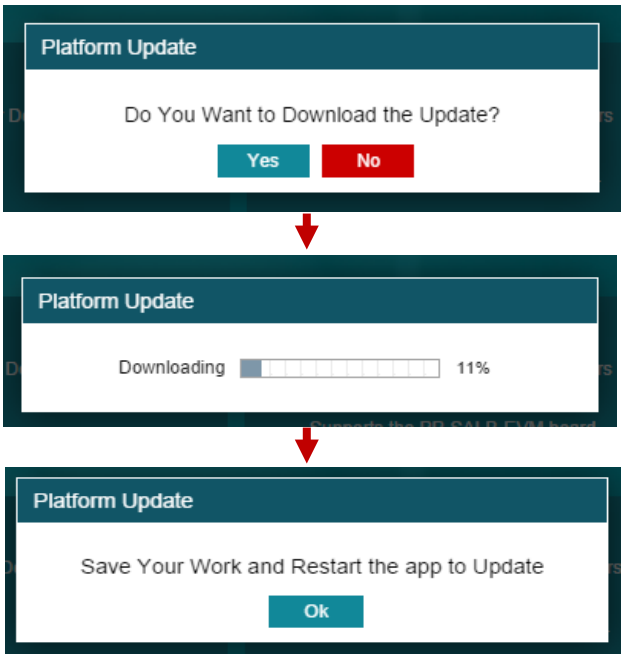

Figure 7. Platform Update Pop up

3 Device Home Page

The Device Home Page displays features that are available for that EVM. When a feature is selected, then the respective page is loaded. For instance, selecting System Calibration takes you to the System Calibration page where the EVM setup is calibrated. An example device home page is shown in Figure 8.

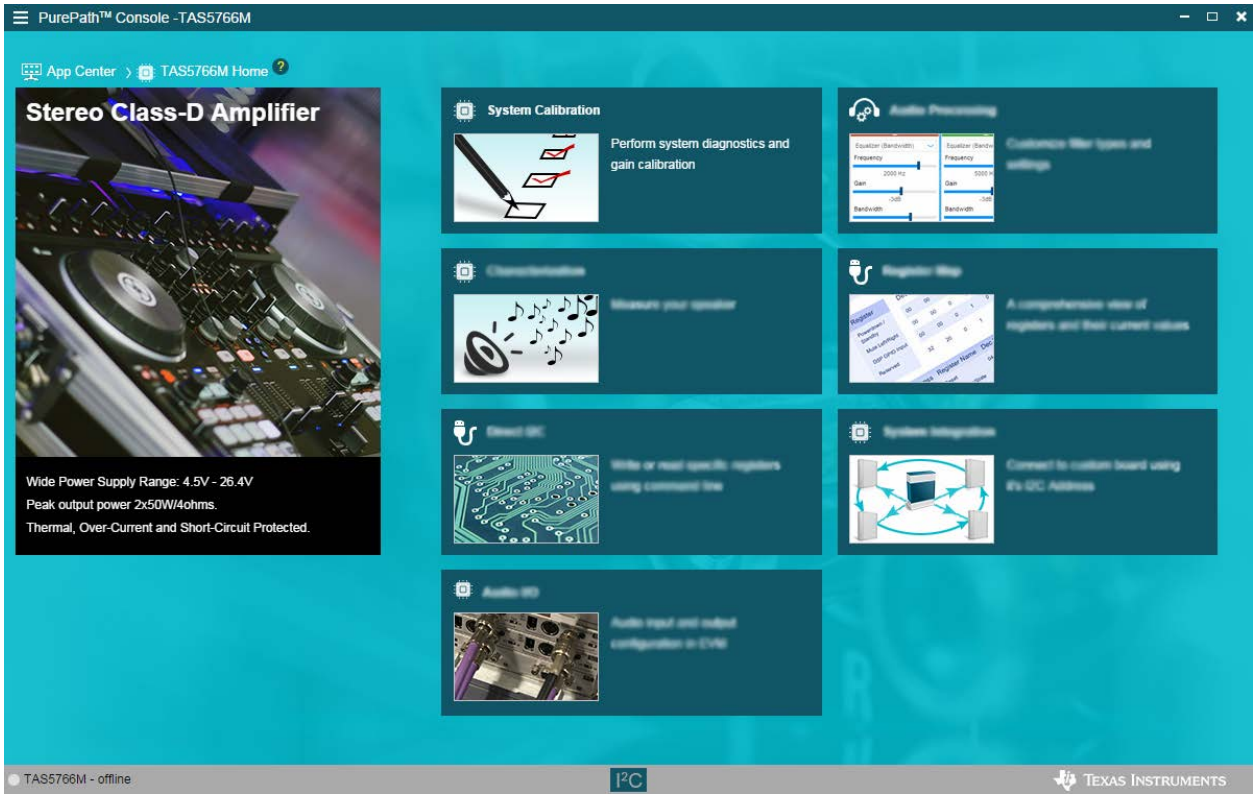

Figure 8. Device Home Page

4 App Walkthrough and Device Status

4.1 App Walkthrough

Use the App Walkthrough by clicking on the question mark icon available next to the page name in all pages, as shown in Figure 9.

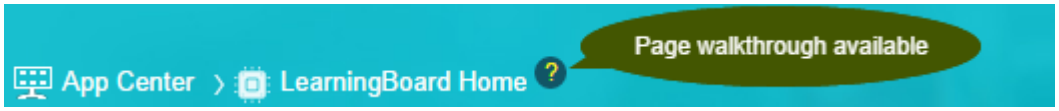

Figure 9. App Walkthrough Icon

The App Walkthrough feature guides the user through various sections in the selected page and displays the comments, explaining the sections as shown in Figure 10. Close the App Walkthrough at any time by clicking on the 'x' mark at the top right corner of the pop up. Clicking on the left and right arrow will guide through previous and next sections in the page.

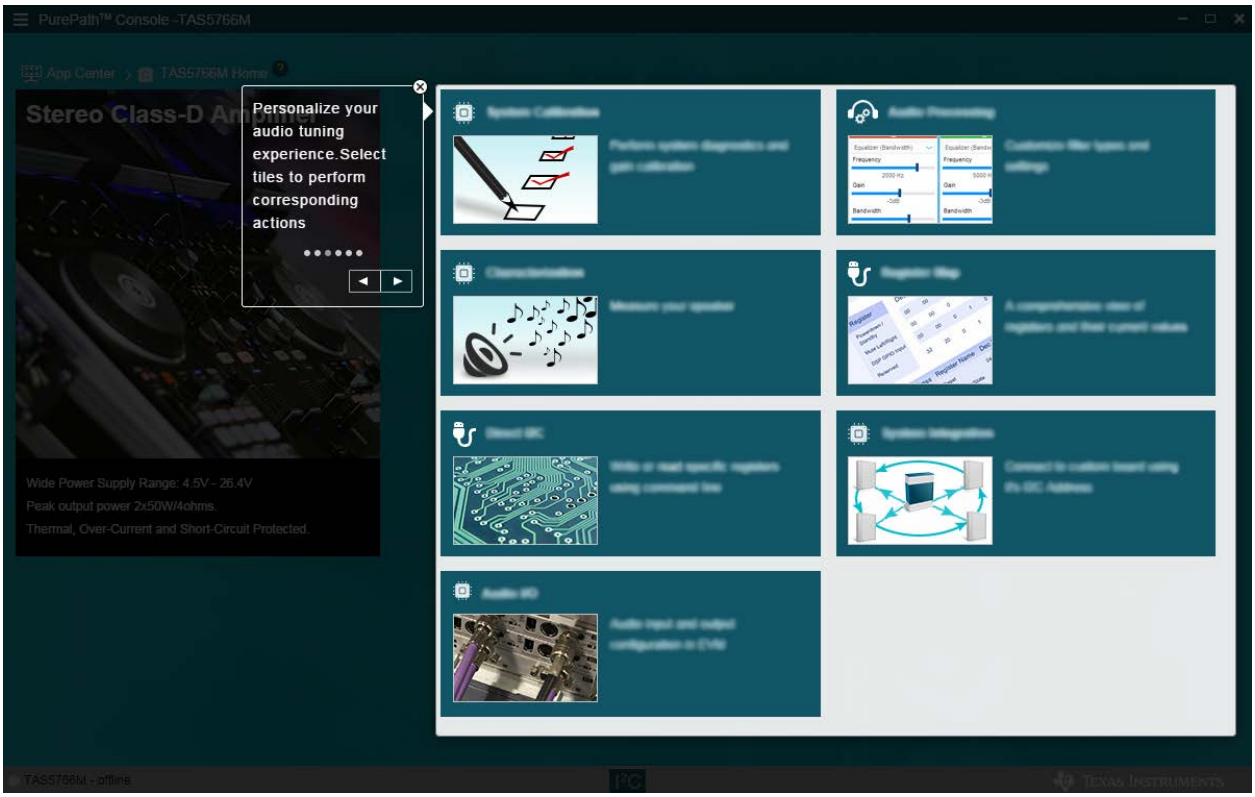

Figure 10. App Walkthrough Demo

4.2 Device Status

Device Status is used to indicate the current connection status of the EVM. This status can be viewed at the bottom left corner of the PPC3 App as highlighted in Figure 11. ‘Online’ represents that the EVM is connected and detected by PPC3 App.

Figure 11. Device Status

4.3 Manual Connect to Device

If the device is not automatically detected by the PPC App GUI, connect to the device manually. This is done by connecting the device to the PC and selecting the connected device’s app from the App Center page. In the device home page, click on the ‘Connect’ button, right next to connection status indication area in the status bar.

Disconnect the device by selecting ‘Disconnect’, which appears when the device is connected.

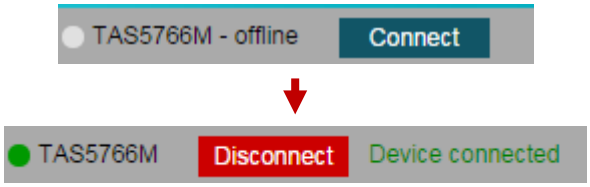

Figure 12. Manual Connect to Device

5 I2C Monitor

Use the I2C Monitor to view the I2C transactions happening as a result of device operation performed using the PPC3 app. The I2C Monitor is also used to execute direct read and write commands.

Access the I2C Monitor by clicking the I2C Monitor icon in the bottom bar, in any of the pages. Hovering over the I2C Monitor icon will pop up a window as shown in Figure 13.

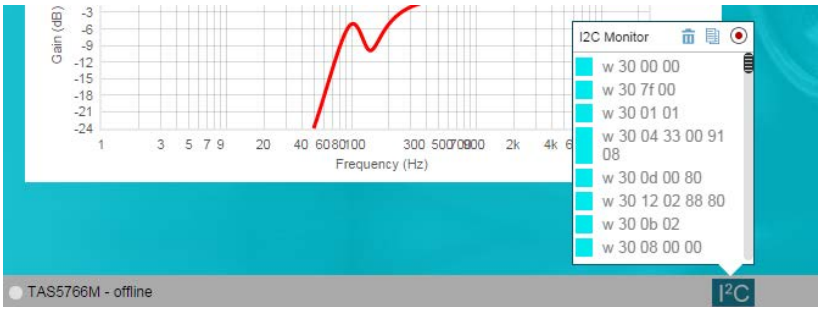

Figure 13. I2C Monitor

Clicking on the I2C Monitor Icon launches the expanded floating window view, as show in Figure 14.

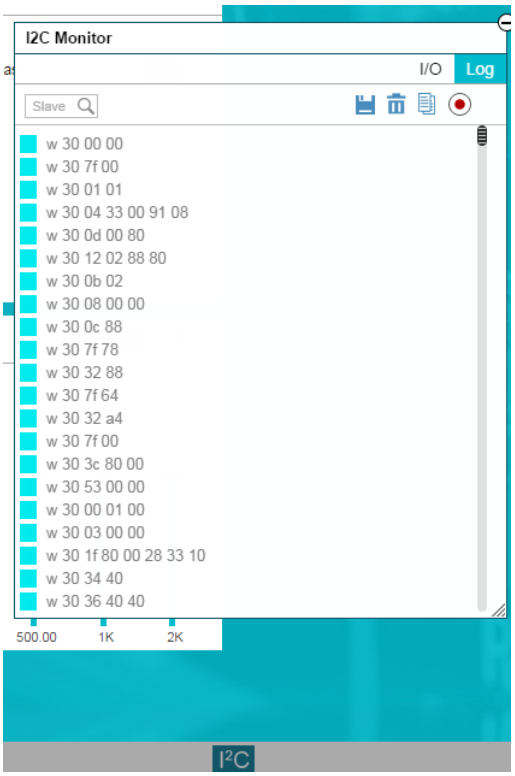

Figure 14. I2C Monitor Expanded View

I2C Monitor has two main tabs, I/O and Log, present in the top right corner of the I2C Monitor window. The following sections explain these features in detail.

5.1 I/O Tab

The I/O tab in the I2C Monitor has two sub sections. The **Input** section has the provision to enter the read or write commands scripts. Clicking the **Execute** button will execute the commands written in the Input section. The status of the execution is displayed in the **Output** section as shown in Figure 15.

Refer to Appendix A for details on how to write scripts.

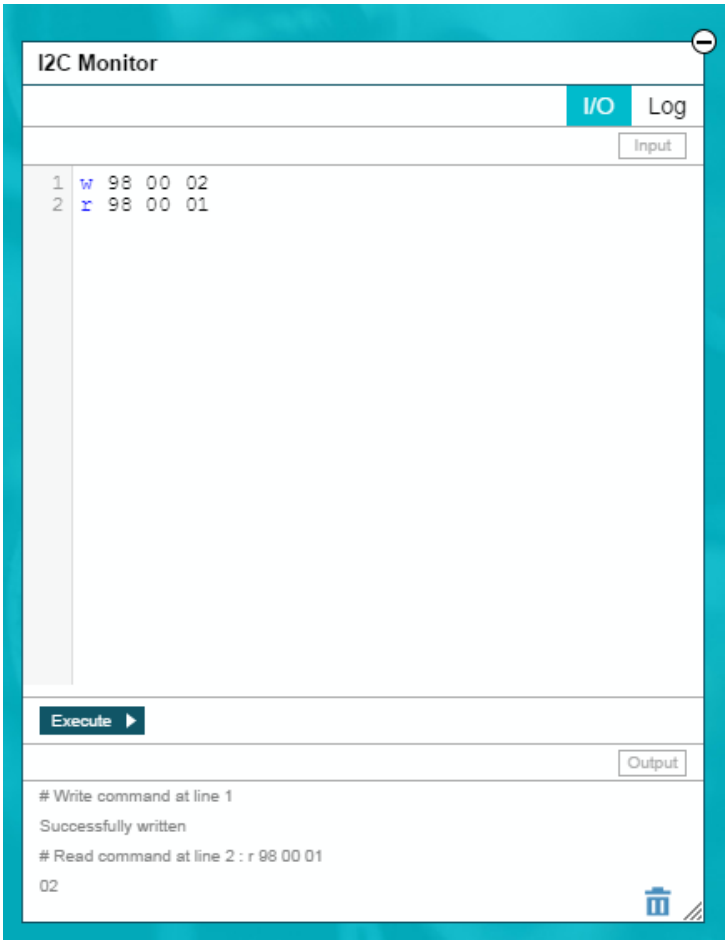

Figure 15. I2C Monitor I/O Section

5.2 Log Tab

The Log tab in the I2C Monitor displays the I2C command history, if the record option is enabled. The log tab has a search option to search for a particular command. The search key can be found at the top left of the window with the search icon. ‘Save to a file’ can be used to save the log as a .Config file to the PC. ‘Delete Output’ clears the log history. ‘Copy to a Clipboard’ copies the log text to the clipboard. Clicking the ‘Start Recording’ button starts recording the I2C transactions and displays them in the log window, ‘Stop Recording’ stops recording I2C transactions.

6 Audio Player, Tuning History Navigator, and Tuning Snapshot

6.1 Audio Player

The Audio Player can be used to play audio tracks available in the PC and perform various tuning operations. The ‘Audio Player’ and ‘Tuning Snapshot’ shortcut are positioned at the top of the PPC3. Clicking on the Audio Player launches the floating window, as shown in Figure 16.

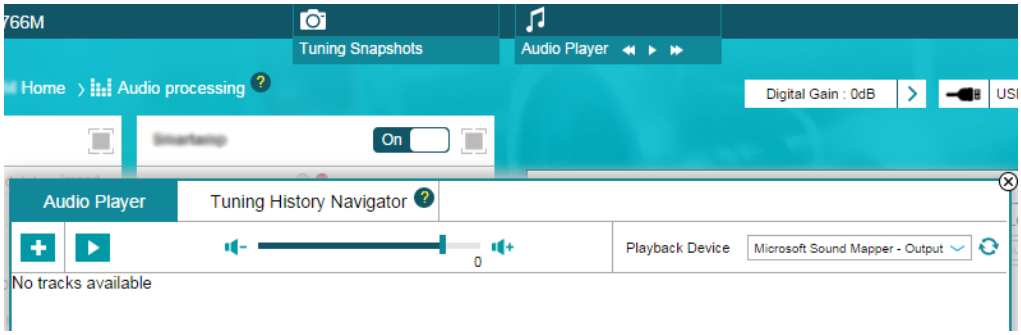

Figure 16. Audio Player Window

6.1.1 Tracks

Audio Tracks can be added to the Player by selecting the ‘+’ icon in the top left of the audio player. Selecting the audio file adds it to the track list. Clicking on the particular track plays the track. Audio Player with tracks added is shown in Figure 17. Clicking on the repeat icon in the track list repeats the particular track. Clicking on the ‘x’ icon at the right most of the each track removes that track from the list. The Play/Pause button, next to add track in the Audio Player, can be used to play/pause the current track. Playing/Pausing, Next Track, and Previous Track options are also available in the Audio Player shortcut at the top center which can be used to go through the list of tracks when the Audio Player is not launched as a window.

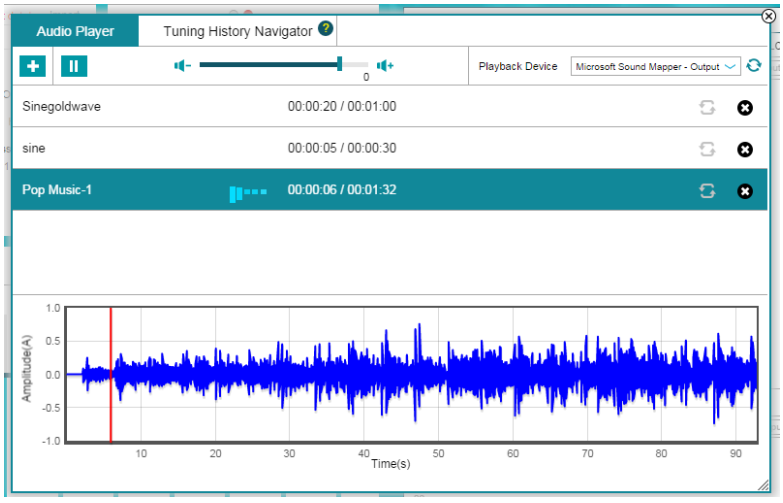

Figure 17. Audio Player with Tracks Added

6.1.2 Waveform Display

When a track is added to the Audio Player list, its corresponding waveform view with Amplitude (A) on Y-axis and Time(s) on X-axis will be displayed as shown in Figure 17.

6.1.3 Volume Control

The volume control slider in the Audio player can be used to increase/decrease the volume played in the selected playback device.

6.1.4 Playback Device

Different Playback devices can be selected from the Playback Device dropdown list box at the top right of the Audio Player, as highlighted in Figure 18. Audio file formats that can be played using the Audio player feature are .wav and .mp3.

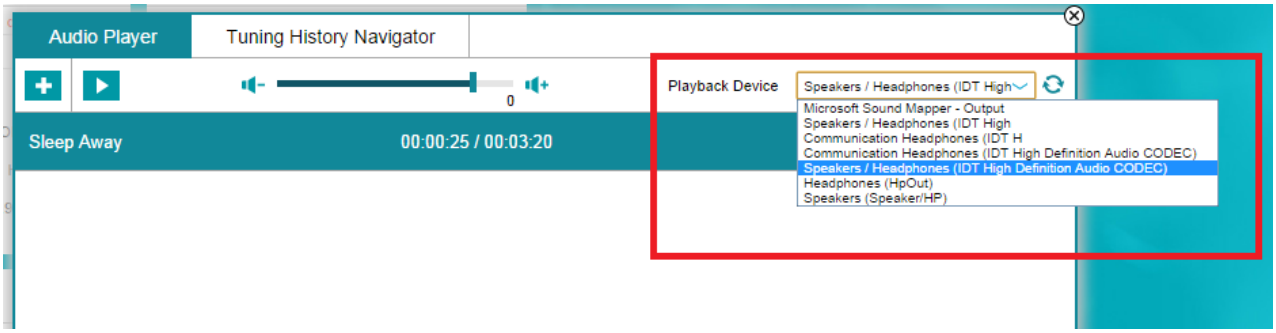

Figure 18. Audio Player Playback Device

6.2 Tuning History Navigator

Tuning History Navigator saves the history of the tuning changes with each node corresponding to a single tune setting changed from the previous node. Clicking on a node navigates to that setting in the GUI and applies that setting to the device. A new branch is created when an existing node is selected and from there, new tuning changes are performed. Hovering over the node displays the tuning change as a result of which the node was created. The following operations can be performed with the Tuning History Navigator:

1. Prune
2. Fit
3. Reset
4. Zoom

A Screenshot of the Tuning History Navigator is shown in Figure 19.

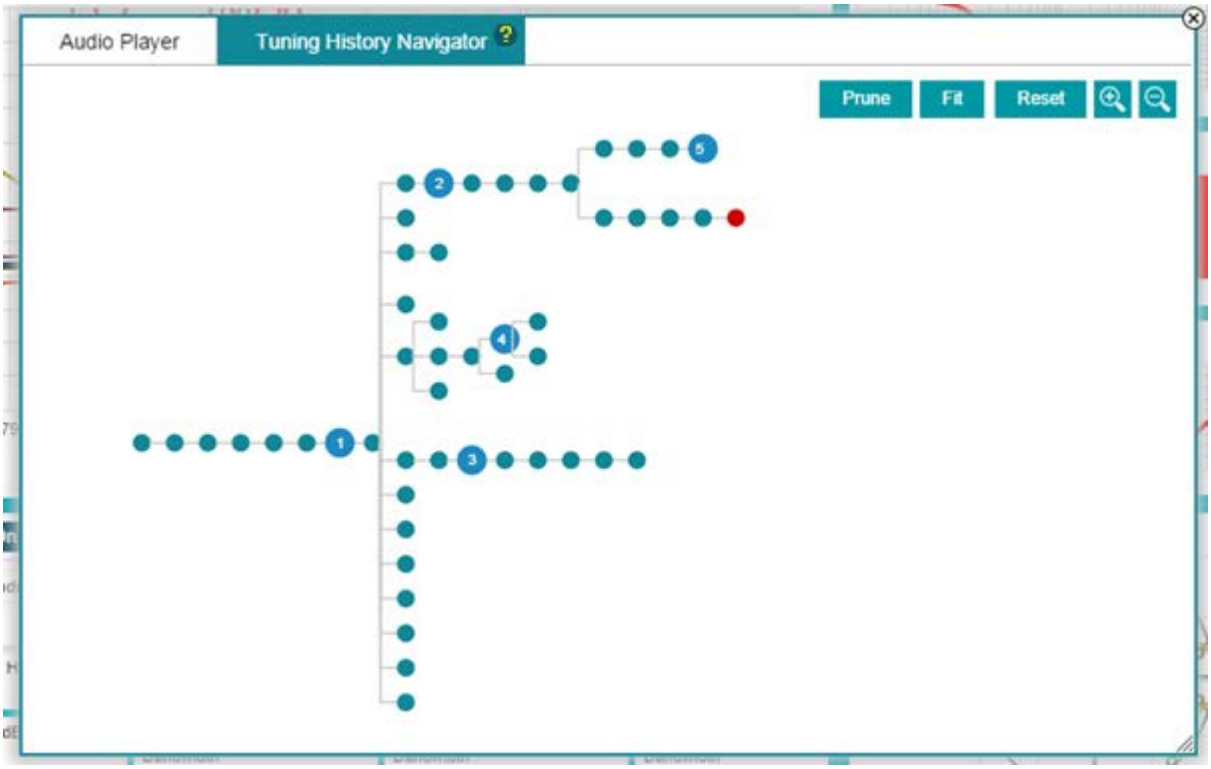

Figure 19. Tuning History Navigator

The Tuning History Navigator operations are explained in the following sections.

6.2.1 Prune

Use the Prune Operation to remove intermediate nodes other than Snapshot nodes and current node. Use this operation if there are lots of intermediate node history data that are no longer required. Figure 20 shows the result of the Prune Operation on the nodes from Figure 19.

Note: Once pruned, the removed nodes cannot be recovered and only snapshot nodes are present in the tree.

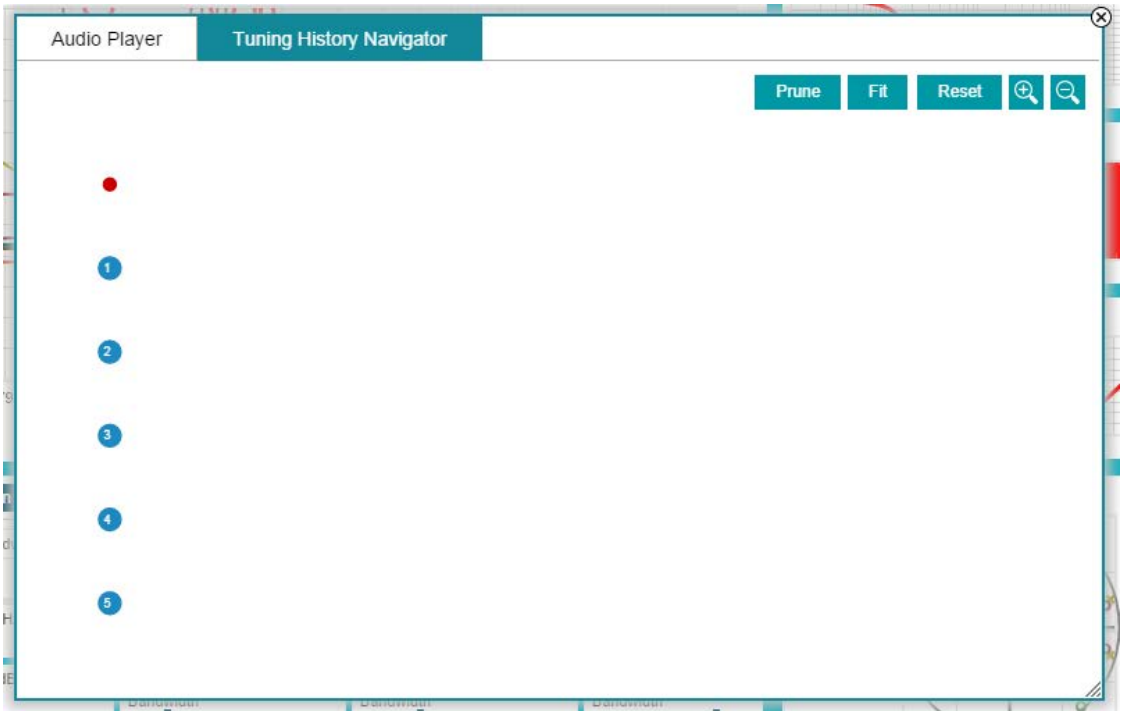

Figure 20. Tuning History Navigator Prune Function

6.2.2 Fit

Use the Fit Operation to fit the entire tree to the Tuning History Navigator section. This operation can be used if the tree size has grown more than the size of the Tuning History Navigator section. Clicking this button fits the tree to the screen with all nodes visible, as shown in Figure 21.

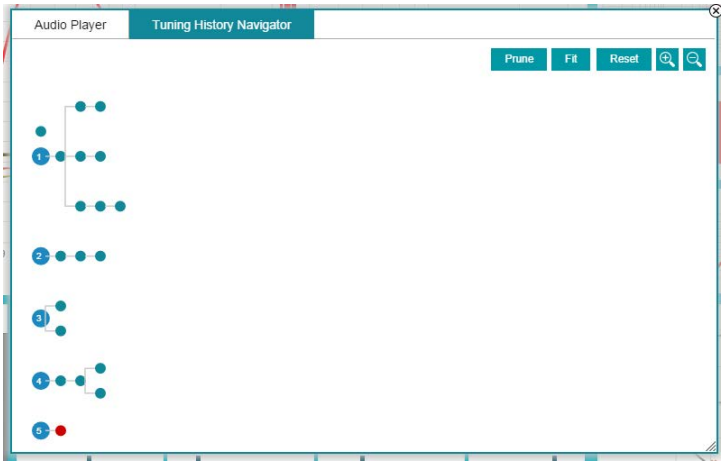

Figure 21. Tuning History Navigator Fit Function

6.2.3 Reset

This option resets the graph to default zoom level.

6.2.4 Zoom

Use the Zoom Operator to Zoom In/Out of the Tree. This operation can be used to navigate to the particular section of history, if the history tree is very long. Figure 22 shows the Zoomed Out History Tree.

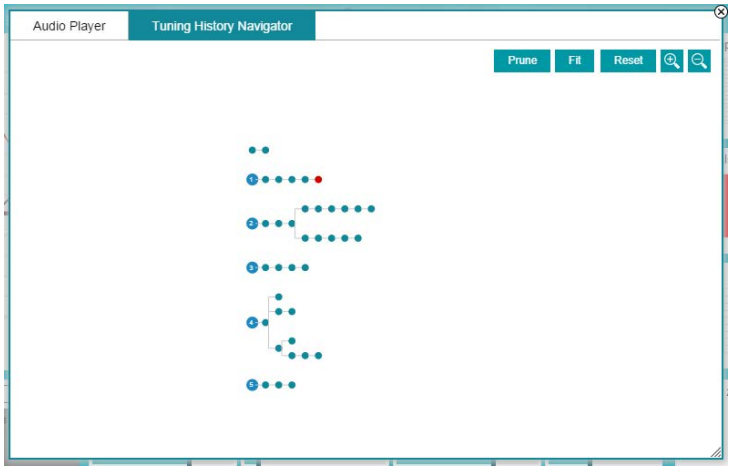

Figure 22. Tuning History Navigator Zoom Function

6.3 Tuning Snapshot

Tuning History Navigator creates a new node for any single tuning change, but Tuning Snapshot takes all the values of the current setting and stores it. A maximum of 5 Snapshots can be stored. Tuning Snapshot is found to the left of the Audio Player as shown in Figure 23. Clicking on the Respective Snapshot number applies the settings stored for that snapshot.

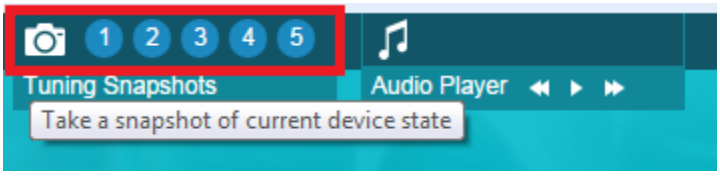

Figure 23. Tuning Snapshot

6.3.1 Take Snapshot

Clicking on the camera icon, as highlighted in Figure 23, takes a snapshot of the current setting. If the snapshot of the current setting is already available, a status message is displayed with the text containing the current tuning snapshot is already available.

6.3.2 Delete Snapshot

Any snapshot taken already can be deleted by hovering over the snapshot number and clicking the delete button from the window shown below the snapshot number, as highlighted in Figure 23.

6.3.3 Retake Snapshot

Any Snapshot saved already can be overwritten with the current tuning by hovering over the respective snapshot number and clicking the camera icon at the bottom right corner from the window shown below the snapshot number, as highlighted in Figure 24.

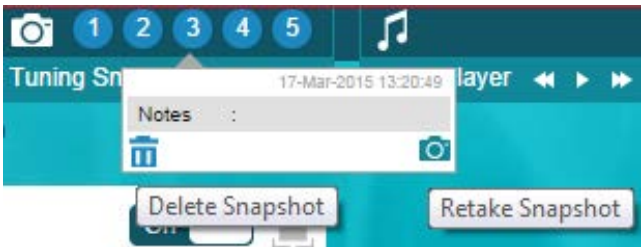

Figure 24. Retake/Delete Snapshot

6.3.4 Notes for Snapshots

Each snapshot can be given custom notes, so that the snapshots can be identified based on the notes. Notes can be entered for each snapshot by clicking on the pen icon in the window, which appears while hovering over the snapshot number, as shown in Figure 25.

Figure 25. Edit Notes

7 Save / Import

The Save/Import feature is used to store a tuning, other high level settings of the entire EVM to a file (.ppc3), and load the tuning and high level settings from the configuration file to the EVM and GUI.

'Save' and 'Save As' options are available under the app menu, displayed when hovering over the menu button to the left of the App Name at the top bar, as shown in Figure 26. Selecting 'Save' for the first time in a session, asks for a .ppc3 file to be selected and the current project settings are saved to the selected file. Subsequent 'Save' selections will overwrite the initially selected file.

Also, when the app is closed, a pop up asks if the current project settings should be saved to a file, as shown in Figure 27.

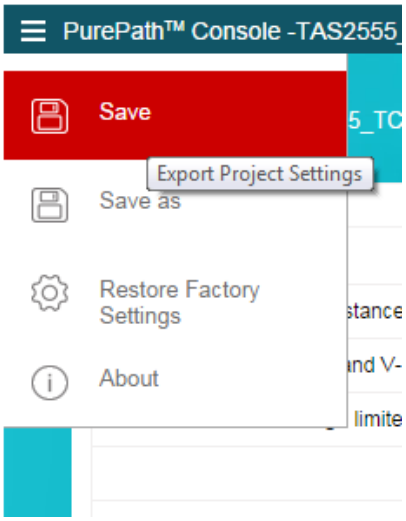

Figure 26. Save/Save As Menu Option.

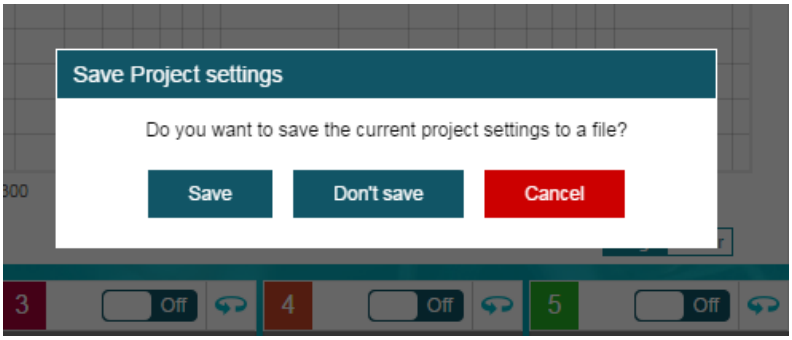

Figure 27. Save Project Settings Pop-up

Settings saved to a file can be imported in two ways. When device home page is loaded for the first time in the session, a pop up appears as shown in Figure 28. Selecting 'Open' will ask for .ppc3 file and project settings saved in the chosen file will be applied to the GUI and the device (register values are calculated based on the settings and written to the device). Also, the recently saved .ppc3 files are listed in the 'Recent' section and can be easily selected.

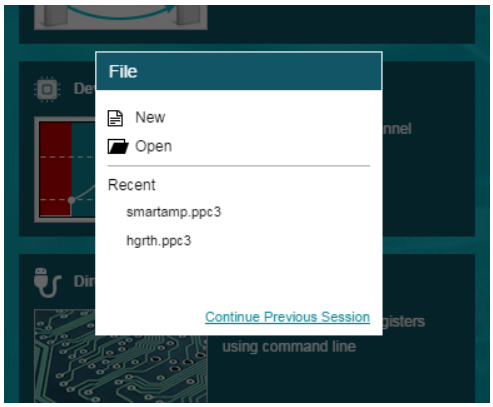

Figure 28. File Select Pop-up

Selecting 'New' will set the GUI at default state, whereas selecting 'Continue Previous Session' at the bottom on the pop up, will restore the GUI at the state with which it was closed during the earlier session.

Settings can also be imported by selecting 'Import' button in the Audio processing page as shown in Figure 29. When the desired .ppc3 is selected, a pop up lists the features that are present in the selected file as shown in Figure 30. Any of those features can be selected and it will be applied to the GUI.

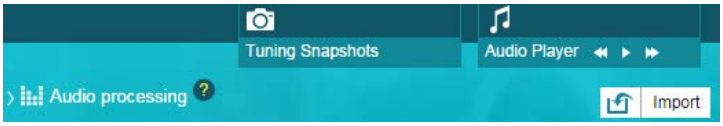

Figure 29. 'Import' button in Audio Processing page

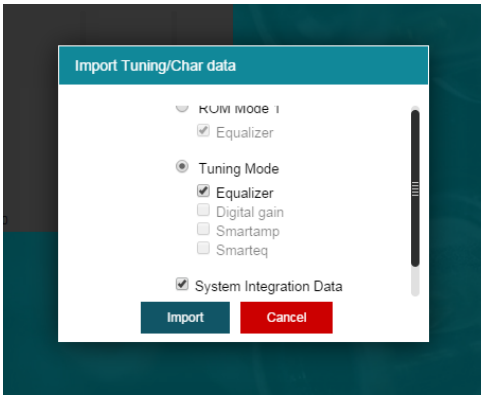

Figure 30. Import Tuning/Char Data Pop-up

Appendix A. Configuration Script (CFG) Specification 3.0

A configuration script is used to write and read data through I^{2}C, SPI as well as to instruct the interpreter to perform simple tasks such as delays and breakpoints. It has a .cfg file extension.

The scripting language is quite simple. Each line in a script file is one command. A line is terminated by a line feed (*0x0a*) or a carriage return and line feed (*0x0d0a*). Characters can be either upper or lower case as all are ultimately converted to lower case (except for the string text in the Break command). Whitespace between commands in a line is accepted.

The first character of a line is the command. The commands are:

```
# Comment
i Interface Select
w Write
> Write Continuation (extend writes below a w)
r Read
f Wait for Flag
m Masked Write
b Breakpoint
d Delay
t Test and Skip
```

Comment

A comment can be at the beginning of a line or after a command as shown in the following example. Anything after the '#' symbol is ignored.

Example:

```
# This is a comment
w 30 00 00#in-line comment
```

Interface Select i

This command selects between I^{2}C, SPI, and GPIO interfaces.

Example:

```
i i2cfast
w 30 00 AA # First byte is I2C Slave Address, second byte is register offset
i spi8
w 00 01 55 # First byte is ignored in SPI mode
i gpio
w 00 00 01 # First byte is ignored, second byte selects GPIO mode
```

Write w / >

These commands instruct the parser to write data to the selected interface.

Example:

```
i i2cfast
w 30 00 2c    # Page 44
w 30 08 55 AA # Write 55 AA 55 00 AA starting at register 8
# A comment can also be here
> 55 00 AA
```

Read r

This command instructs the parser to read data from the selected interface.

Example:

```
i i2cfast
w 30 00 2c    # Page 44
r 30 08 20    # Read 32 bytes starting at register 8
```

Wait for Flag f (currently unsupported in PPC3)

This command reads the specified register to check if it matches a bit field.

Example:

```
i i2cfast
w 30 00 2c                # Page 44
f 30 01 xxxxxxx0          # Wait for bit D0 to clear
f 30 01 xxxxxxx0 100      # Wait for bit D0 to clear, 100 retries
f 30 01 xxxxxxx0 20 100   # Wait for bit D0 to clear, 20ms delay, 100 retries
```

Masked Write m (currently unsupported in PPC3)

This command modifies the contents of a register based on a mask.

Example:

```
i i2cfast
w 30 00 2c    # Page 44
m 30 08 01 01 # Set bit D0 = 1. First byte after 08 is data, second byte is mask.
```

Breakpoint b

This command stops execution and displays a string.

Example:

```
i i2cfast
w 30 00 2c    # Page 44
b “Execution Halted. Press OK to continue”
w 30 00 00    # Page 0
```

Delay d

This command instructs the parser to stop execution for the specified time (in ms).

Example:

```
i i2cfast
w 30 00 2c    # Page 44
d 1500        # Delay for 1.5 seconds
w 30 00 00    # Page 0
```

Test and Skip t (currently unsupported in PPC3)

This command instructs the parser to skip execution for the specified number of commands. Comments (#) and (>) are not counted as commands.

Example:

```
i i2cfast
t 30 0001     # Check if device with slave address 0x30 is present, skip by 1 if not.
d 1500        # Delay for 1.5 seconds. This line is skipped if 0x30 is not present.
w 98 00 00    # Write to device with I2C Slave Address = 0x98.
```

Command Syntax

The syntax for some commands might differ from one interface to the other. Table A-1 provides the syntax for commands that are compatible with all interfaces.

Table A-1. CFG Script Syntax for All Interfaces

Command	Name	Syntax
#	Comment	[*text*]#[*text*]
i	Interface Select	**i** {**i2cstd**|**i2c**|**i2cfast**|**spi**|**spi8**|**spi16**|**gpio**} **i2cstd** — I²C 100kHz **i2cstd16** — I²C 100kHz, 16-bit register addressing **i2c, i2cfast** — I²C 400kHz (default) **i2c16, i2cfast16** — I²C 400kHz, 16-bit register addressing **spi, spi8** — SPI with 8-bit register addressing **spi16** — SPI with 16-bit register addressing **gpio** — USB controller GPIO
b	Breakpoint	**b** [*"text"*] [*"text"*] — Text within quotes (string)
d	Delay	**d** <*delay_ms*> <*delay_ms*> — Delay (decimal, 0-32767)

The syntax specific to I²C interfaces is shown in Table A-2.

Table A-2. CFG Script Syntax for I²C Interfaces

Command	Name	Syntax
w >	Write Write Continuation	w <slave_addr> <reg_off> <data ...> [> <data ...>] [> <data ...>] ... <slave_addr> I²C Slave Address (hex, 0-FF), <reg_off> Register offset (hex, 0-FF) <data ...> Data separated by whitespace (hex, 00-FF)
r	Read	r <slave_addr> <reg_off> <count> <slave_addr> I²C Slave Address (hex, 0-FF) <reg_off> Register offset (hex, 0-FF) <count> Number of registers to read (hex, 0-FF)
f	Wait for Flag	f <slave_addr> <reg_off> <bitfield> f <slave_addr> <reg_off> <bitfield> [count] f <slave_addr> <reg_off> <bitfield> [delay_ms] [count] <slave_addr> I²C Slave Address (hex, 0-FF) <reg_off> Register offset (hex, 0-FF) <bitfield> 8-bit character field of {**0**|**1**|**x**}, "x" is don't care [delay_ms] Delay between reads. Default = 0 (decimal, 0-32767) [count] Number of retries. Default = 255 (decimal, 0-32767)
m	Masked Write	m <slave_addr> <reg_off> <data> <mask> <slave_addr> I²C Slave Address (hex, 0-FF) <reg_off> Register offset (hex, 0-FF) <data> Data (hex, 0-FF) <mask> Data mask (hex, 0-FF)
t	Test and Skip	t <slave_addr> <count> <slave_addr> I²C Slave Address (hex, 0-FF) <count> Number of commands to skip (hex, 0-FFFF)

NOTE: For 16-bit mode, <*reg_off*> is a string of two bytes (for example, 00 01).

The syntax specific to SPI interfaces is shown in Table A-3.

Table A-3. CFG Script Syntax for SPI Interfaces

Command	Name	Syntax
w >	Write Write Continuation	w <*ignored*> <*reg_off*> <*data ...*> [> <*data ...*>] [> <*data ...*>] ... <*ignored*> Ignored, write 00 <*reg_off*> Register offset (hex, 0-FF) <*data ...*> Data separated by whitespace (hex, 0-FF)
r	Read	r <*ignored*> <*reg_off*> <*count*> <*ignored*> Ignored, write 00 <*reg_off*> Register offset (hex, 0-FF) <*count*> Number of registers to read (hex, 0-FF)
f	Wait for Flag	f <*ignored*> <*reg_off*> <*bitfield*> f <*ignored*> <*reg_off*> <*bitfield*> [*count*] f <*ignored*> <*reg_off*> <*bitfield*> [*delay_ms*] [*count*] <*ignored*> Ignored, write 00 <*reg_off*> Register offset (hex, 0-FF) <*bitfield*> 8-bit character field of {**0**|**1**|**x**}, "x" is don't care [*delay_ms*] Delay between reads. Default = 0 (decimal, 0-32767) [*count*] Number of retries. Default = 255 (decimal, 0-32767)
m	Masked Write	m <*ignored*> <*reg_off*> <*data*> <*mask*> <*ignored*> Ignored, write 00 <*reg_off*> Register offset (hex, 0-FF) <*data*> Data (hex, 0-FF) <*mask*> Data mask (hex, 0-FF)
t	Test and Skip	t <*ignored*> <*count*> <*ignored*> Ignored, write 00 <*count*> Number of commands to skip (hex, 0-FFFF)

NOTE: For 16-bit mode, <*reg_off*> is a string of two bytes (for example, 00 01).

The syntax specific to the GPIO interface is shown in Table A-4.

Table A-4. CFG Script Syntax for the GPIO Interface

Command	Name	Syntax
w	Write	w <*ignored*> <*command*> <*data*> <*ignored*> Ignored, write 00 <*command*> 00: Bit Set 01: Bit Clear 02: Write 04: Set Bank <*data*> Data (hex, 0-FF)
r	Read	r <*ignored*> <*command*> <**01**> <*ignored*> Ignored, write 00 <*command*> 03: Read 05: Get Bank
f	Wait for Flag	f <*ignored*> <*ignored*> <*bitfield*> f <*ignored*> <*ignored*> <*bitfield*> [*count*] f <*ignored*> <*ignored*> <*bitfield*> [*delay_ms*] [*count*] <*ignored*> Ignored, write 00 <*bitfield*> 8-bit character field of {**0**|**1**|**x**}, "x" is don't care [*delay_ms*] Delay between reads. Default = 0 (decimal, 0-32767) [*count*] Number of retries. Default = 255 (decimal, 0-32767)

Table A-5 summarizes the GPIO pin assignments for the TAS1020B USB controller. USB firmware versions 3.50 and newer boot with Bank 1 selected as default. Older versions boot with Bank 0 as default.

Table A-5. TAS1020B GPIO Interface Assignments

Bank	D7	D6	D5	D4	D3	D2	D1	D0
0	x	P3.5	P3.4	P3.3	P1.3	P1.2	P1.1	P1.0
1	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
2	x	x	P3.5	P3.4	P3.3	P3.2	P3.1	P3.0

Appendix B. Microcontroller Code Implementation

PurePath™ Console 3 is able to generate its output, a compiled process flow, in the form of a 2-dimensional C array suitable for inclusion into a program running on the application processor or microcontroller. Such arrays specify a list of individual 8-bit register writes to the amplifier. Additionally, the format allows one to specify delay (meaning, the application needs to pause for a specified time at that point) and also includes a mechanism to cause the application switch between such arrays.

The purpose is to specify this data structure, extend it in a backwards-compatible manner for burst writes (that is, register writes with a payload of greater than one byte) and provide some example application to code to illustrate its use.

Header File

The header file is generated based on a Dump Definition File (.ddf), as explained in Appendix C.

The output header file will always contain these definitions:

```
typedef unsigned char cfg_u8;
typedef union {
    struct {
        cfg_u8 offset;
        cfg_u8 value;
    };
    struct {
        cfg_u8 command;
        cfg_u8 param;
    };
} cfg_reg;

#define CFG_META_SWITCH (255)
#define CFG_META_DELAY  (254)
#define CFG_META_BURST  (253)
```

Then, the header will contain the following definition for registers[]:

```
cfg_reg registers[] = {

… registers here

};
```

The definitions of individual register writes are tuples of *{register, value}*.

In the case of a burst write, a *Burst* meta command *{CFG_META_BURST, length}* is appended followed by tuples with 2 values in each. Note that length takes into account both the register and data bytes. Odd number of bytes are padded with an extra byte.

```
cfg_reg registers[] = {
    { offset, value },
    { offset, value },
    ...
    { CFG_META_BURST, length},
    { offset, value },  < Begins with offset
    {  value, value },  < only values
    {  value, value },
    ...
    { offset, value },
    { offset, value },
};
```

The *Burst* meta command becomes beneficial when code space in the MCU is limited.

The following full example shows the output:

```
cfg_reg registers[] = {
    { 0x00, 0x00 },
    { 0x7f, 0x00 },
    { 0x06, 0x21 },
    { 0x07, 0x05 },
    { 0x08, 0x04 },
    { 0x09, 0xb0 },
    { 0x0a, 0x01 },
    { 0x0b, 0x02 },
    { 0x0c, 0x08 },
    { 0x0d, 0x00 },
    { 0x0e, 0x80 },
    { 0x12, 0x02 },
    { 0x13, 0x08 },
    { 0x14, 0x80 },
    { 0x00, 0x00 },
    { 0x7f, 0x50 },
    { 0x00, 0x01 },
    { CFG_META_BURST, 9 },
    { 0x1c, 0x40 },
    { 0x00, 0x00 },
    { 0x00, 0x7f },
    { 0xff, 0xff },
    { 0x00, 0x00 },  < 1 byte 0x00 padding since 9 is odd
};
```

Data Structure Details

To support commands other than single 8-bit writes, we reserve offset values greater than 127. This is not as great an inconvenience as it may seem because most devices do not use registers with offset higher than 127. We will call such reserved offsets “meta commands” – every 2-tuple in the array is a write command by default, except when the offset is greater than 127 in which case it is a meta command. The byte following the meta command serves as a parameter to it. The PurePath™ Studio GDE supports two different meta commands and are still supported for compatibility purposes. We shall define one more and leave the other 124 for future use.

1. The **Delay** meta command uses an offset of 254 and serves as an indication to the host processor to wait for a certain number of milliseconds, as specified by its parameter, before proceeding to the next command. It is typically used after a software reset of some devices.

```
cfg_reg registers[] = {
    ...
    {  254, 100  }, // Wait for hundred milliseconds at this point
    { 0x01, 0x03 }, // Write 0x03 at offset 1
    ...
};
```

2. The **Switch** meta command uses an offset of 255 and tells the host processor to halt processing in this array and switch to another array of commands, as indicated by its parameter. After finishing that array of commands, the host processor switches back to the original array of commands at the point where it left off. This mechanism is used to decouple system configuration from miniDSP instructions/coefficients and serves to minimize the size of stored arrays by allowing re-use of common sub-sequences.

```
cfg_reg registers[] = {
    ...
    {  255, 1    }, // Switch to array#1
    { 0x01, 0x03 }, // Restart here after finishing array#1
    ...
};
```

3. The **Burst** write meta command shall indicate that what follows is a burst write. The parameter specifies the burst length, including the register offset. Unlike the case for other commands, what follows this command is not another command, but the burst payload itself. After executing the burst write, the host processor must calculate the index of the next valid command by incrementing the current index.

```
cfg_reg registers[] = {
    ...
    {  253, 5    }, // A burst payload of length 5 follows
    { 0xc0, 0x01 }, // Burst is written to register at offset 0xc0, and
    { 0x02, 0x03 }, // the data being written is {1, 2, 3 4}.
    { 0x04, 0x00 }, // Notice that the last byte is wasted since we require
                    // the next command to start at an even position
    { 0x01, 0x03 }, // Next command position
    ...
};
```

Microcontroller Code

To illustrate how such an array may be used in application code the following example is provided. We shall assume the existence of an external library that provides us with an API to do I2C writes, to introduce a delay and to obtain a pointer to other register command arrays.

```
// Externally implemented function that can write n-bytes to the device
extern int i2c_write(unsigned char *data, int n);
// Externally implemented function that delays execution by n milliseconds
extern int delay(int n);
// Externally implemented function to obtain other register command arrays
extern void *switch_array(int array_id, int *array_size);
```

The implementation of these functions is not important, nor their exact prototypes. The application code in the following example may easily be adapted for such changes.

```
void transmit_registers(cfg_reg *r, int n)
{
    int i = 0, sz
    cfg_reg *sw_r;
    while (i < n) {
        switch (r[i].command) {
        case CFG_META_SWITCH:
            sw_r = switch_array(r[i].param, &sz);
            transmit_registers(sw_r, sz);
            break;
        case CFG_META_DELAY:
            delay(r[i].param);
            break;
        case CFG_META_BURST:
            i2c_write((unsigned char *)&r[i+1], r[i].param);
            i += (r[i].param + 1)/2;
            break;
        default:
            i2c_write((unsigned char *)&r[i], 2);
            break;
        }
        i++;
    }
}
```

Appendix C. Dump Definition File (DDF) Specification 3.0

Dump Definition Files (.ddf) are used by the *End-System Integration* page of an *App* to customize the end-system code that is generated. There are 3 types of commands in a .ddf file:

1. Register *Write* Command
2. Register *Dump* Commands
3. Comments.

A proper script uses a mix of these 3 commands as shown in the following snippet:

```
# ------------------------------------------------------------------------------
# Standby (allows I2C access to C-RAM and I-RAM)
# ------------------------------------------------------------------------------
# Select Page 0
0 = 0x00 < Register Write Command
# Set the device into Standby < Comment
2 = 0x10

# ------------------------------------------------------------------------------
# Begin Coefficient Memory (C-RAM_D), Buffer A Dump
# ------------------------------------------------------------------------------
# Page 44 (0x2C) Dump
0 = 0x2C < Register Write Command
0x2C, 0x08-0x7F = xx < Register Dump Command
```

Register Write Command

The Write Command simply causes a specific register to be inserted into a .cfg or .h file. Note that PurePath Console 3 does not send any register read or writes to the EVM for this type of command. The Register Write Command syntax is shown in Table C-1. Note that [register] and [value] can either be in decimal or hex.

Table C-1. Register Write Command Syntax

.ddf file	Description	.cfg file	.h file	Dump Commands

`[register] = [value]`	**Write** value vv to register rr to slave address ss.	`w ss rr vv`	`cfg_reg registers[] = {`
			`  { rr, vv },`
			`  ...`
			`};`
				N/A

An example is shown in Table C-2.

Table C-2. Example Write Command

.ddf file	.cfg file	.h file

`0 = 0x00`
`2 = 0x11`
	`w 98 00 00`
	`w 98 02 11`
		`cfg_reg registers[] = {`
		`  { 0x00, 0x00 },`
		`  { 0x02, 0x11 },`
		`};`

Register Dump Commands

The Dump Commands simply dump the register contents of the specified book, page, and register and then insert them into a .cfg or .h file. In .cfg files, the dump is appended after a 'w' command. A '**xx**' after an '**=**' symbol indicates a Dump Command.

In order to perform a dump of the correct book and page, PurePath Console 3 automatically sends book and page switch commands before performing the read via I^2C. See the Dump Commands column of Table C-3.

For devices with books (for example, TAS2555), it is recommended to use the syntax in the first row. For devices with only pages and registers (for example, TAS5766M), it is recommended to use the syntax in the second row. Note that [book], [page], [register(s)] and [value] can either be in decimal or hex. Also, [register(s)] can be a single register or a range separated by a dash (-) symbol.

Table C-3. Register Dump Command Syntax

.ddf file	Description	.cfg file	.h file	Dump Commands

`[book],[page],[register(s)] = xx`	**Dump** contents of Book bb, Page pp, Register rr into value vv.	`w ss rr vv`	`cfg_reg registers[] = {`
			`  { rr, vv },`
			`  ...`
			`};`
				`w ss 00 00`
				`w ss 7F bb`
				`w ss 00 pp`
				`r ss rr 01`

`[page],[register(s)] = xx`	**Dump** contents of Page pp, Register rr into value vv.	`w ss rr vv`	`cfg_reg registers[] = {`
			`  { rr, vv },`
			`  ...`
			`};`
				`w ss 00 pp`
				`r ss rr 01`

`[register(s)] = xx`	**Dump** contents of Register rr into value vv.	`w ss rr vv`	`cfg_reg registers[] = {`
			`  { rr, vv },`
			`  ...`
			`};`
				`r ss rr 01`

Table C-4 shows an example on how to dump a single register or a range of registers. Note that it is very important to understand that the *Dump* Command only inserts the result into the output file. This means that a *Write* Command to switch book and/or page must be written above the *Dump* Command.

Table C-4. Example Dump Command

.ddf file	.cfg file	.h file

`0x00,0x2C,0x08 = xx`
`0x2C, 0x08 = xx`
`44, 0x08-0x0B = xx`
	`w 98 08 55`
	`w 98 08 55`
	`w 98 08 55`
	`> AA`
	`> 55`
	`> 00`
		`cfg_reg registers[] = {`
		`  { 0x08, 0x55 },`
		`  { 0x08, 0x55 },`
		`  { CFG_META_BURST, 0x05 },`
		`  { 0x08, 0x55 },`
		`  { 0xAA, 0x55 },`
		`  { 0x00, 0x00 },`
		`};`

Comments

Comments are specified by the hash '#' symbol. This command is echoed directly to the output file. Any other command is ignored by the parser.

Table C-5. Example Write Command

.ddf file	.cfg file	.h file
# Comment ; Not visible in the output file // Not visible either	# Comment	// Comment

Full Example

The following code shows an example implementation of a ddf file:

```
; Dump Definition File
; ------------------------------------------------------------------------------
; Project: TAS5766MDCAEVM
; Version: 20150522
; ------------------------------------------------------------------------------

# ------------------------------------------------------------------------------
# Reset
# ------------------------------------------------------------------------------
# Select Page 0
0 = 0x00
# Set the device into Powerdown
2 = 0x11
# Reset Device
1 = 0x11

# ------------------------------------------------------------------------------
# Standby (allows I2C access to C-RAM and I-RAM)
# ------------------------------------------------------------------------------
# Select Page 0
0 = 0x00
# Set the device into Standby
2 = 0x10

# ------------------------------------------------------------------------------
# Begin Coefficient Memory (C-RAM_D), Buffer A Dump
# ------------------------------------------------------------------------------
# Page 44 (0x2C) Dump
0 = 0x2C
0x2C, 0x08-0x7F = xx
...
...
# ------------------------------------------------------------------------------
# Wake
# ------------------------------------------------------------------------------
# Select Page 0
0 = 0x00
# Wake from Standby
2 = 0x00
```

The following code shows a snippet of the generated .cfg file:

```
# ------------------------------------------------------------------------------
# Reset
# ------------------------------------------------------------------------------
# Select Page 0
w 98 00 00
# Set the device into Powerdown
w 98 02 11
# Reset Device
w 98 01 11

# ------------------------------------------------------------------------------
# Standby (allows I2C access to C-RAM and I-RAM)
# ------------------------------------------------------------------------------
# Select Page 0
w 98 00 00
# Set the device into Standby
w 98 02 10

# ------------------------------------------------------------------------------
# Begin Coefficient Memory (C-RAM_D), Buffer A Dump
# ------------------------------------------------------------------------------
# Page 44 (0x2C) Dump
w 98 00 2C
w 98 08 40
> 00
> 00
> 00
> 7F
> FF
> FF
> 00
...
...
# ------------------------------------------------------------------------------
# Wake
# ------------------------------------------------------------------------------
# Select Page 0
w 98 00 00
# Wake from Standby
w 98 02 00
```

The following code shows a snippet of the generated .h file:

```
cfg_reg registers[] = {
// ------------------------------------------------------------------------------
// Reset
// ------------------------------------------------------------------------------
// Select Page 0
  { 0x00, 0x00 },
// Set the device into Powerdown
  { 0x02, 0x11 },
// Reset Device
  { 0x01, 0x11 },

// ------------------------------------------------------------------------------
// Standby (allows I2C access to C-RAM and I-RAM)
// ------------------------------------------------------------------------------
// Select Page 0
  { 0x00, 0x00 },
// Set the device into Standby
  { 0x02, 0x10 },

// ------------------------------------------------------------------------------
// Begin Coefficient Memory (C-RAM_D), Buffer A Dump
// ------------------------------------------------------------------------------
// Page 44 (0x2C) Dump
  { 0x00, 0x2C },
  { CFG_META_BURST, 0x79 },
  { 0x08, 0x40 },
  { 0x00, 0x00 },
  { 0x00, 0x7F },
  { 0xFF, 0xFF },
  { 0x00, 0x7F },
...
...
// ------------------------------------------------------------------------------
// Wake
// ------------------------------------------------------------------------------
// Select Page 0
  { 0x00, 0x00 },
// Wake from Standby
  { 0x02, 0x00 },
};
```

Note that the value next to CFG_META_BURST includes the register byte (0x7F - 0x08 + 1) + 1 = 121. In PurePath™ Console 3, this would correspond to a Burst setting of 120.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265