Si46xx SDK Users Guide

Version 2.7
May-2-2016

Contents

Introduction 4
Software Stack – Master Module 4
Software Stack – Slave Module 5
Customization Process 6
Platform Selection – platform_selector.h 7
Platform Options – platform_options.h 7
Variable Types – COMMON_TYPES.h 12
Drivers 13
    CI.c 13
    Delay.c 13
    In_chip_eeprom.c 13
    Key.c 13
    LCD.c 13
    Si468x_bus.c 13
    SST25VF032B.c 13
    Sys.c 13
    Usb_descriptor.c, Usb_hid.c, Usb_isr.c, USB_process.c, Usb_std_req.c 14
    Utility.c 14
Low Level SDK Interfaces 15
    Hal.c 15
    Firmware_Load_Helpers.c 16
    Firmware_Calibration_Helpers.c 17
High Level SDK Interfaces 18
    Firmware_API_Manager.h 18
    SDK Callbacks to MMI – SDK_Callbacks.h 34
        Data buffer update notifications 34
        Service List update notifications – HD Radio 34
        Other Notifications – FM / FMHD 35
        Service List update notifications - DAB 35
        Other Notifications - DAB 35
        Miscellaneous 36

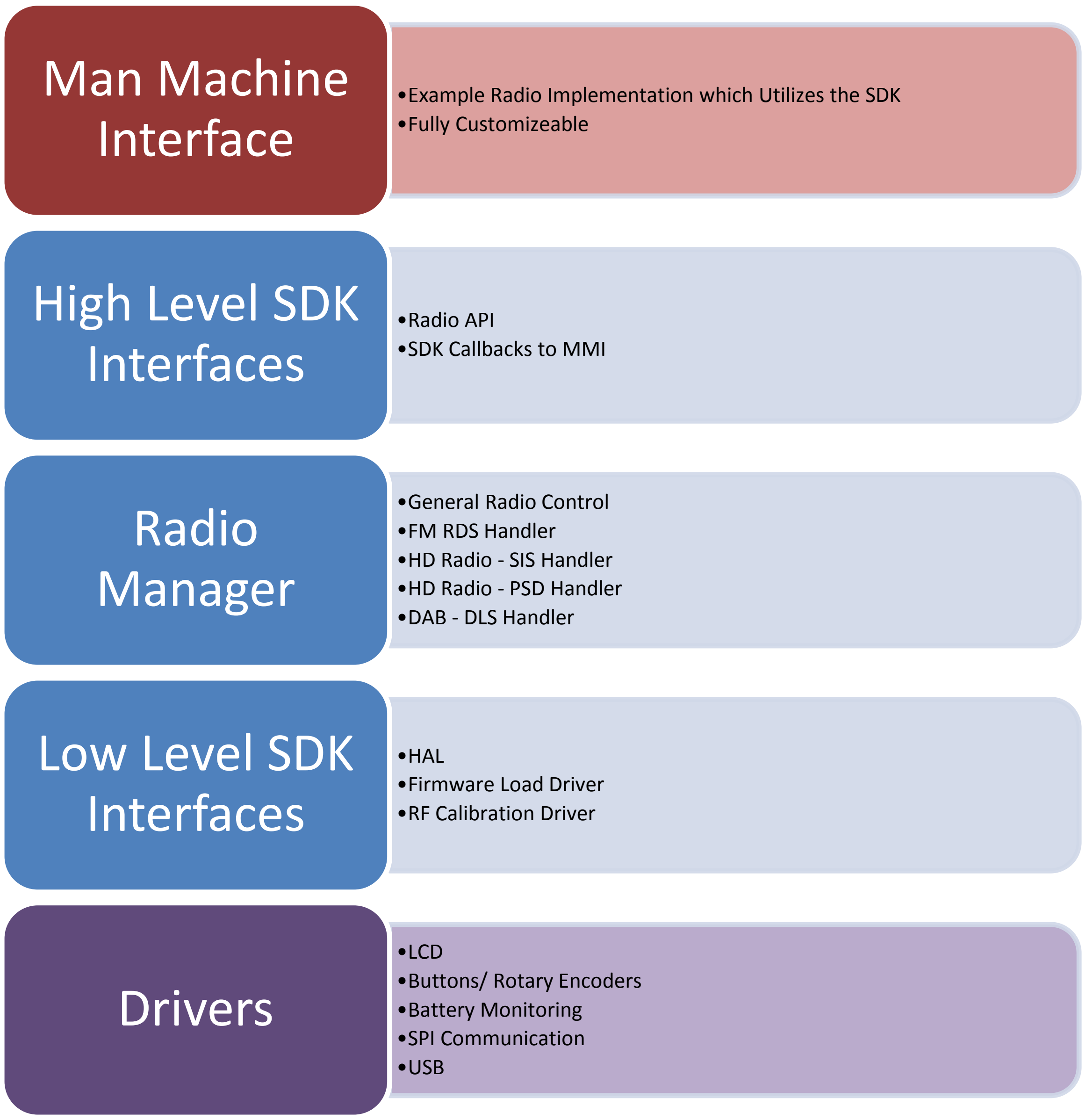
Man Machine Interface 37
mode_dab.c 37
mode_fm_am.c 37
mode_fmhd_amhd.c 37
mode_common.c 37
mode_other.c 37
Slave Control Interface 38
mode_slave.c 38
SDK Revision History 39
Document Revision History 46

Introduction

The si46xx product family is able to support multiple digital radio standards as well as traditional analog radio. With this additional capability over a simple FM tuner, comes some additional software complexity in order to operate the radio chip. In order to ease the transition into digital radio, Silicon Laboratories has created hardware modules and a software development kit (SDK). This document focuses on the SDK software stack and explains its proper usage.

Software Stack – Master Module

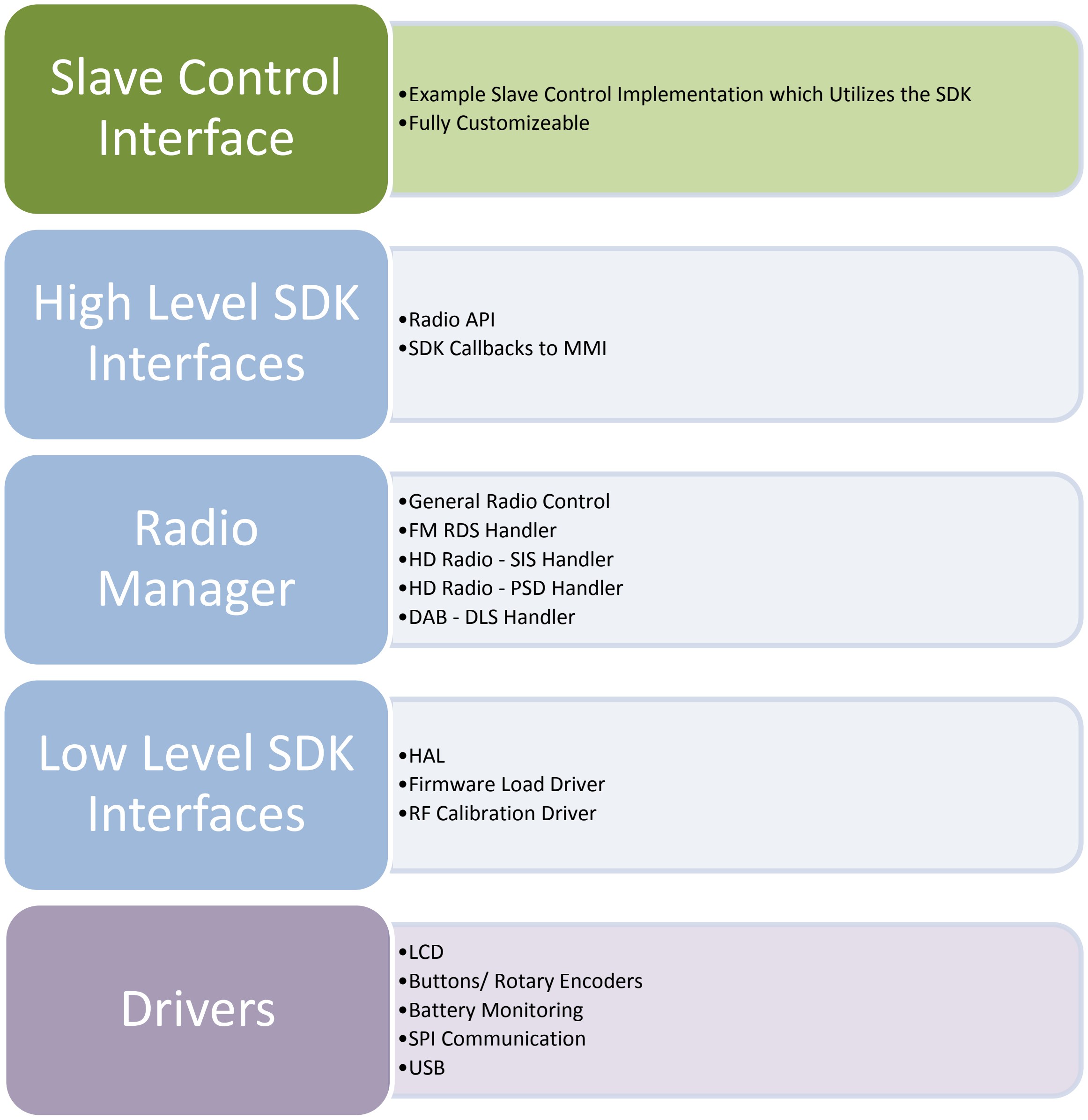
The SDK is organized in a modular and customizable way, using the following basic software stack. This document will address the stack elements from bottom to top.



Software Stack – Slave Module

The SDK package also provides a sample Slave Control Interface which replaces the SDK MMI for those who wish to use a different host controller for system and MMI functions.

Note: this code is not enabled by default: use platform_option - OPTION__OPERATE_AS_SLAVE_NO_MMI



Customization Process

In order to customize the SDK to your platform or existing MMI it is recommended to follow the following procedure.

1. Ensure you have selected or created a platform in the "platform_selector.h"
    a. Example: #define PLATFORM_F380_MODULE_V2_0

2. Identify the "platform_options.h" you which to include in your firmware image
    a. Note these should correspond to the platform selected
    b. Example: #define OPTION__DECODE_PSD

3. Ensure your platform's memory types are correctly assigned in "COMMON_TYPES.h"

4. **Drivers**: Update/Add any drivers (as necessary) for your platform which will be called by your MMI.

5. **Low Level SDK Interfaces:** Update (as necessary) the hardware APIs required by the SDK, making sure that any which are called by the Radio Manager fit within the definitions in "HAL.h"

6. **Low Level SDK Interfaces**: Update the "Firmware_Load_Helpers.c" driver to ensure you are loading from the proper flash addresses.
    a. Alternately a new driver could be created which load the firmware from an external source and sends it directly from the host MCU.

7. **Low Level SDK Interfaces**: Update the "Hardware_Calibration_Helpers.c" to include the proper front-end calibration values for your hardware.

8. **High Level SDK Interfaces**: Update the "SDK_Callbacks.c" to interact with your MMI as desired.

9. **Select Control Type**:
    a. Man Machine Interface: (*default*) Update the MMI source code as desired. MMI calls should be made into "Firmware_API_Manager.h" APIs
        i. Example usage in: mode_dab.c, mode_fm.c, mode_fmhd.c, mode_common.c, mode_other.c
    b. Slave Control Interface: Update the slave module to match your desired protocol and information needs.
        i. Example usage in: mode_slave.c
        ii. Enable slave mode using the platform_option – "OPTION__OPERATE_AS_SLAVE_NO_MMI"

Platform Selection – platform_selector.h

The SDK has been written to support different sets of hardware drivers and build options without needing to have different source code.

Throughout the project you will find cases where "#ifdef" is used to include/exclude drivers based upon which platform is selected. This enables you to have a single source set which is backwards compatible to previous hardware generations when building images.

In addition, the platform options (which features should be included for a given platform) are globally selected by the enabled platform.

Current Active Platforms:

- PLATFORM_F380_MODULE_V2_0
- PLATFORM_F340_DEMOBOARD
- PLATFORM_EVB

Platform Options – platform_options.h

There are many options available to conditionally compile certain features or select between different implementations of a given handler targeted to a specific platform's limitations.

The following options should either be #define in order to select the option, unless otherwise stated.

1) Image Type (select one): OPTION__IMAGETYPE__DAB_FM | OPTION__IMAGETYPE__DAB_FM_AM | OPTION__IMAGETYPE__FMHD | OPTION__IMAGETYPE__ALL

* Depending on the options selected and MMI complexity, a single image for all modes may not fit in the MCU code space, this option enables basic mode selection for code inclusion

- OPTION__IMAGETYPE__DAB_FM: this options includes items for compilation which build a host image for a DAB + FM radio
- OPTION__IMAGETYPE__DAB_FM_AM: this options includes items for compilation which build a host image for a DAB + FM + AM radio
- OPTION__IMAGETYPE__FMHD: this options includes items for compilation which build a host image for an FMHD radio
- OPTION__IMAGETYPE__ALL: this options includes all code which build a host image for a DAB + FM + FMHD + AM + AMHD radio - this image is too large to fit on the SDK Demo Module

2) OPTION__REMOVE_FIRMWARE_FUNCTIONS_NOT_USED_SMALL_SDK - Code Size Reduction. This removes from compilation, functions which are not currently used in the SDK code targeting modules

3) OPTION__MINIMAL_FIRMWARE_FUNCTION_IMPLEMENTATION_SMALLER_SDK - Code Size Reduction. This removes from compilation, elements in some data structures which are not currently used in the SDK code targeting modules.
4) OPTION__INCLUDE_FRONT_END_TEST_FUNCTIONS - This option should be defined during the hardware calibration phase of your design, but should be removed for the shipping product
5) OPTION__INCLUDE_POWER_SAVING_MODE - This option includes the driver which puts the F380 MCU into a lower power state when possible

6) Boot Mode (select one): OPTION__BOOT_FROM_HOST | OPTION__BOOT_FROM_FLASH | OPTION__BOOT_FROM_FLASH_MINI
- OPTION__BOOT_FROM_HOST: this option selects the initialization code which expects the host MCU to load all firmware images
- OPTION__BOOT_FROM_FLASH: this option selects the initialization code which expects all necessary firmware images be loaded from external NVSPI connected flash (not applicable for ROM00)
- OPTION__BOOT_FROM_FLASH_MINI: this option includes the mini-patch in the code space which is loaded from the host, so that all other firmware images can be loaded from external NVSPI connected flash.
- OPTION__BOOT_FROM_FLASH_FULL: this option includes the full-patch in the code space which is loaded from the host, so the patch does not have to be kept in the external NVSPI connected flash.

7) OPTION__AUDIO_ANALOG_ENABLE - enables the code which provides analog audio output

8) OPTION__AUDIO_I2S_ENABLE - enables the code which provides I2S digital audio output
- OPTION__AUDIO_I2S__MASTER_SLAVE: sub-option which sets if the I2S port on the si46xx should operate as Master or Slave (Master 1, Slave 0)
- OPTION__AUDIO_I2S__SAMPLE_RATE_KHZ: sub-option which sets the I2S port sample rate in kHz: range between 32000 and 48000
- OPTION__AUDIO_I2S__SAMPLE_SIZE: sub-option which sets the I2S port sample size: range between 2 and 24
- OPTION__AUDIO_I2S__FRAMING_FORMAT: sub option which sets the I2S framing format: (I2S 0, DSP 6, Left Justified DSP 7, Left Justified 8, Right Justified 9)

9) OPTION__INTB_ENABLE - enables the INTB line fron the si46xx, currently the SDK does not rely on hardware interrupts

10) SPI Bus buffer Size (select one): OPTION__FULL_SIZE_SPI_BUFFER (need ~ 8k ram) | OPTION__SMALL_SPI_BUFFER (need ~ 256 bytes ram)
- OPTION__FULL_SIZE_SPI_BUFFER: this SPI buffer size is intended for advanced hosts with lots of memory

- OPTION__SMALL_SPI_BUFFER: this SPI buffer size is intended for basic hosts (modules) which have very limited RAM resources.

11) OPTION__MCU_PLATFORM - include this option to build the MCU SDK drivers in "HAL.c, Firmware_Load_Helpers.c, and Hardware_Calibration_Helpers.c"
- this is to prevent commonly named SDK Callback files from being build by the incorrect project files. Example differentiating between PLATFORM_EVB and PLATFORM_F380_V2_0

12) OPTION__32BIT_IS_LONG - If selected platform type (INT) is not a 32 bit number and uses LONG instead (ex 8051)

13) OPTION__CONVERT_BIG_ENDIAN - If selected platform architecture is BIG_ENDIAN, include this option because the output of the si46xx is little endian and must be converted

14) RDS decode mode basic/advanced (select one or do not define to disable):
OPTION__RDS_DECODE_ADVANCED | OPTION__RDS_DECODE_BASIC | OPTION__RDS_BUFFER_V4L2
- Determines if the basic or advanced PS/RT RDS display functions are used
- OPTION__RDS_DECODE_ADVANCED:The advanced functions attempt to detect errors and only display complete messages
- OPTION__RDS_DECODE_BASIC: The basic functions are closely tied to the recommendations in the RBDS specification and are faster to update but more error prone
- OPTION__RDS_BUFFER_V4L2: Not currently implemented

15) OPTION__DECODE_SIS - define this option to include the HD Radio SIS data handler
16) OPTION__DECODE_PSD - define this option to include the HD Radio PSD data handler
17) OPTION__PSD_FORMAT_ID3 - define this option to use the HD radio PSD in ID3 format, note: this requires a larger reply buffer to work properly and should not be selected unless modifying the SDK source
18) OPTION__DECODE_DLS - define this option to include the DAB DLS data handler

19) OPTION__FORCE_MEMORY_SPACES - If selected platform is based upon the 8051 architecture, define this option to set memory locations

20) OPTION__DECODE_FMHD_SERVICE_LIST - define this option to include the HD Radio service list handler, this is necessary for identifying and tracking digital audio services
21) OPTION__DECODE_DAB_SERVICE_LIST - define this option to include the DAB service list handler, this is necessary for identifying and tracking digital audio services
22) OPTION__SERVICE_LIST_SMALL - Code Size/RAM Usage Reduction - define this option to limit the service list handlers to only tracking audio services with the minimal set of information from the full list
23) OPTION__DAB_SERVICE_LIST_TIMEOUT_TUNE_MS - the value for this option sets the amount of time, in ms, where the software will exit a failed DAB tune. Example: 10000

24) OPTION__DAB_SERVICE_LIST_SCVLIST_MS - the value for this option sets the amount of time, in ms, where the software will exit a failed DAB service list acquisition. Example: 20000
25) OPTION__DAB_SERVICE_LIST_TIMEOUT_CALLBACK_RATE_MS - the value for this option sets the amount of time, in ms, between callbacks to the MMI for UI update, during the DAB scan/tune process. Example: 100
26) OPTION__PROPERTY_DAB_VALID_RSSI_THRESHOLD - the value for this option is used for the property "DAB_VALID_RSSI_THRESHOLD" a value which should be tuned for the antenna in the system to best represent the RSSI for a good frequency during band scan
27) OPTION__FM_TIMEOUT_SEEK_MS - the value for this option sets the amount of time, in ms, where the software will exit a failed FM Seek. Example: 10000
28) OPTION__FM_SEEK_CALLBACK_RATE_MS - the value for this option sets the amount of time, in ms, between callbacks to the MMI for UI update, during the FM Seek process. Example: 100
29) OPTION__DECODE_HD_ALERTS - define this option to include the HD Radio Alert handler.
30) OPTION__DAB_SUPPORT_DAB_DAB_LINKING - define this to include the code necesssary for service linking between multiple DAB ensembles - in the A1/A2 case (SID=SID)
31) OPTION__DAB_SUPPORT_DAB_DAB_LINKING_A3 - define this to include the code necesssary for service linking between multiple DAB ensembles - in the A3 case (SID != SID, hard links)
32) OPTION__DAB_SERVICE_LIST_PERSISTENCE - define this option to include the ability to store the service list in persistant storage, this can take 4k or more of flash and may affect your ability to fit any code additions into the MCU
33) OPTION__FMHD_SEEK_HDLEVEL_THRESHOLD - the value for this option sets the HDLEVEL which should be used to qualify a valid HD station to seek stop in HD Seek mode
34) OPTION__FMHD_SEEK_OFDM_SAMPLE_NUMBER - the value for this options sets the FM_RSQ_HD_DETECTION property SAMPLES field. See AN649 for explanation of this property.
35) OPTION__OPERATE_AS_SLAVE_NO_MMI - define this option to remove all code related to the MMI in a "Master" module usage. If you are creating a "Slave" module using the SL200, this will free up code space for additional features.
36) OPTION__ADVANCED_METRICS - define this option to include additional metrics for use during evaluation, the advanced metrics are probably unecessary for customer usage so it is recommended not to include this code on the final build.
37) OPTION__MCU_HARDWARE_SPI - define this option to replace the previous software SPI with a driver which uses the F380 SPI hardware block
38) OPTION__DAB_PRESETS_AS_FAVORITES - define this option to replace the previous "stack" style preset implementation with a "favorites" style implementation
   - "Stack" style usage:
      - Add a preset: press and hold the preset+ button
      - Remove a preset: from the preset selection screen, select the desired preset to remove and hold the preset- button.
      - A press to preset+ or preset- enters the preset selection screen.

- Disadvantages: approximately 1k more code space required, limited to 10 presets.
- "Favorites style usage:
    - Add a preset: from any service screen (now playing or browse) press (not hold) the preset+ button to mark that service
    - Remove a preset: from any service screen (now playing or browse) press (not hold) the preset- button to remove that service
    - Enter Browse mode of only presets: hold the preset+ button
    - Browse all serivices: hold the preset- button
    - Advantages: smaller code requirements, as many presets as there are services

39) OPTION__FM_SHOW_STATION_NAME_PST - define this option to show the PST for a given station in the MMI main panel. Note: this is not desirable for US based radios as PST scrolls
40) OPTION__TIME_SETTING - define this option to include MMI code for manually adjusting the clock time (this is auto disabled on the OPTION__IMAGETYPE__DAB_FM_AM and OPTION__IMAGETYPE__DAB_FM images)
41) OPTION__MENU_SETTING - define this option to include the MMI code for a software menu of additional device options/modes (this is auto disabled on the OPTION__IMAGETYPE__DAB_FM_AM image)
42) OPTION__RADIODNS - define this option to include the SDK code necessary to generate a RadioDNS address from FM-RDS and/or DAB
43) OPTION__NVMSPI_RATE_MHZ - this value (if defined will be used to adjust the NVM SPI boot rate (when booting from flash). Do not define the option to remove the associated code.
44) OPTION__DECODE_DAB_ANNOUNCEMENTS - Note: not yet supported in the Radio Manager, but included as a placeholder to include the firmware api elements
45) OPTION__COMMAND_INPUT_MINIMAL_OPTIONS_SMALLER_SDK - commands with a large number of inputs as bits in a byte are changed to the whole byte at once - results in some code space savings
46) OPTION__EXCLUDE_MFGTEST - code related to the UART based MFGTest is removed from the build regardless of it being selected in platform_selector.h
47) OPTION__DAB_LINK_DISQUALIFIER_SNR_SETTLING_TIME_MS - the allowed time for the SNR metric to settle before checking it for the below threshold
48) OPTION__DAB_LINK_DISQUALIFIER_SNR_THRESHOLD - this is the SNR level at which a potential DAB link will be excluded from the candidates which attempt acquisition

SILICON LABORATORIES

Variable Types – COMMON_TYPES.h

The variable types in the SDK have been abstracted to be more portable between platforms. Please check that the selected platform options correctly assign the following types.

uint8_t	8 bit unsigned
int8_t	8 bit signed
uint16_t	16 bit unsigned
int16_t	16 bit signed
uint32_t	32 bit unsigned
int32_t	32 bit signed

RETURN_CODE – The return values for the API functions are 8 bit unsigned

#define **SUCCESS**	0x00
#define **HAL_ERROR**	0x01
#define **INVALID_INPUT**	0x02
#define **INVALID_MODE**	0x04
#define **TIMEOUT**	0x08
#define **COMMAND_ERROR**	0x20
#define **UNSUPPORTED_FUNCTION**	0x80

Drivers

The SDK release provides a sample set of drivers for the F380 based platform "PLATFORM_F380_V2_0". These drivers are separated into files which correspond to their general purpose.

CI.c

This driver is related to the "field_update" feature over USB. It provides the functions for replacing the F380 MCU code while it is running.

Delay.c

This driver implements the timed delay functions which spin the MCU for a requested amount of time. This functionality is required by the Radio Manager to be accessible through the Low Level SDK Interfaces - hal.h.

In_chip_eeprom.c

This driver provides a method for storing persistent information on power down in the MCU flash. It reserves code space for this purpose. Examples of information which the MMI might need to store here include: presets, current audio channel/service, last radio mode used, DAB service list, etc.

Key.c

This driver implements the keypad handler.

LCD.c

This driver implements the LCD functions for the JHD162A. If a different LCD is desired this driver should be replaced with one for the desired LCD model.

Si468x_bus.c

This driver provides code for using the SPI or I2C bus to control the si46xx radio chip. This functionality is required by the Radio Manager to be accessible through the Low Level SDK Interfaces - hal.h.

SST25VF032B.c

This driver provides a method for directly writing and reading from the SST25VFxxx family of SPI flash chips. Note: not all platforms have direct access to the serial flash.

Sys.c

This driver contains all the basic setup functionality for the MCU including: I/O configuration, USB initialization, System clock startup, and initial hardware states.

Usb_descriptor.c, Usb_hid.c, Usb_isr.c, USB_process.c, Usb_std_req.c

These driver files all relate to enabling USB on the F380 MCU – for use with the “field update” feature. Note: If “field update” is not desired, removing these drivers provides a large amount of code space.

Utility.c

This driver is the place where general utility functions are implemented which don’t belong exclusively to any other category.

Low Level SDK Interfaces

The following files have some SDK to driver level callbacks which need to be implemented or modified for the desired platform – care should be taken that the function prototypes are an exact match.

Hal.c

The HAL functions are the basic driver interfaces required by the SDK.

extern RETURN_CODE initHardware(void);
- This function should initialize any hardware drivers, MCU settings, and reset the si46xx

extern RETURN_CODE audioEnable(void);
- This function should turn on any amplifier hardware for the audio path

extern RETURN_CODE audioDisable(void);
- This function should turn off any amplifier hardware for the audio path

extern RETURN_CODE powerDownHardware(void);
- This function should put all the drivers, MCU and Si46xx into the low power state

extern RETURN_CODE writeCommand (uint16_t length, uint8_t *buffer);
- This function is the Si46xx control bus write command. It can be implemented for SPI or I2C.

extern RETURN_CODE readReply (uint16_t length, uint8_t *buffer);
- This function is the Si46xx control bus read command. It can be implemented for SPI or I2C.

extern RETURN_CODE wait_for_CTS (uint32_t timeout_ms);
- This function does a poll, wait loop periodically checking the CTS bit in the reply status from the Si46xx. This function is called for the polling implementation used in the SDK to determine when a command is complete.

extern void delay (uint32_t delay_ms);
- A generic delay function used in many areas of the SDK. Making this function more power efficient is important.

extern RETURN_CODE writePersistentStorage(uint16_t start_addr, uint16_t length, uint8_t* buffer);
- This function provides a method for the SDK to write to the system flash (in MCU, or external to MCU) for retaining service list data in DAB mode. It is only included when both OPTION__INCLUDE_MODE__DAB and OPTION__DAB_SERVICE_LIST_PERSISTENCE are selected.
- Note: care should be used when creating/modifying this driver as page size and erase ability need to be understood and accounted for. Persistent

extern RETURN_CODE readPersistentStorage(uint16_t start_addr, uint16_t length, uint8_t* buffer);

- This function provides a method for the SDK to read from the system flash (in MCU, or external to MCU) for retaining service list data in DAB mode. It is only included when both OPTION__INCLUDE_MODE__DAB and OPTION__DAB_SERVICE_LIST_PERSISTENCE are selected.
- Note: care should be used when creating/modifying this driver as page size and erase ability need to be understood and accounted for.

extern void erasePersistentStorage();

- This function provides a method for the SDK to read from the system flash (in MCU, or external to MCU) for retaining service list data in DAB mode. It is only included when both OPTION__INCLUDE_MODE__DAB and OPTION__DAB_SERVICE_LIST_PERSISTENCE are selected.
- Note: care should be used when creating/modifying this driver as page size and erase ability need to be understood and accounted for.

Firmware_Load_Helpers.c

The firmware load helpers are functions required by the SDK to address variations in location and delivery method of the Si46xx firmware images.

extern RETURN_CODE firmwareSetImage(RADIO_MODE_TYPE mode);

- This function puts the load helper into a specific image mode.
- See "COMMON_TYPES.h" for the definition of RADIO_MODE_TYPE.

extern uint16_t firmwareGetSegment(uint16_t requestedLength, uint8_t** firmwareSegment);

- This function is used to provide the host with a segment (which fits in the command buffers) of a firmware image being loaded using the HOST_LOAD process.
    - Note: the current implementation for "PLATFORM_F380_V2_0" only provides the mini-patch segments. If you intend to use the HOST_LOAD process for all your firmware images, this function should be extended to provide segments of the currently selected image set using "firmwareSetImage()".
- This is still necessary for some platforms which are booting using FLASH_LOAD to load the mini-patch. (OPTION__BOOT_FROM_FLASH_MINI).

extern RETURN_CODE firmwareGetFlashAddress(uint32_t* addr);

- This function returns the appropriate flash addresses (in the NVSPI connected boot flash) for the currently selected image set using "firmwareSetImage()".
- For a platform which does not use the FLASH_LOAD process, this is unnecessary.

Firmware_Calibration_Helpers.c

These functions are related to providing the SDK with the proper calibrations settings for the given hardware platform.

RETURN_CODE calibrationGetPowerUpArgs(uint8_t *ctsien, uint8_t *clk_mode, uint8_t *tr_size, uint8_t *ibias, uint32_t *xtal_freq, uint8_t *ctun, uint8_t *ibias_run)

- This function provides the arguments sent in the POWER_UP si46xx firmware API command.
- These values will depend on the hardware platform, and may need to be adjusted if any clocking or significant layout changes have been made.

RETURN_CODE calibrationGetFrontEndArgs(uint8_t mode, uint16_t *slope, uint16_t *intercept, uint8_t *switchConfig)

- This function provides the slope, intercept, and front end switch values for the selected “mode”.
- If creating new hardware or using a new antenna, it will be necessary to adjust these values for each supported radio mode to ensure maximum performance can be achieved.

High Level SDK Interfaces

This section describes the interfaces to the SDK for the MMI code.

Firmware_API_Manager.h

The MMI code will call directly into the drivers to manage the system I/O, but all radio operation should be performing using this APIs found in “Firmware_API_Manager.h”.

There should be no reason to include internal headers to the SDK or to write commands to the Si46xx directly.

In addition, there are data buffer pointer access functions in this API which provide access to the various data objects so that no additional memory is necessary to utilize those features.

```
/***************************************************************************************
               Silicon Laboratories Broadcast Digital Radio SDK

   EVALUATION AND USE OF THIS SOFTWARE IS SUBJECT TO THE TERMS AND CONDITIONS OF
     THE SOFTWARE LICENSE AGREEMENT IN THE DOCUMENTATION FILE CORRESPONDING
     TO THIS SOURCE FILE.
   IF YOU DO NOT AGREE TO THE LIMITED LICENSE AND CONDITIONS OF SUCH AGREEMENT,
     PLEASE RETURN ALL SOURCE FILES TO SILICON LABORATORIES.

   Top Level API for the Radio Manager of the SDK - MMI interface
   FILE: Firmware_API_Manager.h
   Supported IC : Si468x
   Date: June 20 2014
  (C) Copyright 2014, Silicon Laboratories, Inc. All rights reserved.
****************************************************************************************/

#ifndef __SI46XX_INTERFACE_HANDLER_TYPE__
#define __SI46XX_INTERFACE_HANDLER_TYPE__

#include "common_types.h"
#include "platform_options.h"
#include "feature_types.h"
#include "si46xx_firmware_api_status.h"

typedef enum _GENERAL_SERVICE_TYPE
{
	//SERVICE_TYPE_FMRX_RDS = 0x00,

	SERVICE_TYPE_HD_AUDIO = 0x10,
	//SERVICE_TYPE_FMHD_SIS = 0x11,
	//SERVICE_TYPE_FMHD_PSD = 0x12,
	SERVICE_TYPE_FMHD_LOT = 0x13,

	SERVICE_TYPE_DAB_AUDIO = 0x20,
	SERVICE_TYPE_DAB_SLS = 0x21,
	//SERVICE_TYPE_DAB_DLS = 0x22, // associated with the current audio service - no need to start seperately
	SERVICE_TYPE_DAB_MOT = 0x23,
		SERVICE_TYPE_DMB_VIDEO = 0x30,

		SERVICE_TYPE_ERR
}GENERAL_SERVICE_TYPE;

typedef enum _BASEBAND_STATUS
{
		BASEBAND_IDLE = 0x00,
		BASEBAND_INIT,
		BASEBAND_READY,
		BASEBAND_PROCESSING_COMMAND,
		BASEBAND_SCANNING,
		BASEBAND_FINALIZE,
   BASEBAND_FIRMWAREUPDATE,
		BASEBAND_ERROR
}BASEBAND_STATUS;

typedef enum _AUDIO_MUTE
{
		NOT_MUTED = 0,
		MUTE_LEFT_ONLY = 1,
		MUTE_RIGHT_ONLY = 2,
		MUTE_LEFT_RIGHT = 3,
		MUTE_ERR
}AUDIO_MUTE;

typedef enum _AUDIO_MONO_STEREO_SWITCH
{
		AUDIO_STEREO = 0,
		AUDIO_MONO = 1,
		AUDIO_ERR
}AUDIO_MONO_STEREO_SWITCH;


#ifdef OPTION__HANDLE_ADVANCED_SERVICES
typedef struct _ServiceList ServiceList;
struct _ServiceList
{
		GENERAL_SERVICE_TYPE type;
		uint32_t service_id;
		uint32_t component_id;
		ServiceList *next;
		uint8_t *bufferPtr;
		uint8_t ignore_packet_count;
};
#endif

// Startup Functions
//****************************************************************************************************
******************
//****************************************************************************************************
******************

//****************************************************************************************************
******************
// Function: Initialize
// Inputs:
//     - mode: the selected mode you would like to start the radio in.  Note this is limited by the part's supported modes
//         example: fmonly, fmhd, dab
// Outputs:
//     - RETURN_CODE
// Description: this is the first function you call when starting or changing radio modes.  It handles firmware download
//     and part configuration
//*****************************************************************************************************
******************
RETURN_CODE Initialize(RADIO_MODE_TYPE mode);

//*****************************************************************************************************
******************
// Function: Finalize
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: this is the final function you call when you are finished with the radio.  It will put the part in receiver
//     in its lowest power state
//*****************************************************************************************************
******************
RETURN_CODE Finalize();



// FM + FMHD Functions
//*****************************************************************************************************
******************
//*****************************************************************************************************
******************

//*****************************************************************************************************
******************
// Function: Tune
// Inputs:
//     - freq: the desired tune frequency
//          - for fm/fmhd in 10kHz units. For example - 102.3Mhz = 10230
//          - for am/amhd in 1kHz units. For example - 1000kHz = 1000
// Outputs:
//     - RETURN_CODE
// Description: This function should be called after initial startup in FM/FMHD mode to establish an initial frequency.
//     after initial tune, it is desirable to use TuneStep() and SeekStart() to move between frequencies.
//     Be sure that any calls the SetStepSize() or SetBandLimits() are performed prior to this initial Tune()
//
//     Note: because some amount of time is required for Tune to complete - the host should call SeekTuneCompleteCheck()
//         to verify completion.  No callback is issued because all tune processing occurs in the receiver chip.
//*****************************************************************************************************
******************
RETURN_CODE Tune(uint16_t freq);

//*****************************************************************************************************
******************
// Function: SetStepSize
// Inputs
//     - stepsize_khz: the desired frequency spacing for the region in use.  Example: 100 kHz (100) or 200 kHz (200).
// Outputs:
//     - RETURN_CODE
// Description: This function is used to configure the channel spacing for the current region.
//     Thus function should be called before Tune() or SeekStart()
//*****************************************************************************************************
******************
RETURN_CODE SetStepSize(uint8_t stepsize_khz);
```

```
#ifndef OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
//**************************************************************************************************
******************
// Function: SetBandLimits
// Inputs
//     - limits: the band limits for the region in use.  Defined in common_types.h: JAPAN_76_90, STANDARD_875_1079,
WIDE_78_108
// Outputs:
//     - RETURN_CODE
// Description: This function is used to configure the band limits for the current region.
//     Thus function should be called before Tune() or SeekStart()
//**************************************************************************************************
******************
RETURN_CODE SetBandLimits(FM_BAND_LIMITS limits);
#endif //OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI

//**************************************************************************************************
******************
// Function: TuneStep
// Inputs
//     - stepup: 0: step down, 1: step up.  Used to select the direction for the step.
// Outputs:
//     - RETURN_CODE
// Description: This function is used to advance the current service to the next in the selected direction.
//     For fmonly mode: this will step the frequency: current_frequency + step_size (as defined in SetStepSize())
//         Note: this will also wrap to the other side of a band boundry
//     For fmhd mode:
//         - when HD is acquired on the current frequency, will move to the next service while staying on the frequency.
//         If there is no more subchannels, will advance to the next frequency
//                                        - when HD is not acquired on the current frequency, will move to the next frequency
//         Note: this will also wrap to the other side of a band boundry
//**************************************************************************************************
******************
RETURN_CODE TuneStep(uint8_t stepup);

//**************************************************************************************************
******************
// Function: SeekStart
// Inputs
//     - seekup: 0: seek down, 1: seek up.  Used to select the direction for the seek.
//     - wrap: 0: do not wrap/stop at band edge, 1: wrap.  Used to select if the seek should wrap at the band edge or stop
//     - hdseekonly: 0: seek only with analog indicators for a valid station, 1: seek stop only on stations with HD.
// Outputs:
//     - RETURN_CODE
// Description: This function is used to start a seek to the next valid channel/service.
//     For fmonly mode: this will seek to the next valid frequency according to the RSSI/SNR thresholds
//     For fmhd mode:
//         - when HD is acquired on the current frequency, will move to the next service while staying on the frequency.
//         If there is no more subchannels, will seek to the next valid frequency according to the RSSI/SNR thresholds
//                                        - when HD is not acquired on the current frequency, will seek to the next valid frequency according
to the RSSI/SNR thresholds
//                        Note: this command will start the seek process in the receiver, the user can end the seek early by called
SeekStop().
//                        Note: The host should keep checking SeekTuneCompleteCheck() for a non-zero value to determine the seek
is complete.
//         .  No callback is issued because all seek processing occurs in the receiver chip.
//      Calls to UpdateMetrics are valid to update the current frequency the seek is on while seeking.
```

```
//***************************************************************************************************************
******************
RETURN_CODE SeekStart(uint8_t seekup, uint8_t wrap, uint8_t hdseekonly);

#ifndef OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
//***************************************************************************************************************
******************
// Function: SeekStop
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: This function is used to cancel a seek currently in progress.
//***************************************************************************************************************
******************
RETURN_CODE SeekStop();
#endif //OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI

//***************************************************************************************************************
******************
// Function: SeekTuneCompleteCheck
// Inputs: none
// Outputs:
//     - uint8_t: returned boolean indicating seek/tune is complete.  0 = still processing, 1 = complete.
// Description: Because some amount of time is required for Tune or Seek to complete,
//                          the host should call SeekTuneCompleteCheck() to verify completion
//***************************************************************************************************************
******************
uint8_t SeekTuneCompleteCheck();

//***************************************************************************************************************
******************
// Function: ForceMono_FM
// Inputs:
//     - audio_MonStereo:         AUDIO_STEREO = 0, AUDIO_MONO = 1,
// Outputs:
//     - RETURN_CODE.
// Description: It is desirable during FM testing to force mono for some tests.  This is not necessary for normal operation.
//     Note: this is only applicable to "fmonly" mode.
//***************************************************************************************************************
******************
RETURN_CODE ForceMono_FM(AUDIO_MONO_STEREO_SWITCH audio_MonStereo); //Added for testing device using
SDK based platform

//***************************************************************************************************************
******************



// DAB Functions
//***************************************************************************************************************
******************
//***************************************************************************************************************
******************

//***************************************************************************************************************
******************
// Function: SetFrequencyList_DAB
// Inputs:
//     - numFrequencies - the number of frequencies in the new scan list
//    - frequencyListPtr - a pointer to the 32 bit frequency array for the new frequency list
// Outputs:
//    - RETURN_CODE
// Description: This function changes the list of frequencies used by the software when scanning and tuning by index.
//     NOTE: after changing this list, any existing service list information becomes invalid and must be regenerated (through
ScanBand_DAB() or ManualTune_DAB())
//
//     NOTE: after changing the list, if saving the currently playing frequency when radio is turned off in order to start previous
service upon power on (without rescan) becomes more complicated as well.
//                                Power Down
//          1) extract the current ServiceId,ComponentId - GetCurrentServiceIdentification_DAB
//          2) extract the current frequencyIndex - from dabMetrics
//                                  3) extract which list mode (full/reduced) is currently being used - from MMI code
//          4) store this information + power down
//                                Power Up
//                                  1) extract the stored variables
//          2) set the frequency list back reduced mode if previously in that mode (the chip will default to full mode)
//                                  3) manually tune to the previous index - ManualTune_DAB
//          4) manually restart the original service - StartProcessingChannel (with the type as SERVICE_TYPE_DAB_AUDIO),
and NULL for the bufferPtr)
//          5) to restore the full service list a rescan will be necessary - which will interrupt audio
//***************************************************************************************************
******************
RETURN_CODE SetFrequencyList_DAB(uint8_t numFrequencies, uint32_t* frequencyListPtr);

//***************************************************************************************************
******************
// Function: GetFrequencyList_DAB
// Inputs:
//    - bufferSize - the total size of the frequency buffer in bytes.  Necessary to prevent memory overflow.
// Outputs:
//    - numFrequencies - the number of frequencies in the new scan list
//    - frequencyListPtr - a pointer to the 32 bit frequency array for the new frequency list
//    - RETURN_CODE
// Description: This function reads back the current frequency list being used by the radio chip.  It will default to the full european
list upon reset
//***************************************************************************************************
******************
RETURN_CODE GetFrequencyList_DAB(uint8_t bufferSize, uint8_t* numFrequencies, uint32_t* frequencyListPtr);

//***************************************************************************************************
******************
// Function: ScanBand_DAB
// Inputs: none
// Outputs:
//    - RETURN_CODE
// Description: This function intiates a scan of the European DAB band collecting service list information of available audio
//    programs.  After each valid ensemble's service list is collected, or invalid ensemble skipped, a callback
//    will hand over control to the MMI for UI update: CALLBACK_Updated_Data(DAB_TUNE_SCAN_PROCESS_UPDATE);
//    Once the scan is complete there is a final call to
CALLBACK_Updated_Data(DAB_TUNE_SCAN_PROCESS_COMPLETE);
//***************************************************************************************************
******************
RETURN_CODE ScanBand_DAB();

//***************************************************************************************************
******************
// Function: ScanBandCancel_DAB
// Inputs: none
// Outputs: none
// Description: This function procides a way to cancel a DAB scan currently in progress.
//*******************************************************************************************************
******************
void ScanBandCancel_DAB();

//*******************************************************************************************************
******************
// Function: ManualTune_DAB
// Inputs:
//    - freqIndex - the frequency index desired to tune to - based upon those determined by
SetFrequencyList_DAB()/GetFrequencyList_DAB()
// Outputs:
//    - RETURN_CODE - will give TIMEOUT indication if the frequency was unable to tune.
// Description: This function provides a way to manually set the ensemble to a user indicated preference, the service list will
update
//*******************************************************************************************************
******************
RETURN_CODE ManualTune_DAB(uint8_t freqIndex);

//*******************************************************************************************************
******************
// Function: ReacquireCurrentService_DAB
// Inputs: none
// Outputs:
//    - RETURN_CODE - will give TIMEOUT/INVALID_INPUT indication if the attempt was unsuccessful.  Will give
INVALID_MODE if currently ACQUIRED.
// Description: This function provides a way to run a check if the currently playing service needs to attempt to be restarted.
//       This can be called periodically with little overhead.
//*******************************************************************************************************
******************
RETURN_CODE ReacquireCurrentService_DAB();

//*******************************************************************************************************
******************
// Function: BrowseServicesReset_DAB
// Inputs: none
// Outputs: none
// Description: This function resets the browse index in the service list back to the current playing service.
//      For example:
//      a service list has 3 entries BBC1, BBC2, and BBC3 - the current playing service is BBC2.  If a user browses up one service
//      the UI would show information for BBC3.  If the user does not indicate to start the service after some time, the UI
//      would return to the "Now Playing" screen.  Without a callto BrowseServicesReset_DAB, on the next browse up,
//      the browsed service would be BBC1 instead of BBC3.
//*******************************************************************************************************
******************
void BrowseServicesReset_DAB();

//*******************************************************************************************************
******************
// Function: BrowseServicesChangeSelection_DAB
// Inputs:
//    - stepup: 0: step down, 1: step up.  Used to select the direction for the step in the service list.
// Outputs:
//    - service_name - a pointer to the string for the current browsed service (length = 16 characters)
//    - service_name_encoding - a variable created by caller to be changed to the encoding for the service_name string.
//                                   EBU = 0, UCS2 = 6, UTF8 = 0x0F,
//    - service_pty - a variable created by the caller to be changed to the pty for the current browsed service
//    - RETURN_CODE
```

SILICON LABORATORIES

```
// Description: This function moves up/down the service list and provides UI information for enabling a user to select a service
//      Note: be sure to call BrowseServicesReset_DAB() after leaving the browse menu in order to start browsing at the current
service
//*****************************************************************************************************
******************
RETURN_CODE BrowseServicesChangeSelection_DAB(uint8_t stepup, uint8_t* service_name, uint8_t*
service_name_encoding, uint8_t* service_pty);

//*****************************************************************************************************
******************
// Function: BrowseServicesStartCurrentSelection_DAB
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: While in a browsing mode, this function starts the current browsed service.
//      For example: the user has browsed up 2 times with calls to BrowseServicesChangeSelection_DAB(..) and has decided to
//          start that service - this function would then be called.
//*****************************************************************************************************
******************
RETURN_CODE BrowseServicesStartCurrentSelection_DAB();


#ifdef OPTION__DAB_PRESETS_AS_FAVORITES
//*****************************************************************************************************
******************
// Function: FavoritesSetCurrentService_DAB
// Inputs: none
// Outputs: none
// Description: This function sets the currently visible service as a favorite.
//*****************************************************************************************************
******************
void FavoritesSetCurrentService_DAB();

//*****************************************************************************************************
******************
// Function: FavoritesRemoveCurrentService_DAB
// Inputs: none
// Outputs: none
// Description: This function removed the currently visible service from the favorite list
//*****************************************************************************************************
******************
void FavoritesRemoveCurrentService_DAB();

//*****************************************************************************************************
******************
// Function: FavoritesBrowseOnly_DAB
// Inputs:
//     - enable = 0 = browse all services, 1 = browse only favorites.  toggle to enable favorites mode
// Outputs: none
// Description: This function enables or disables the favorites only service browse mode.
//*****************************************************************************************************
******************
void FavoritesBrowseOnly_DAB(uint8_t enable);

//*****************************************************************************************************
******************
// Function: FavoritesIsCurrentServiceAFavorite_DAB
// Inputs: none
// Outputs:
```

```
//     - uint8_t: 0 = not a favorite, 1 = favorite
// Description: This function provides easy access to the favorites status for a given service for updating the UI
//******************************************************************************************************
******************
uint8_t FavoritesIsCurrentServiceAFavorite_DAB();
#endif

//******************************************************************************************************
******************
// Function: GetCurrentServiceIdentification_DAB
// Inputs: none
// Outputs:
//     - serviceId = the current playing service's service id
//     - componentId = the current playing service's component id
// Description: This function returns the service id and component id of the currently playing audio service.  This is
//      necessary to manually restart the current service after power down using the - ManualTune_DAB and
StartProcessingChannel apis
//******************************************************************************************************
******************
void GetCurrentServiceIdentification_DAB(uint32_t* serviceId, uint16_t* componentId);

//******************************************************************************************************
******************
// Function: StartLastServiceByIndex
// Inputs:
//      - serviceListIndex = the index obtained from the DAB Metrics for the previously played service's location in the service list
// Outputs: none
// Description: This function will attempt to start the previously played service.
//******************************************************************************************************
******************
RETURN_CODE StartLastServiceByIndex(uint8_t serviceListIndex);


#ifdef OPTION__DAB_SERVICE_LIST_PERSISTENCE
//******************************************************************************************************
******************
// Function: SaveServiceList_DAB
// Inputs: none
// Outputs: none
// Description: This function stores the current service list database into persistent storage for use when powering down the radio
//******************************************************************************************************
******************
RETURN_CODE SaveServiceList_DAB();

//******************************************************************************************************
******************
// Function: LoadServiceList_DAB
// Inputs: none
// Outputs:
//      - lastPlayedIndex = the service list index of the last played service, this is the correct input for calling
StartLastServiceByIndex
//                to resume the previous service.
// Description: This function loads the saved service list database into RAM for use when powering up the radio
//******************************************************************************************************
******************
RETURN_CODE LoadServiceList_DAB(uint8_t* lastPlayedIndex);

//******************************************************************************************************
******************
```

```
// Function: EraseServiceList_DAB
// Inputs: none
// Outputs: none
// Description: This function clears the saved service list
//*******************************************************************************************************
******************
void EraseServiceList_DAB();

#endif

//*******************************************************************************************************
******************
// Function: GetCurrentServiceString_DAB
// Inputs: none
// Outputs:
//     - service_name - a pointer to the string for the current playing service (length = 16 characters)
//     - service_name_encoding - a variable created by caller to be changed to the encoding for the service_name string.
//                                             EBU = 0, UCS2 = 6, UTF8 = 0x0F,
//     - service_pty - a variable created by the caller to be changed to the pty for the current playing service
//     - RETURN_CODE
// Description: This function provides service information for display about the current playing service.
//      For example: While in a "now playing" mode, this function would be called to populate the UI
//*******************************************************************************************************
******************
RETURN_CODE GetCurrentServiceString_DAB(uint8_t* service_name, uint8_t* service_name_encoding, uint8_t*
service_pty);

//*******************************************************************************************************
******************
// Function: GetCurrentBrowseServiceString_DAB
// Inputs: none
// Outputs:
//     - service_name - a pointer to the string for the current playing service (length = 16 characters)
//     - service_name_encoding - a variable created by caller to be changed to the encoding for the service_name string.
//                                             EBU = 0, UCS2 = 6, UTF8 = 0x0F,
//     - service_pty - a variable created by the caller to be changed to the pty for the current playing service
//     - RETURN_CODE
// Description: This function provides service information for display about the current selected service for browsing.
//      This is related to "BrowseServicesChangeSelection_DAB()" but enables a re-read of the information without adjusting the
browse index
//*******************************************************************************************************
******************
RETURN_CODE GetCurrentBrowseServiceString_DAB(uint8_t* service_name, uint8_t* service_name_encoding, uint8_t*
service_pty);


#ifndef OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
//*******************************************************************************************************
******************
// Function: GetCurrentEnsembleNameString_DAB
// Inputs: none
// Outputs:
//     - ensemble_name - a pointer to the string for the currently tuned ensemble (length = 16 characters, encoding = EBU)
//     - RETURN_CODE
// Description: This function provides the string which describes the currently tuned ensemble.
//*******************************************************************************************************
******************
RETURN_CODE GetCurrentEnsembleNameString_DAB(uint8_t* ensemble_name);
#endif //OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
```

```
//*******************************************************************************************************
******************
// Function: GetCurrentTime_DAB
// Inputs:
//     - region - the location desired for the current time (see DAB_TIME_REGION_TYPE)
// Outputs:
//     - time_data - the current time info from the tuned ensemble (see DAB_TIME for the struct format)
//     - RETURN_CODE
// Description: This function provides the current time for the tuned ensemble in local or UTC.
//*******************************************************************************************************
******************
RETURN_CODE GetCurrentTime_DAB(DAB_TIME_REGION_TYPE region, dab_time* time_data);

// Common Functions
//*******************************************************************************************************
******************
//*******************************************************************************************************
******************

//*******************************************************************************************************
******************
// Function: AudioLevel
// Inputs:
//     - level: the volume level of the audio output by the receiver.  Range = 0 -> 63
// Outputs:
//     - RETURN_CODE
// Description: This function provides control of the volume level of the audio output by the receiver.
//*******************************************************************************************************
******************
RETURN_CODE AudioLevel(uint16_t level);

//*******************************************************************************************************
******************
// Function: AudioMute
// Inputs:
//     - mute: the desired mute type:
//             NOT_MUTED = 0, MUTE_LEFT_ONLY = 1, MUTE_RIGHT_ONLY = 2, MUTE_LEFT_RIGHT = 3,
// Outputs:
//     - RETURN_CODE
// Description: This function provides control of mute for the audio output by the receiver.
//*******************************************************************************************************
******************
RETURN_CODE AudioMute(AUDIO_MUTE mute);

//*******************************************************************************************************
******************
// Function: UpdateMetrics
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: This function updates the receiver metrics for the current operating mode.  This function is intended to be
//      called periodically to provide up to date info for signal level indicators.
//
//      To get the metrics: call the listed function for the current mode to get a pointer to the metrics which were updated
//      - fm_metrics* MetricsGetFMPtr();
//      - fmhd_metrics* MetricsGetFMHDPtr();
//      - dab_metrics* MetricsGetDABPtr();
//      Note: you do not need to create a variable simply call the function and dereference the result:
```

```
//          example MetricsGetFMPtr()->FREQUENCY_10KHZ
//
// Be careful not to call this function too often as it will always read from the reciever to perform the update.
//*****************************************************************************************************
******************
RETURN_CODE UpdateMetrics();

//*****************************************************************************************************
******************
// Function: UpdateServiceList
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: This function is called whenever the host expects a service list for the current freq/ensemble should be updated.
//*****************************************************************************************************
******************
RETURN_CODE UpdateServiceList();

//*****************************************************************************************************
******************
// Function: UpdateDataServiceData
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: This function handles the necessary processes to update the Data services running.  It should be called periodically.
//      dab: updates DLS
//      fmhd: updates SIS and PSD, as well as RDS
//      fmonly: updates RDS
//
//      To access the data buffers - see the pointer functions in the Data Buffer Pointers section for the desired data type.
//                                      example: PSDGetTitlePtr, SISGetSloganPtr, DLSGetPtr, RDSGetRadioTextPtr, etc
//*****************************************************************************************************
******************
RETURN_CODE UpdateDataServiceData();



// Special Functions
//*****************************************************************************************************
******************
//*****************************************************************************************************
******************
RETURN_CODE Get_part(uint16_t* chip_part);

// Advanced Functions
RETURN_CODE StartProcessingChannel(GENERAL_SERVICE_TYPE type, uint32_t service_id, uint32_t component_id,
uint8_t *service_buffer);

#ifndef OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
   RETURN_CODE StopProcessingChannel(uint32_t service_id, uint32_t component_id);
#endif

status_bits GetSi46XXStatus();


// The following Properties are excluded from SetProperty as there are other direct APIs do perform the desired action
//   - Set Property will return INVALID_INPUT if called for these properties
// ************************************************************
// FM_SEEK_BAND_BOTTOM
```

SILICON LABORATORIES

```
// FM_SEEK_BAND_TOP
// FM_SEEK_FREQUENCY_SPACING
// ****************************************************************
RETURN_CODE SetProperty(uint16_t prop_id, uint16_t value);
RETURN_CODE GetProperty(uint16_t prop_id, uint16_t *value);


//*****************************************************************************************************
******************
// Function: StartFirmwareUpdate
// Inputs: none
// Outputs:
//     - RETURN_CODE
// Description: This function puts the part into firmware update mode - listening on the USB for image download commands
//*****************************************************************************************************
******************
RETURN_CODE StartFirmwareUpdate();


#ifdef OPTION__RADIODNS
//*****************************************************************************************************
******************
// Function: UpdateRadioDNS_FMRDS
// Inputs:
//     - none
// Outputs:
//     - elements - all the elements needed to compose the radioDNS address string
//     - RETURN_CODE
// Description: This function calculates each of the pieces of the FM RDS RadioDNS string
//     Note: This does not support the country code form of the string as that information is not available from the broadcast alone
// String format: <frequency>.<pi>.<gcc>.fm.radiodns.org
//     - frequency: in 5 characters (leading 0 for frequencies below 100MHz)
//     - pi: in 4 characters
//     - gcc: in 3 characters
//*****************************************************************************************************
******************
RETURN_CODE UpdateRadioDNS_FMRDS(radioDNS_fm* elements);

//*****************************************************************************************************
******************
// Function: UpdateRadioDNS_DAB
// Inputs:
//     - none
// Outputs:
//     - elements - all the elements needed to compose the radioDNS address string
//     - RETURN_CODE
// Description: This function calculates each of the pieces of the DAB RadioDNS string
//     Note: This does not support the <apptype-uatype>.<pa>. variant
// String format: <scids>.<sid>.<eid>.<gcc>.dab.radiodns.org
//     - scids: in 1 character
//     - sid: in 4 characters (when the upper 16 bits are 0) or
//            in 8 characters (when the upper 16 bits are non-zero)
//     - eid: in 4 characters
//     - gcc: in 3 characters
//*****************************************************************************************************
******************
RETURN_CODE UpdateRadioDNS_DAB(radioDNS_dab* elements);
#endif
```

```
#ifdef OPTION__INCLUDE_FRONT_END_TEST_FUNCTIONS
//****************************************************************************************************
******************
// Function: Test_DeterminePeakANTCAP_DAB
// Inputs:
//     - freqIndex - the DAB frequency to use for the test
// Outputs:
//     - antcapPeak - the antcap value determined to generate the best highres RSSI
// Description: This function performs a single run of the front end calibration process to collect the peak ANTCAP value
//****************************************************************************************************
******************
void Test_DeterminePeakANTCAP_DAB(uint8_t freqIndex, uint16_t* antcapPeak);
extern uint16_t _ANTCAP_DAB; // variable available globally for updating the user with the state of the process

//****************************************************************************************************
******************
// Function: Test_GetBER_DAB
// Inputs:
//     - berPattern - the 8 bit pattern for the test ETI.  example 0b11111111 or 0b00000000
// Outputs:
//     - errorBits - the number of error bits recieved from the (audio) test pattern ETI
//     - totalBits - the total number of bits recieved from the (audio) test pattern ETI
//     - passResult - the boolean (1 = pass, 0 = fail) for the BER calculation being less than 10e-4.
//     - RETURN_CODE
// Description: This function provides the current BER.  the BER test is expected to capture 1,000,000 samples
//****************************************************************************************************
******************
RETURN_CODE Test_GetBER_DAB(uint8_t berPattern, uint32_t* errorBits, uint32_t* totalBits, uint8_t* passResult);


//****************************************************************************************************
******************
// Function: Test_GetBER_FMHD
// Inputs: none, the BER test is expected to capture 1,000,000 samples per iBiquity
// Outputs:
//     - errorBits - the number of error bits recieved from IB_FMr208c_e1wfc204
//     - totalBits - the total number of bits recieved from IB_FMr208c_e1wfc204
//     - passResult - the boolean (1 = pass, 0 = fail) for the BER calculation being less than 5e-5.
//     - RETURN_CODE
// Description: This function provides the current BER
//****************************************************************************************************
******************
RETURN_CODE Test_GetBER_FMHD(uint32_t* errorBits, uint32_t* totalBits, uint8_t* passResult);

//****************************************************************************************************
******************
// Function: Test_StartFreeRunningBER_FMHD
// Inputs: none
// Outputs: none
// Description: This function enables and resets the BER counters, this is intended to be used before
Test_GetFreeRunningBER_FMHD
//****************************************************************************************************
******************
RETURN_CODE Test_StartFreeRunningBER_FMHD();

//****************************************************************************************************
******************
// Function: Test_GetFreeRunningBER_FMHD
// Inputs:
//     - bitType: an enumeration for the HD BER type to check: see definition of FMHD_BER_TYPE
// Outputs:
//     - errorBits - the number of error bits recieved
//     - totalBits - the total number of bits recieved
// Description: This function provides the current BER
//*****************************************************************************************************
******************
RETURN_CODE Test_GetFreeRunningBER_FMHD(uint8_t bitType, uint32_t* errorBits, uint32_t* totalBits);


#endif

// Data Buffer Pointers
//*****************************************************************************************************
******************
//*****************************************************************************************************
******************

//Metrics
fm_metrics* MetricsGetFMPtr();
fmhd_metrics* MetricsGetFMHDPtr();
dab_metrics* MetricsGetDABPtr();
am_metrics* MetricsGetAMPtr();
amhd_metrics* MetricsGetAMHDPtr();

//Service Lists
#ifdef OPTION__DECODE_FMHD_SERVICE_LIST
    fmhd_service_list_fast* FMHDServiceListFastPtr();
    fmhd_service_list* FMHDServiceListAudioPtr();
#endif

#ifdef OPTION__DECODE_DAB_SERVICE_LIST
    dab_service_list* DABServiceListAudioPtr();
#endif

// RDS
uint16_t RDSGetProgramIdentifier();
uint8_t  RDSGetProgramType();
#ifdef OPTION__RADIODNS
    uint8_t RDSGetECC();
#endif
#ifdef OPTION__RDS_BUFFER_V4L2
    //TODO: add V4L2
#else
    #ifndef OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI
        void ConfigureRDSOptions(uint8_t ignore_AB_flag);
    #endif //OPTION__REMOVE_SDK_FUNCTIONS_NOT_USED_BY_MMI

    uint8_t* RDSGetRadioTextPtr();   // Displayed Radio Text (64 bytes)
    uint8_t* RDSGetProgramServiceTextPtr();    // Displayed Program Service text (8 bytes)
    rds_time RDSGetCurrentTime();
#endif
//TODO: add BLER interface if desired
//TODO: add AF APIs


#ifdef OPTION__DECODE_SIS
    fmhd_sis_slogan* SISGetSloganPtr();
    fmhd_sis_universal_short_name* SISGetUSNPtr();
    fmhd_sis_station_message* SISGetSMPtr();
    uint8_t* SISGetSNLPtr();
    uint8_t* SISGetSNSPtr();
    uint8_t* SISGetStationIDPtr();
    uint8_t* SISGetStationLocLatPtr();
    uint8_t* SISGetStationLocLongPtr();
#endif

#ifdef OPTION__DECODE_PSD
    fmhd_psd_generic_string* PSDGetTitlePtr();
    fmhd_psd_generic_string* PSDGetArtistPtr();
    fmhd_psd_generic_string* PSDGetAlbumPtr();

    // On Genre: Need to check for (x) format amd flag
    // C-SDK layer will not process those as the memory footprint is too large to carry the lookup table
    fmhd_psd_generic_string* PSDGetGenrePtr();
#endif

#ifdef OPTION__DECODE_HD_ALERTS
    fmhd_alert_string* HDAlertGetPtr();
#endif

#ifdef OPTION__DECODE_DLS
    dab_dls_string* DLSGetPtr();
#endif



#endif
```

SDK Callbacks to MMI – SDK_Callbacks.h

The SDK does provide some callbacks to the MMI which are used for notifying the MMI of certain data updates, events, etc. The current callbacks are listed below.

The MMI is expected to implement the usage of the callbacks in "SDK_Callbacks.c"

```
typedef enum _SDK_CALLBACKS_UPDATED_DATA_TYPE
{
```

Data buffer update notifications

```
   UPDATED_RDS_PI,
   UPDATED_RDS_PTY,
   UPDATED_RDS_RT,
   UPDATED_RDS_PST,
   UPDATED_RDS_TIME,

   UPDATED_SIS_SLOGAN,
   UPDATED_SIS_UNIVERSAL_SHORT_NAME,
   UPDATED_SIS_STATION_MESSAGE,
   UPDATED_SIS_STATION_NAME_LONG,
   UPDATED_SIS_STATION_NAME_SHORT,
   UPDATED_SIS_STATION_ID,
   UPDATED_SIS_LOC_LAT,
   UPDATED_SIS_LOC_LON,

   UPDATED_PSD_TITLE,
   UPDATED_PSD_ARTIST,
   UPDATED_PSD_ALBUM,
   UPDATED_PSD_GENRE,

   UPDATED_HD_ALERT,

   UPDATED_DLS_STRING,
   CLEAR_DLS_COMMAND,
```

Service List update notifications – HD Radio

UPDATED_SERVICE_LIST_FAST – the quick "service list" which allows audio service detection with no other information

UPDATED_SERVICE_LIST_AUDIO – the full audio service list is ready

UPDATED_SERVICE_LIST_DATA – the full data service list is ready

Other Notifications – FM / FMHD

FM_SEEK_PROCESS_UPDATE - The seek process can take considerable time to complete. This callback hands off execution to the MMI to perform any pending LCD updates or button processing, etc. This callback will occur at the rate defined in "OPTION__FM_SEEK_CALLBACK_RATE_MS"

FM_SEEK_PROCESS_COMPLETE – the current seek has completed.

Service List update notifications - DAB

UPDATED_SERVICE_LIST_DAB – The band list has been updated with a new ensemble (multiplex) services.

SERVICE_LIST_BUFFER_FULL_ERROR – The current list has exceeded to allocated list size or there was a related handler error. This is a major error and should not occur, if it does the buffer for service needs to be increased to accommodate such a densely filled region.

UPDATED_SERVICE_LINKING_DAB – The linking info has been updated for the current ensemble.

Other Notifications - DAB

CLEAR_DLS_COMMAND – The currently scrolling DLS string should be cleared. It also notifies that the DLS buffer has been cleared.

DAB_TUNE_SCAN_PROCESS_UPDATE – The band scan, manual tune, and service list processing processes can take considerable time to complete. This callback hands off execution to the MMI to perform any pending LCD updates or button processing, etc. This callback will occur at the rate defined in "OPTION__DAB_SERVICE_LIST_TIMEOUT_CALLBACK_RATE_MS"

DAB_TUNE_REQUIRED_UPDATE – This callback indicates the selected service is in a different ensemble (multiplex) and has required a re-tune. This should notify the user of this delay.

DAB_TUNE_SCAN_PROCESS_COMPLETE – The current band scan or manual tune process has completed.

EVENT_ENSEMBLE_RECONFIGURATION – An ensemble reconfiguration has occurred
EVENT_ENSEMBLE_RECONFIGURATION_WARNING – An ensemble reconfiguration is pending

EVENT_CURRENT_SERVICE_NO_LONGER_AVAILABLE – This callback notifies the user that the currently selected service no longer available.

EVENT_FINDING_ALTERNATE_SERVICE – The attempt the start the current service failed, or the current service has lost acquisition; the radio is automatically searching for alternates.

Miscellaneous

EVENT_READY_FOR_FIRMWARE_UPDATE – The device has completed entering the firmware update mode. It is now OK to being the process on the PC side.

DAB_FRONT_END_CAL_PROCESS_UPDATE – The ANTCAP peak measurement loop has completed – ready to display the data to the user.

Man Machine Interface

The SDK release provides a sample MMI which shows proper usage of the SDK callbacks and APIs in the High Level SDK Interfaces. The MMI is separated into files which correspond to their general purpose.

mode_dab.c

This file contains the code which is unique to running the MMI in DAB mode. This includes initialization, keypad handling, and the general LCD update handler.

mode_fm_am.c

This file contains the code which is unique to running the MMI in analog FM /AM mode. This includes initialization, keypad handling, and the general LCD update handler.

mode_fmhd_amhd.c

This file contains the code which is unique to running the MMI in HD Radio mode. This includes initialization, keypad handling, and the general LCD update handler.

mode_common.c

This file contains the code which is common between analog FM, HD Radio, and DAB mode. This includes string data display, radio metric display, time management, presets, and volume.

In addition there are some items specific to FM and HD Radio only. This includes tune up/down, seek up/down,

mode_other.c

This file contains general purpose code for clock, alarm, audio control, and the menu system.

Slave Control Interface

The SDK release provides a sample Slave Control Interface which demonstrates usage of the UART for command handling and data transfer using APIs in the High Level SDK Interfaces. This sample can be modified to match an existing protocol you may have implemented for an existing design, or used as provided.

mode_slave.c

This file contains the code which is unique to running the MMI in DAB mode. This includes initialization, keypad handling, and the general LCD update handler.

Note: this code is not enabled by default: use platform_option - OPTION__OPERATE_AS_SLAVE_NO_MMI

SDK Revision History

- Revision 1.9.7
  - Added new logic for starting services which requires a sufficiently high SNR before attempting the service – this is important for service following logic to avoid stopping on a weak service which cannot be decoded, but passed the ACQ check.
- Revision 1.9.6
  - Updated included firmware for the Si468x
    - FMHD – 5.0.4
    - AMHD – 2.0.11
  - SDK: further improvements for service linking / service following use cases
- Revision 1.9.5
  - Updated included firmware for the Si468x
    - DAB – 5.0.5
  - SDK: bug fixes for service linking / service following use cases
- Revision 1.9.4
  - SDK: Removed previous fix where the first 2 characters bug was ignored if zero was passed to the parse PS method. This should be fixed because of the change needed in 1.9.3.
- Revision 1.9.3
  - SDK: Fixed bug where RDSFIFOUSED was not being checked correctly and an empty FIFO was being parsed as valid RDS data.
- Revision 1.9.2
  - SDK: Fixed bug where first 2 characters of RDS-PS would not be displayed correctly.
- Revision 1.9.1
  - Updated included firmware for the Si468x
    - FMHD – 4.0.12
  - SDK: updated firmware api support
    - Updated FM/AM Tune and Seek methods to match new FMHD image.
    - Updated FM RSQ Status to match new command signature.
- Revision 1.9.0
  - Updated included firmware for the Si468x
    - DAB – 4.0.5
  - SDK: updated firmware api support
    - Announcements (not yet supported in Radio manager)
    - Service Following related FIGs 0/6, 0/21, 0/24 API update
  - SDK: added support for service following when SID != SID case
  - SDK: after attempting all service candidates and failing, the service start/following logic will now continue to retry.
- Revision 1.8.0
  - Updated included firmware for the Si468x
    - DAB – 4.0.3

SILICON LABORATORIES

  - SDK: changed the DAB tune/scan code to utilize the new "Fast Detect" feature when qualifying stations
- Revision 1.7.4
  - Updated included firmware for the Si468x
    - DAB – 3.2.12
  - SDK/MMI: Added RadioDNS Feature for DAB and FM Modes
  - MMI: Removed automatic band scan when initial service fails to start in DAB
  - MMI: Fix DLS update issue
- Revision 1.7.3
  - Updated included firmware for the Si468x
    - AMHD – 1.0.5
    - DAB – 3.2.10
  - Include licensing text in each source file
- Revision 1.7.2
  - AMHD work mode support
    - SDK:add AMHD relation function include AMHD tune/seek/SIS/PSD
    - MMI:add AMHD UI support-tune,seek and preset function
    - Rename the mode_fmhd.c/h to mode_fmhd_amhd.c/h
      - Updated IDE project file (.wsp) be sure to update your project accordingly
  - DAB SDK changes to improve service start success rate and prioritize RF when multiple ensembles exist for a given service (DRUK requirement).
- Revision 1.7.1
  - Updated included firmware for the Si468x
    - FMHD – 3.0.19
    - DAB – 3.2.7
  - FM Seek Radio Manager code refactored to be more code efficient
  - DAB fixes for particular test signals
  - General SDK cleanup from 1.7.0
- Revision 1.7.0
  - Updated include firmware for the si468x
    - FMHD – 3.1.19
    - DAB – 3.2.6
    - AM – 1.0.4
  - AM work mode support
    - SDK: Add AM relation function,include AM tune, seek, status function.
    - MMI: Add AM UI support –tune, seek and preset function
    - Rename mode_fm.c to mode_fm_am.c
    - Remove Menu item and calendar setting for saving code by default,

- Revision 1.6.2.1
  - Updated included firmware for the Si468x
    - FMHD – 3.0.17

        - DAB – 3.2.1
    - Note: No SDK code changes – equivalent to 1.6.2 release
- Revision 1.6.2
    - Driver improvements
        - Added new “ultra low power” mode for when the module is isle for 10 seconds in the “Off” state. The screen will shut off and the MCU enters a slower clocked state
        - XTAL oscillator driver modified to help oscillator start on hardware which previously has an oscillator start issue
    - DAB improvements
        - Band scan is forced after frequency list change in order to prevent service list index corruption
        - Added additional logic to handle when a service does not start or disappears due to a reconfiguration
        - Manual Tune no longer clears the service list
        - Widened the UK DAB frequency list to include 10B, 10C, 10D, and 11A.
        - Service browse screen now takes longer to time out to the “Now Playing” screen
        - “Favorites” can now be toggled with either preset key, “Favorites” browse mode can now be toggles with holding either preset key.
- Revision 1.6.1
    - Fixed an issue in DAB SDK during reconfigurations.
- Revision 1.6.0
    - Field Update Improvements - ***This will require devices programmed with 1.4.1 or 1.5.0 to be flashed using the debugger (C2) interface.***
        - Fixed issue found in 1.4.1 and 1.5.0 releases preventing MCU update and unrecoverable modules using USB update.
        - Bootloader patches are no longer erased during default update
        - Additional CRC checks and handshaking during update to improve reliability.
        - New “blank” flash recover method provided so if an update to the boot flash fails, the module can still be repaired through USB
    - DAB improvements
        - SDK/MMI: Added new DAB preset option which marks services as “favorites” rather than independently saving a “stack” of 10 presets. This enables as many presets as available services while saving MCU code space.
            - Enabled by default using OPTION__DAB_PRESETS_AS_FAVORITES
        - SDK/MMI: Updated the storage of state information is save whenever any state is changed.
            - + a new service/frequency is selected (after stabilizing)
            - + the volume is changed (after stabilizing)

            - + preset changes (add/remove)
            - + service list updates (dab)
            - mode switch (previously implemented)
            - power down (previously implemented)
        - MMI: changed the auto time from DAB radio to use “local” mode rather than “UTC”
        - MMI: Added Front End Calibration test to default build
        - New DAB firmware inclusion – DAB_RADIO 3.2.0
    - <u>FMHD improvements</u>
        - SDK: Added new property to correct time alignment error in FMHD 3.0.16 image
        - SDK: Improved the HD-Only seek to properly stop when no HD stations are found.
        - SDK/MMI: Added new controls for enabling HD Split mode as required by iBiquity for testing.
    - Improved boot logic to no longer load/require the FM image
    - MMI: Added a sample slave control interface when using OPTION__OPERATE_AS_SLAVE_NO_MMI
    - Drivers: Update to the SPI driver to enable the MCU SPI block to be used rather than manual control.
        - Enabled by default using OPTION__MCU_HARDWARE_SPI

- Revision 1.5.0
    - SDK/MMI: Added new option to aid in “slave” module software development using the SL200. This option will remove all MMI specific code which can be replaced with a custom slave interface parser/driver/additional system features.
        - To enable define OPTION__OPERATE_AS_SLAVE_NO_MMI
        - Free space available for customization of slave device:
            - DAB/FM: ~22 kB of 64 kB
            - FMHD: ~29 kB of 64 kB
            - DAB/FM/FMHD (World Mode): ~11 kB of 64 kB
    - SDK/MMI: Added HD Only Seek feature
        - Modified SDK API – will require any MMI code which was customized and calls this API to be changed.
            - RETURN_CODE SeekStart(uint8_t seekup, uint8_t wrap, uint8_t hdseekonly);
        - Use HD Seek button on module board to toggle this mode on and off (FMHD mode only)
    - SDK/MMI: Changed default FMHD BER
        - By request: the BER is now a free running counter of error and total bits for all available HD BER types. Use the tuning knob or up/down keys to scroll between the available P1,P2,P3, PIDS Bit, or PIDS Block.

SILICON LABORATORIES

        - New APIs:
            - RETURN_CODE Test_StartFreeRunningBER_FMHD()
            - RETURN_CODE Test_GetFreeRunningBER_FMHD(uint8_t bitType, uint32_t* errorBits, uint32_t* totalBits);
    - MMI: retain "Volume" level at power off
    - MMI: disabled radio mode switch while in radio off state. Press ON/OFF to enable radio mode, then switch bands if you are not in the desired mode.
    - MMI: changed default FM frequency spacing to 100 kHz to aid in testing "menu-less" implementations.
    - SDK/MMI: Added DAB metrics debug option "OPTION__ADVANCED_METRICS" which provides FIC_Quality and FFT_Offset metrics in addition to RSSI and SNR.
        - Note: due to space limitations, the metrics are not labeled but are in the following order: RSSI SNR FIC_Quality FFT_Offset

- Revision 1.4.1
    - The same as 1.4.0, with "field update" script change for certain upgrade paths which we causing SI468x firmware load issues.

- Revision 1.4.0
    - Overall code size reductions/project setting changes
        - MMI: new method for converting MJD which does not require floating point
        - Build settings now optimize for size rather than speed
    - Driver: Improved Field Update (USB firmware update) speed
    - Driver: Improved system speed (spi bus improvement)
    - MMI: preset improvements
    - SDK/MMI: added "service following" support to DAB image
    - SDK: when signal is lost in DAB, the radio will automatically search for the same service on duplicate ensembles/services found in band scan or signaled through FIG data
    - SDK: DAB service list persistence added – now the service list is stored when powering down and the last played service will resume automatically at power up. This includes mode switches between FM and DAB.
    - DRUK requirements met (see service following & service list persistence additions above)
    - NOTE: this image will require the SI468x firmware versions released in 1.3.0. Be sure to perform a full field update of the boot firmware – see the readme.txt

- Revision 1.3.0
    - New DAB/FMHD firmware inclusion
        - DAB – 3.0.5
        - FMHD – 3.0.16
    - SDK stability improvements

- ATS command support additions
- Additional changes for iBiquity requirements: HD Logo, Alerts
- Removed “alarm” example MMI code from default build to make room for new additions
- MMI preset improvements
- MMI character set support improvements
- DRUK requirement updates (pending one final change)

- Revision 1.2.1
  - SDK stability improvements
  - ATS command support additions
  - FMHD BER test will exit quickly in low signal conditions
  - DAB BER test will exit quickly in low signal conditions
  - Some DRUK requirement changes added: program type visible while browsing in the MMI
  - FMHD seek change requested for narrow-band seek then switch to wide-band after seek stop – improves seek performance.

- Revision 1.1.0
  - SDK stability improvements
  - Updated front end calibration for new v2.1 BOM
  - ATS command support added
  - FMHD BER test added
  - DAB BER test accuracy improvement
  - Mode switch to the same mode no longer causes restart
  - Some DRUK requirement changes added: DLS clear, component info in service list.
  - EBU (previously included), UCS-2, and UTF-8 support for DAB

- Revision 1.0.0
  - DAB firmware update to v2.0.3
  - FM/FMHD firmware update to v3.0.11
  - Image separation by mode added as build option: typical builds include DAB+FM and FMHD
  - Flash update pass-through support added
  - Battery measurement/external power detection
  - On/Off improvements for information retention and faster restart
  - EBU character set support for LCD
  - SDK stability improvements
  - De-emphasis options added to MMI
  - Frequency spacing options added to MMI
  - UK only DAB band mode option added to MMI
  - DAB manual tune added to MMI
  - Added Signal Error metric to MMI for radio positioning

Document Revision History

- Revision 2.7
    - Updated the section “Platform options – platform_option.h”
        - OPTION__DAB_LINK_DISQUALIFIER_SNR_SETTLING_TIME_MS
        - OPTION__DAB_LINK_DISQUALIFIER_SNR_THRESHOLD
- Revision 2.6
    - Updated SDK release notes for 1.9.6 release
- Revision 2.5
    - Updated SDK release notes for 1.9.5 release
- Revision 2.4
    - Updated SDK release notes for 1.9.4 release
- Revision 2.3
    - Updated SDK release notes for 1.9.3 release
- Revision 2.2
    - Updated SDK release notes for 1.9.2 release
- Revision 2.1
    - Updated SDK release notes for 1.9.1 release
- Revision 2.0
    - Updated SDK release notes for 1.9.0 release
    - Updated the section “Platform option – platform_option.h”
        - OPTION__COMMAND_INPUT_MINIMAL_OPTIONS_SMALLER_SDK
        - OPTION__EXCLUDE_MFGTEST
        - OPTION__DAB_SUPPORT_DAB_DAB_LINKING_A3
- Revision 1.9
    - Updated SDK release notes for 1.8.0 release
- Revision 1.8
    - Updated SDK release notes for 1.7.4 release
    - Updated the section “Platform option – platform_option.h”
        - OPTION__RADIODNS
        - OPTION__NVMSPI_RATE_MHZ
    - Updated the section “si46xx_Firmware_API.h”
        - RETURN_CODE UpdateRadioDNS_FMRDS(radioDNS_fm* elements);
        - RETURN_CODE UpdateRadioDNS_DAB(radioDNS_dab* elements);
- Revision 1.7
    - Updated SDK release notes for 1.7.3 release
- Revision 1.6
    - Updated SDK release notes for 1.7.2 release
- Revision 1.5
    - Updated SDK release notes for 1.7.1 release
- Revision 1.4

    - Updated SDK release notes for the 1.7.0 release
    - Updated the section “Platform option – platform_option.h”
        - OPTION__IMAGETYPE__DAB_FM_AM
    - Updated the section “si46xx_Firmware_API.h”
        - RETURN_CODE fm_am_tune_freq__command(uint8_t open_loop, uint8_t force_wb, uint8_t tunemode, uint8_t injection, uint16_t freq, uint16_t antcap);
    - Add the section “si46xx_firmware_api.h”
        - RETURN_CODE am_seek_start__command(uint8_t open_loop, uint8_t force_wb, uint8_t tunemode, uint8_t injection, uint8_t seekup, uint8_t wrap, uint16_t antcap);
        - RETURN_CODE am_rsq_status__reply(am_rsq_status__data* replyData);
        - RETURN_CODE am_rsq_status__command(uint8_t rsqack, uint8_t attune, uint8_t cancel, uint8_t stcack);
- Revision 1.3
    - Updated SDK release notes for the 1.6.2.1 release
- Revision 1.2
    - Updated SDK release notes for the 1.6.2 release
- Revision 1.1
    - Updated SDK release notes for the 1.6.1 release
- Revision 1.0
    - Updated SDK release notes for the 1.6.0 release
    - Added the section “Software Stack – Slave Module”
    - Updated the section “Platform Options – platform_options.h”
        - OPTION__MCU_HARDWARE_SPI
        - OPTION__DAB_PRESETS_AS_FAVORITES
    - Updated the section “Firmware_API_Manager.h”
        - void FavoritesRemoveCurrentService_DAB();
        - void FavoritesBrowseOnly_DAB(uint8_t enable);
        - uint8_t FavoritesIsCurrentServiceAFavorite_DAB();
        - RETURN_CODE GetCurrentBrowseServiceString_DAB(uint8_t* service_name, uint8_t* service_name_encoding, uint8_t* service_pty);
- Revision 0.7
    - Updated the section “Platform Options – platform_options.h”
        - OPTION__FMHD_SEEK_HDLEVEL_THRESHOLD
        - OPTION__FMHD_SEEK_OFDM_SAMPLE_NUMBER
        - OPTION__OPERATE_AS_SLAVE_NO_MMI
        - OPTION__ADVANCED_METRICS
    - Updated the section “Firmware_API_Manager.h”
        - RETURN_CODE Test_StartFreeRunningBER_FMHD();
        - RETURN_CODE Test_GetFreeRunningBER_FMHD(uint8_t bitType, uint32_t* errorBits, uint32_t* totalBits);
- Revision 0.6
    - Updated the section “Platform Options – platform_options.h”

        - OPTION__DAB_SUPPORT_DAB_DAB_LINKING
        - OPTION__DAB_SERVICE_FOLLOWING_POLL_COUNT
        - OPTION__DAB_SERVICE_LIST_PERSISTENCE
    - Updated the section "SDK Callbacks"
        - UPDATED_SERVICE_LINKING_DAB
        - EVENT_FINDING_ALTERNATE_SERVICE
    - Updated the section "Firmware_API_Manager.h"
        - SaveServiceList_DAB();
        - LoadServiceList_DAB(uint8_t* lastPlayedIndex);
        - StartLastServiceByIndex(uint8_t serviceListIndex);
        - EraseServiceList_DAB();
    - Updated the section "Hal.c"
        - writePersistentStorage(…);
        - readPersistentStorage(…);
        - erasePersistentStorage();
    - Added new section for SDK revision history
- Revision 0.5
    - Seek is now handled by the SDK like the DAB BandScan. New callbacks and options were added to customize how the SDK operated during the managed scan.
        - "Platform Options – platform_options.h"
            - OPTION__FM_TIMEOUT_SEEK_MS
            - OPTION__FM_SEEK_CALLBACK_RATE_MS
        - "SDK Callbacks – SDK_Callbacks.h"
            - FM_SEEK_PROCESS_UPDATE
            - FM_SEEK_PROCESS_COMPLETE
    - FMHD mode now support alerts
        - "SDK Callbacks – SDK_Callbacks.h"
            - UPDATED_HD_ALERT
        - Data for String:
            - fmhd_alert_string* HDAlertGetPtr();
- Revision 0.4
    - Updated the section "Firmware_API_Manager.h"
        - Test_GetBER_FMHD()
- Revision 0.3
    - Updated the section "Platform Options – platform_options.h"
        - OPTION__PSD_FORMAT_ID3
    - Updated the section "SDK Callbacks"
        - CLEAR_DLS_COMMAND
- Revision 0.2
    - Updated the section "Platform Options – platform_options.h"
        - Added - 1) Image Type (select one):
        - Added - 25) OPTION__PROPERTY_DAB_VALID_RSSI_THRESHOLD

        - Renumbered items to give priority to new item #1.

    - Updated the section “Firmware_API_Manager.h”
        - GetFrequencyList_DAB()

- Initial revision: 0.1