SDRplay Limited.

Software Defined Radio API

Applications

Revision History

Revision	Release Date:	Reason for Change:	Originator
Up to 2.x	Various	Support up to 2.x API (See old API documentation)	APC
3.0	19th June 2018	Support 3.0 API (Service/Daemon)	APC
3.01	21st August 2018	Improvements for dual tuner and exit handling	APC
3.02	14th March 2019	New AGC scheme. Fixes to RSP1/RSPduo control	APC
3.03	9th April 2019	Updated heartbeat & comms systems	APC
3.04	8th July 2019	Updated for Diversity and other improvements	APC
3.06	22nd November 2019	Added RSPdx Support and extra error reporting	APC
3.07	December 2019	Added debug function, fixed RSP1A Bias-T operation	APC

Contents

1 Introduction

This document provides a description of the SDRplay Software Defined Radio API. This API provides a common interface to the RSP1, RSP2, RSP2pro, RSP1A, RSPduo and RSPdx from SDRplay Limited which make use of the Mirics USB bridge device (MSi2500) and the multi-standard tuner (MSi001).

From version 3.0 the API will be delivered as a service on Windows and as a daemon on non-Windows based systems. The service/daemon manages the control and data flow from each device to the end application.

The basic method of operation is in 3 main stages…

1. Set the API parameters based on the selected device
2. Initialise the device to start the stream
3. Change variables and perform an update to the API

This process can be seen in the example code in section 4.

The first function call must be to sdrplay_api_Open() and the last must be to sdrplay_api_Close() otherwise the service can be left in an unknown state.

In the header file descriptions in section 2, you will find the parameters that need to be set depending on the type of device. All parameters have a default setting.

The RSPduo can operate in single tuner mode (just like an RSP2/RSPdx), in dual tuner mode (both streams in a single instance) or in master/slave mode. If the RSPduo is already in use in master mode, then accessing the device again will mean that only slave mode is available. In master/slave mode, parameters that affect both tuners are only allowed to be set by the master.

Pages 4 and 5 of the RSPduo introduction document (https://www.sdrplay.com/wp-content/uploads/2018/05/RSPduo-Introduction-V3.pdf) present more information about valid states and supported sample rates for dual tuner operation.

The structures are defined in a hierarchy. For example, to enable the Bias-T on RSP1A, use…

```
deviceParams->rxChannelA->rsp1aTunerParams.biasTEnable = 1;
```

If this was before an initialisation, then there would be nothing else to do. To enable the Bias-T during stream, then after setting the variable, a call to the update function is required…

```
sdrplay_api_Update(chosenDevice->dev, chosenDevice->tuner,
                   sdrplay_api_Update_Rsp1a_BiasTControl, sdrplay_api_Update_Ext1_None);
```

2 API Data Types

The header files providing the definitions of the external data types and functions provided by this API are:

```
sdrplay_api.h
sdrplay_api_rx_channel.h
sdrplay_api_dev.h
sdrplay_api_tuner.h
sdrplay_api_control.h
sdrplay_api_rsp1a.h
sdrplay_api_rsp2.h
sdrplay_api_rspDuo.h
sdrplay_api_rspDx.h
sdrplay_api_callback.h
```

2.1 sdrplay_api.h

The top-level header file to be included in all applications making use of the sdrplay_api API. Defines the available functions and the structures used by them - further detail of sub-structures is contained in the subsequent sections describing the contents of each header file.

2.1.1 API Functions

```
sdrplay_api_ErrT        sdrplay_api_Open(void);
sdrplay_api_ErrT        sdrplay_api_Close(void);
sdrplay_api_ErrT        sdrplay_api_ApiVersion(float *apiVer);
sdrplay_api_ErrT        sdrplay_api_LockDeviceApi(void);
sdrplay_api_ErrT        sdrplay_api_UnlockDeviceApi(void);
sdrplay_api_ErrT        sdrplay_api_GetDevices(sdrplay_api_DeviceT *devices,
                                               unsigned int *numDevs,
                                               unsigned int maxDevs);
sdrplay_api_ErrT        sdrplay_api_SelectDevice(sdrplay_api_DeviceT *device);
sdrplay_api_ErrT        sdrplay_api_ReleaseDevice(sdrplay_api_DeviceT *device);
const char*             sdrplay_api_GetErrorString(sdrplay_api_ErrT err);
sdrplay_api_ErrorInfoT* sdrplay_api_GetLastError(sdrplay_api_DeviceT *device);
sdrplay_api_ErrT        sdrplay_api_DisableHeartbeat(void); // Must be called before
                                                            // sdrplay_api_SelectDevice()
sdrplay_api_ErrT        sdrplay_api_DebugEnable(HANDLE dev,
                                                sdrplay_api_DbgLvl_t dbgLvl);
sdrplay_api_ErrT        sdrplay_api_GetDeviceParams(HANDLE dev,
                                                    sdrplay_api_DeviceParamsT **deviceParams);
sdrplay_api_ErrT        sdrplay_api_Init(HANDLE dev,
                                         sdrplay_api_CallbackFnsT *callbackFns,
                                         void *cbContext);
sdrplay_api_ErrT        sdrplay_api_Uninit(HANDLE dev);
sdrplay_api_ErrT        sdrplay_api_Update(HANDLE dev,
                                           sdrplay_api_TunerSelectT tuner,
                                           sdrplay_api_ReasonForUpdateT reasonForUpdate,
                                           sdrplay_api_ReasonForUpdateExtension1T reasonForUpdateExt1);
sdrplay_api_ErrT        sdrplay_api_SwapRspDuoActiveTuner(HANDLE dev,
                                                          sdrplay_api_TunerSelectT *currentTuner,
                                                          sdrplay_api_RspDuo_AmPortSelectT tuner1AmPortSel);
sdrplay_api_ErrT        sdrplay_api_SwapRspDuoDualTunerModeSampleRate(HANDLE dev,
                                                                      double *currentSampleRate);
```

2.1.2 API Function Prototypes

```
typedef sdrplay_api_ErrT        (*sdrplay_api_Open_t)(void);
typedef sdrplay_api_ErrT        (*sdrplay_api_Close_t)(void);
typedef sdrplay_api_ErrT        (*sdrplay_api_ApiVersion_t)(float *apiVer);
typedef sdrplay_api_ErrT        (*sdrplay_api_LockDeviceApi_t)(void);
typedef sdrplay_api_ErrT        (*sdrplay_api_UnlockDeviceApi_t)(void);
typedef sdrplay_api_ErrT        (*sdrplay_api_GetDevices_t)(sdrplay_api_DeviceT *devices,
                                     unsigned int *numDevs,
                                     unsigned int maxDevs);
typedef sdrplay_api_ErrT        (*sdrplay_api_SelectDevice_t)(sdrplay_api_DeviceT *device);
typedef sdrplay_api_ErrT        (*sdrplay_api_ReleaseDevice_t)(sdrplay_api_DeviceT *device);
typedef const char*             (*sdrplay_api_GetErrorString_t)(sdrplay_api_ErrT err);
typedef sdrplay_api_ErrorInfoT* (*sdrplay_api_GetLastError_t)(sdrplay_api_DeviceT *device);
typedef sdrplay_api_ErrT        (*sdrplay_api_DisableHeartbeat_t)(void);
typedef sdrplay_api_ErrT        (*sdrplay_api_DebugEnable_t)(HANDLE dev,
                                     sdrplay_api_DbgLvl_t dbgLvl);
typedef sdrplay_api_ErrT        (*sdrplay_api_GetDeviceParams_t)(HANDLE dev,
                                     sdrplay_api_DeviceParamsT **deviceParams);
typedef sdrplay_api_ErrT        (*sdrplay_api_Init_t)(HANDLE dev,
                                     sdrplay_api_CallbackFnsT *callbackFns,
                                     void *cbContext);
typedef sdrplay_api_ErrT        (*sdrplay_api_Uninit_t)(HANDLE dev);
typedef sdrplay_api_ErrT        (*sdrplay_api_Update_t)(HANDLE dev,
                                     sdrplay_api_TunerSelectT tuner,
                                     sdrplay_api_ReasonForUpdateT reasonForUpdate,
                                     sdrplay_api_ReasonForUpdateExtension1T reasonForUpdateExt1);
typedef sdrplay_api_ErrT        (*sdrplay_api_SwapRspDuoActiveTuner_t)(HANDLE dev,
                                      sdrplay_api_TunerSelectT *tuner,
                                      sdrplay_api_RspDuo_AmPortSelectT tuner1AmPortSel);
typedef sdrplay_api_ErrT        (*sdrplay_api_SwapRspDuoDualTunerModeSampleRate_t)(
                                      double *currentSampleRate);
```

2.1.3 Constant Definitions

```
#define SDRPLAY_API_VERSION    (float)(3.07)
#define SDRPLAY_MAX_DEVICES           (16)    // Maximum devices supported by the API
#define SDRPLAY_MAX_TUNERS_PER_DEVICE (2)     // Maximum number of tuners available on one device

#define SDRPLAY_MAX_SER_NO_LEN        (64)    // Maximum length of device serial numbers
#define SDRPLAY_MAX_ROOT_NM_LEN       (32)    // Maximum length of device names

// Supported device IDs
#define SDRPLAY_RSP1_ID               (1)
#define SDRPLAY_RSP1A_ID              (255)
#define SDRPLAY_RSP2_ID               (2)
#define SDRPLAY_RSPduo_ID             (3)
#define SDRPLAY_RSPdx_ID              (4)
```

2.1.4 Enumerated Data Types

Error Code Enumerated Type:

```
typedef enum
{
    sdrplay_api_Success               = 0,
    sdrplay_api_Fail                  = 1,
    sdrplay_api_InvalidParam          = 2,
    sdrplay_api_OutOfRange            = 3,
    sdrplay_api_GainUpdateError       = 4,
    sdrplay_api_RfUpdateError         = 5,
    sdrplay_api_FsUpdateError         = 6,
    sdrplay_api_HwError               = 7,
    sdrplay_api_AliasingError         = 8,
    sdrplay_api_AlreadyInitialised    = 9,
    sdrplay_api_NotInitialised        = 10,
    sdrplay_api_NotEnabled            = 11,
    sdrplay_api_HwVerError            = 12,
    sdrplay_api_OutOfMemError         = 13,
    sdrplay_api_ServiceNotResponding  = 14,
    sdrplay_api_StartPending          = 15,
    sdrplay_api_StopPending           = 16,
    sdrplay_api_InvalidMode           = 17,
    sdrplay_api_FailedVerification1   = 18,
    sdrplay_api_FailedVerification2   = 19,
    sdrplay_api_FailedVerification3   = 20,
    sdrplay_api_FailedVerification4   = 21,
    sdrplay_api_FailedVerification5   = 22,
    sdrplay_api_FailedVerification6   = 23,
    sdrplay_api_InvalidServiceVersion = 24
} sdrplay_api_ErrT;
```

Debug Level Enumerated Type:

```
typedef enum
{
    sdrplay_api_DbgLvl_Disable       = 0,
    sdrplay_api_DbgLvl_Verbose       = 1,
    sdrplay_api_DbgLvl_Warning       = 2,
    sdrplay_api_DbgLvl_Error         = 3,
    sdrplay_api_DbgLvl_Message       = 4,
} sdrplay_api_DbgLvl_t;
```

Update Enumerated Type:

```
typedef enum
{
    sdrplay_api_Update_None                         = 0x00000000,

    // Reasons for master only mode
    sdrplay_api_Update_Dev_Fs                       = 0x00000001,
    sdrplay_api_Update_Dev_Ppm                      = 0x00000002,
    sdrplay_api_Update_Dev_SyncUpdate               = 0x00000004,
    sdrplay_api_Update_Dev_ResetFlags               = 0x00000008,

    sdrplay_api_Update_Rsp1a_BiasTControl           = 0x00000010,
    sdrplay_api_Update_Rsp1a_RfNotchControl         = 0x00000020,
    sdrplay_api_Update_Rsp1a_RfDabNotchControl      = 0x00000040,

    sdrplay_api_Update_Rsp2_BiasTControl            = 0x00000080,
    sdrplay_api_Update_Rsp2_AmPortSelect            = 0x00000100,
    sdrplay_api_Update_Rsp2_AntennaControl          = 0x00000200,
    sdrplay_api_Update_Rsp2_RfNotchControl          = 0x00000400,
    sdrplay_api_Update_Rsp2_ExtRefControl           = 0x00000800,

    sdrplay_api_Update_RspDuo_ExtRefControl         = 0x00001000,

    sdrplay_api_Update_Master_Spare_1               = 0x00002000,
    sdrplay_api_Update_Master_Spare_2               = 0x00004000,

    // Reasons for master and slave mode
    // Note: sdrplay_api_Update_Tuner_Gr MUST be the first value defined in this section!
    sdrplay_api_Update_Tuner_Gr                     = 0x00008000,
    sdrplay_api_Update_Tuner_GrLimits               = 0x00010000,
    sdrplay_api_Update_Tuner_Frf                    = 0x00020000,
    sdrplay_api_Update_Tuner_BwType                 = 0x00040000,
    sdrplay_api_Update_Tuner_IfType                 = 0x00080000,
    sdrplay_api_Update_Tuner_DcOffset               = 0x00100000,
    sdrplay_api_Update_Tuner_LoMode                 = 0x00200000,

    sdrplay_api_Update_Ctrl_DCoffsetIQimbalance     = 0x00400000,
    sdrplay_api_Update_Ctrl_Decimation              = 0x00800000,
    sdrplay_api_Update_Ctrl_Agc                     = 0x01000000,
    sdrplay_api_Update_Ctrl_AdsbMode                = 0x02000000,
    sdrplay_api_Update_Ctrl_OverloadMsgAck          = 0x04000000,

    sdrplay_api_Update_RspDuo_BiasTControl          = 0x08000000,
    sdrplay_api_Update_RspDuo_AmPortSelect          = 0x10000000,
    sdrplay_api_Update_RspDuo_Tuner1AmNotchControl  = 0x20000000,
    sdrplay_api_Update_RspDuo_RfNotchControl        = 0x40000000,
    sdrplay_api_Update_RspDuo_RfDabNotchControl     = 0x80000000,

} sdrplay_api_ReasonForUpdateT;

typedef enum
{
    sdrplay_api_Update_Ext1_None                    = 0x00000000,

    // Reasons for master only mode
    sdrplay_api_Update_RspDx_HdrEnable              = 0x00000001,
    sdrplay_api_Update_RspDx_BiasTControl           = 0x00000002,
    sdrplay_api_Update_RspDx_AntennaControl         = 0x00000004,
    sdrplay_api_Update_RspDx_RfNotchControl         = 0x00000008,
    sdrplay_api_Update_RspDx_RfDabNotchControl      = 0x00000010,
    sdrplay_api_Update_RspDx_HdrBw                  = 0x00000020,

    // Reasons for master and slave mode
} sdrplay_api_ReasonForUpdateExtension1T;
```

2.1.5 Data Structures

Device enumeration structure:

```
typedef struct
{
    char SerNo[SDRPLAY_MAX_SER_NO_LEN];      // Set by the API on return from
                                             // sdrplay_api_GetDevices() contains the serial
                                             // number of the device
    unsigned char hwVer;                     // Set by the API on return from
                                             // sdrplay_api_GetDevices() contains the Hardware
                                             // version of the device
    sdrplay_api_TunerSelectT tuner;          // Set by the API on return from
                                             // sdrplay_api_GetDevices() indicating which tuners
                                             // are available.
                                             // Set by the application and used during
                                             // sdrplay_api_SelectDevice() to indicate which
                                             // tuner(s) is to be used.
    sdrplay_api_RspDuoModeT rspDuoMode;      // Set by the API on return from
                                             // sdrplay_api_GetDevices() for RSPduo devices
                                             // indicating which modes are available.
                                             // Set by the application and used during
                                             // sdrplay_api_SelectDevice() for RSPduo device to
                                             // indicate which mode is to be used.
    double rspDuoSampleFreq;                 // Set by the API on return from
                                             // sdrplay_api_GetDevices() for RSPduo slaves
                                             // indicating the sample rate previously set by the
                                             // master.
                                             // Set by the application and used during
                                             // sdrplay_api_SelectDevice() by RSPduo masters to
                                             // indicate required sample rate.
    HANDLE dev;                              // Set by the API on return from
                                             // sdrplay_api_SelectDevice() for use in subsequent
                                             // calls to the API. Do not alter!
} sdrplay_api_DeviceT;
```

Device Parameters Structure:

```
typedef struct
{
    sdrplay_api_DevParamsT       *devParams; // All parameters for a single device (except tuner
                                             // parameters)
    sdrplay_api_RxChannelParamsT *rxChannelA;// First tuner parameters for all devices
    sdrplay_api_RxChannelParamsT *rxChannelB;// Second tuner parameters for RSPduo
} sdrplay_api_DeviceParamsT;
```

Extended Error Message Structure

```
typedef struct
{
    char file[256];                          // API file where the error occurred
    char function[256];                      // API function that the error occurred in
    int  line;                               // line number that the error occurred on
    char message[1024];                      // Readable API error message to display
} sdrplay_api_ErrorInfoT;
```

2.2 sdrplay_api_rx_channel.h

2.2.1 Data Structures

Receive Channel Structure:

```
typedef struct
{
    sdrplay_api_TunerParamsT       tunerParams;
    sdrplay_api_ControlParamsT     ctrlParams;
    sdrplay_api_Rsp1aTunerParamsT  rsp1aTunerParams;
    sdrplay_api_Rsp2TunerParamsT   rsp2TunerParams;
    sdrplay_api_RspDuoTunerParamsT rspDuoTunerParams;
    sdrplay_api_RspDxTunerParamsT  rspDxTunerParams;
} sdrplay_api_RxChannelParamsT;
```

2.3 sdrplay_api_dev.h

Provides definitions of non-tuner related parameters

2.3.1 Enumerated Data Types

Transfer Mode Enumerated Type:

```
typedef enum
{
    sdrplay_api_ISOCH = 0,
    sdrplay_api_BULK  = 1
} sdrplay_api_TransferModeT;
```

2.3.2 Data Structures

Default values for each parameter are given - for sub-structures, the default values will be given in the structure definition for that type.

ADC Sampling Frequency Parameters Structure:

```
typedef struct
{
    double fsHz;                        // default: 2000000.0
    unsigned char syncUpdate;           // default: 0
    unsigned char reCal;                // default: 0
} sdrplay_api_FsFreqT;
```

Synchronous Update Parameters Structure:

```
typedef struct
{
    unsigned int sampleNum;             // default: 0
    unsigned int period;                // default: 0
} sdrplay_api_SyncUpdateT;
```

Reset Update Operations Structure:

```
typedef struct
{
    unsigned char resetGainUpdate;      // default: 0
    unsigned char resetRfUpdate;        // default: 0
    unsigned char resetFsUpdate;        // default: 0
} sdrplay_api_ResetFlagsT;
```

Non-Receive Channel Related Device Parameters:

```
typedef struct
{
    double                    ppm;           // default: 0.0
    sdrplay_api_FsFreqT       fsFreq;
    sdrplay_api_SyncUpdateT   syncUpdate;
    sdrplay_api_ResetFlagsT   resetFlags;
    sdrplay_api_TransferModeT mode;          // default: sdrplay_api_ISOCH
    unsigned int              samplesPerPkt; // default: 0 (output param)
    sdrplay_api_Rsp1aParamsT  rsp1aParams;
    sdrplay_api_Rsp2ParamsT   rsp2Params;
    sdrplay_api_RspDuoParamsT rspDuoParams;
    sdrplay_api_RspDxParamsT  rspDxParams;
} sdrplay_api_DevParamsT;
```

2.4 sdrplay_api_tuner.h

2.4.1 Constant Definitions

```
#define MAX_BB_GR                   (59)       // Maximum baseband gain reduction
```

2.4.2 Enumerated Data Types

Bandwidth Enumerated Type:

```
typedef enum
{
    sdrplay_api_BW_Undefined = 0,
    sdrplay_api_BW_0_200     = 200,
    sdrplay_api_BW_0_300     = 300,
    sdrplay_api_BW_0_600     = 600,
    sdrplay_api_BW_1_536     = 1536,
    sdrplay_api_BW_5_000     = 5000,
    sdrplay_api_BW_6_000     = 6000,
    sdrplay_api_BW_7_000     = 7000,
    sdrplay_api_BW_8_000     = 8000
} sdrplay_api_Bw_MHzT;
```

IF Enumerated Type:

```
typedef enum
{
    sdrplay_api_IF_Undefined = -1,
    sdrplay_api_IF_Zero      = 0,
    sdrplay_api_IF_0_450     = 450,
    sdrplay_api_IF_1_620     = 1620,
    sdrplay_api_IF_2_048     = 2048
} sdrplay_api_If_kHzT;
```

LO Enumerated Type:

```
typedef enum
{
    sdrplay_api_LO_Undefined = 0,
    sdrplay_api_LO_Auto      = 1,
    sdrplay_api_LO_120MHz    = 2,
    sdrplay_api_LO_144MHz    = 3,
    sdrplay_api_LO_168MHz    = 4
} sdrplay_api_LoModeT;
```

Minimum Gain Enumerated Type:

```
typedef enum
{
    sdrplay_api_EXTENDED_MIN_GR = 0,
    sdrplay_api_NORMAL_MIN_GR   = 20
} sdrplay_api_MinGainReductionT;
```

Tuner Selected Enumerated Type:

```
typedef enum
{
    sdrplay_api_Tuner_Neither  = 0,
    sdrplay_api_Tuner_A        = 1,
    sdrplay_api_Tuner_B        = 2,
    sdrplay_api_Tuner_Both     = 3,
} sdrplay_api_TunerSelectT;
```

2.4.3 Data Structures

Current Gain Value Structure:

```
typedef struct
{
    float curr;
    float max;
    float min;
} sdrplay_api_GainValuesT;
```

Gain Setting Parameter Structure:

```
typedef struct
{
    int                             gRdB;       // default: 50
    unsigned char                   LNAstate;   // default: 0
    unsigned char                   syncUpdate; // default: 0
    sdrplay_api_MinGainReductionT minGr;      // default: sdrplay_api_NORMAL_MIN_GR
    sdrplay_api_GainValuesT       gainVals;   // output parameter
} sdrplay_api_GainT;
```

RF Frequency Parameter Structure:

```
typedef struct
{
    double        rfHz;                        // default: 200000000.0
    unsigned char syncUpdate;                  // default: 0
} sdrplay_api_RfFreqT;
```

DC Calibration Paramter Structure:

```
typedef struct
{
    unsigned char dcCal;                     // default: 3 (Periodic mode)
    unsigned char speedUp;                   // default: 0 (No speedup)
    int           trackTime;                 // default: 1    (=> time in uSec = (dcCal * 3 *
trackTime)       = 9uSec)
    int           refreshRateTime;           // default: 2048 (=> time in uSec = (dcCal * 3 *
refreshRateTime) = 18432uSec)
} sdrplay_api_DcOffsetTunerT;
```

Tuner Parameter Structure:

```
typedef struct
{
    sdrplay_api_Bw_MHzT        bwType;       // default: sdrplay_api_BW_0_200
    sdrplay_api_If_kHzT        ifType;       // default: sdrplay_api_IF_Zero (master) or
                                             //          sdrplay_api_IF_0_450 (slave)
    sdrplay_api_LoModeT        loMode;       // default: sdrplay_api_LO_Auto
    sdrplay_api_GainT          gain;
    sdrplay_api_RfFreqT        rfFreq;
    sdrplay_api_DcOffsetTunerT dcOffsetTuner;
} sdrplay_api_TunerParamsT;
```

2.5 sdrplay_api_control.h

2.5.1 Enumerated Data Types

AGC Loop Bandwidth Enumerated Type:

```
typedef enum
{
    sdrplay_api_AGC_DISABLE  = 0,
    sdrplay_api_AGC_100HZ    = 1,
    sdrplay_api_AGC_50HZ     = 2,
    sdrplay_api_AGC_5HZ      = 3,
    sdrplay_api_AGC_CTRL_EN  = 4      // Latest AGC scheme (see AGC control parameters structure)
} sdrplay_api_AgcControlT;
```

ADS-B Configuration Enumerated Type:

```
typedef enum
{
    sdrplay_api_ADSB_DECIMATION                  = 0,
    sdrplay_api_ADSB_NO_DECIMATION_LOWPASS       = 1,
    sdrplay_api_ADSB_NO_DECIMATION_BANDPASS_2MHZ = 2,
    sdrplay_api_ADSB_NO_DECIMATION_BANDPASS_3MHZ = 3
} sdrplay_api_AdsbModeT;
```

2.5.2 Data Structures

DC Offset Control Parameters Structure:

```
typedef struct
{
    unsigned char DCenable;               // default: 1
    unsigned char IQenable;               // default: 1
} sdrplay_api_DcOffsetT;
```

Decimation Control Parameters Structure:

```
typedef struct
{
    unsigned char enable;                 // default: 0
    unsigned char decimationFactor;       // default: 1
    unsigned char wideBandSignal;         // default: 0
} sdrplay_api_DecimationT;
```

AGC Control Parameters Structure:

```
typedef struct
{
    sdrplay_api_AgcControlT enable;             // default: sdrplay_api_AGC_50HZ
    int                     setPoint_dBfs;      // default: -60
    unsigned short          attack_ms;          // default: 0
    unsigned short          decay_ms;           // default: 0
    unsigned short          decay_delay_ms;     // default: 0
    unsigned short          decay_threshold_dB; // default: 0
    int                     syncUpdate;         // default: 0
} sdrplay_api_AgcT;
```

Control Parameters Structure:

```
typedef struct
{
    sdrplay_api_DcOffsetT   dcOffset;
    sdrplay_api_DecimationT decimation;
    sdrplay_api_AgcT        agc;
    sdrplay_api_AdsbModeT   adsbMode;      //default: sdrplay_api_ADSB_DECIMATION
} sdrplay_api_ControlParamsT;
```

2.5.3 Valid Setpoint Values vs Sample Rate

-72 <= setpoint_dBfs <= -20dB (or 0dB depending on setting of sdrplay_api_GainT.minGr) for sample rates < 8.064 MSPS
-60 <= setpoint_dBfs <= -20dB (or 0dB depending on setting of sdrplay_api_GainT.minGr) for sample rates in the range 8.064 – 9.216 MSPS
-48 <= setpoint_dBfs <= -20dB (or 0dB depending on setting of sdrplay_api_GainT.minGr) for sample rates > 9.216 MSPS)

2.6 sdrplay_api_rsp1a.h

2.6.1 Constant Definitions

```
#define RSPIA_NUM_LNA_STATES             10   // Number of LNA states in all bands (except where
defined differently below)
#define RSPIA_NUM_LNA_STATES_AM          7    // Number of LNA states in AM band
#define RSPIA_NUM_LNA_STATES_LBAND       9    // Number of LNA states in L band
```

Data Structures

RSP1A RF Notch Control Parameters Structure:

```
typedef struct
{
    unsigned char rfNotchEnable;            // default: 0
    unsigned char rfDabNotchEnable;         // default: 0
} sdrplay_api_Rsp1aParamsT;
```

RSP1A Bias-T Control Parameters Structure:

```
typedef struct
{
    unsigned char biasTEnable;              // default: 0
} sdrplay_api_Rsp1aTunerParamsT;
```

2.7 sdrplay_api_rsp2.h

2.7.1 Constant Definitions

```
#define RSPII_NUM_LNA_STATES             9    // Number of LNA states in in all bands (except
where defined differently below)
#define RSPII_NUM_LNA_STATES_AMPORT      5    // Number of LNA states for HiZ port
#define RSPII_NUM_LNA_STATES_420MHZ      6    // Number of LNA states in 420MHz band
```

2.7.2 Enumerated Data Types

RSP2 Antenna Selection Enumerated Type:

```
typedef enum
{
    sdrplay_api_Rsp2_ANTENNA_A = 5,
    sdrplay_api_Rsp2_ANTENNA_B = 6,
} sdrplay_api_Rsp2_AntennaSelectT;
```

RSP2 AM Port Enumerated Type:

```
typedef enum
{
    sdrplay_api_Rsp2_AMPORT_1 = 1,
    sdrplay_api_Rsp2_AMPORT_2 = 0,
} sdrplay_api_Rsp2_AmPortSelectT;
```

2.7.3 Data Structures

RSP2 External Reference Control Parameters Structure:

```
typedef struct
{
    unsigned char extRefOutputEn;                   // default: 0
} sdrplay_api_Rsp2ParamsT;
```

RSP2 Tuner Parameters Structure:

```
typedef struct
{
    unsigned char                   biasTEnable;    // default: 0
    sdrplay_api_Rsp2_AmPortSelectT  amPortSel;      // default: sdrplay_api_Rsp2_AMPORT_2
    sdrplay_api_Rsp2_AntennaSelectT antennaSel;     // default: sdrplay_api_Rsp2_ANTENNA_A
    unsigned char                   rfNotchEnable;  // default: 0
} sdrplay_api_Rsp2TunerParamsT;
```

2.8 sdrplay_api_rspDuo.h

2.8.1 Constant Definitions

```
#define RSPDUO_NUM_LNA_STATES            10   // Number of LNA states in all bands (except where
defined differently below)
#define RSPDUO_NUM_LNA_STATES_AMPORT     5    // Number of LNA states for HiZ port
#define RSPDUO_NUM_LNA_STATES_AM         7    // Number of LNA states in AM band
#define RSPDUO_NUM_LNA_STATES_LBAND      9    // Number of LNA states in L band
```

2.8.2 Enumerated Data Types

RSPduo Operating Mode Enumerated Type:

```
typedef enum
{
    sdrplay_api_RspDuoMode_Unknown      = 0,
    sdrplay_api_RspDuoMode_Single_Tuner = 1,
    sdrplay_api_RspDuoMode_Dual_Tuner   = 2,
    sdrplay_api_RspDuoMode_Master       = 4,
    sdrplay_api_RspDuoMode_Slave        = 8,
} sdrplay_api_RspDuoModeT;
```

RSPduo AM Port Enumerated Type:

```
typedef enum
{
    sdrplay_api_RspDuo_AMPORT_1 = 1,
    sdrplay_api_RspDuo_AMPORT_2 = 0,
} sdrplay_api_RspDuo_AmPortSelectT;
```

2.8.3 Data Structures

RSPduo External Reference Control Parameters Structure:

```
typedef struct
{
    int extRefOutputEn;                                 // default: 0
} sdrplay_api_RspDuoParamsT;
```

RSPduo Tuner Parameters Structure:

```
typedef struct
{
    unsigned char                   biasTEnable;         // default: 0
    sdrplay_api_RspDuo_AmPortSelectT tuner1AmPortSel;    // default: sdrplay_api_RspDuo_AMPORT_2
    unsigned char                   tuner1AmNotchEnable; // default: 0
    unsigned char                   rfNotchEnable;       // default: 0
    unsigned char                   rfDabNotchEnable;    // default: 0
} sdrplay_api_RspDuoTunerParamsT;
```

2.9 sdrplay_api_rspDx.h

2.9.1 Constant Definitions

```
#define RSPDX_NUM_LNA_STATES               28   // Number of LNA states in all bands (except
where defined differently below)
#define RSPDX_NUM_LNA_STATES_AMPORT2_0_12  19   // Number of LNA states when using AM Port 2
between 0 and 12MHz
#define RSPDX_NUM_LNA_STATES_AMPORT2_12_60 20   // Number of LNA states when using AM Port 2
between 12 and 60MHz
#define RSPDX_NUM_LNA_STATES_VHF_BAND3     27   // Number of LNA states in VHF and Band3
#define RSPDX_NUM_LNA_STATES_420MHZ        21   // Number of LNA states in 420MHz band
#define RSPDX_NUM_LNA_STATES_LBAND         19   // Number of LNA states in L-band
#define RSPDX_NUM_LNA_STATES_DX            26   // Number of LNA states in DX path
```

2.9.2 Enumerated Data Types

RSPdx Antenna Selection Enumerated Type:

```
typedef enum
{
    sdrplay_api_RspDx_ANTENNA_A = 0,
    sdrplay_api_RspDx_ANTENNA_B = 1,
    sdrplay_api_RspDx_ANTENNA_C = 2,
} sdrplay_api_RspDx_AntennaSelectT;
```

RSPdx HDR Mode Bandwidth Enumerated Type:

```
typedef enum
{
    sdrplay_api_RspDx_HDRMODE_BW_0_200  = 0,
    sdrplay_api_RspDx_HDRMODE_BW_0_500  = 1,
    sdrplay_api_RspDx_HDRMODE_BW_1_200  = 2,
    sdrplay_api_RspDx_HDRMODE_BW_1_700  = 3,
} sdrplay_api_RspDx_HdrModeBwT;
```

2.9.3 Data Structures

RSPdx Control Parameters Structure:

```
typedef struct
{
    unsigned char hdrEnable;                            // default: 0
    unsigned char biasTEnable;                          // default: 0
    sdrplay_api_RspDx_AntennaSelectT antennaSel;        // default: sdrplay_api_RspDx_ANTENNA_A
    unsigned char rfNotchEnable;                        // default: 0
    unsigned char rfDabNotchEnable;                     // default: 0
} sdrplay_api_RspDxParamsT;
```

RSPdx Tuner Parameters Structure:

```
typedef struct
{
    sdrplay_api_RspDx_HdrModeBwT hdrBw;            // default: sdrplay_api_RspDx_HDRMODE_BW_1_700
} sdrplay_api_RspDxTunerParamsT;
```

2.10 sdrplay_api_callback.h

2.10.1 Enumerated Data Types

Power Overload Event Enumerated Type:

```
typedef enum
{
    sdrplay_api_Overload_Detected      = 0,
    sdrplay_api_Overload_Corrected     = 1,
} sdrplay_api_PowerOverloadCbEventIdT;
```

RSPduo Event Enumerated Type:

```
typedef enum
{
    sdrplay_api_MasterInitialised      = 0,
    sdrplay_api_SlaveAttached          = 1,
    sdrplay_api_SlaveDetached          = 2,
    sdrplay_api_SlaveInitialised       = 3,
    sdrplay_api_SlaveUninitialised     = 4,
    sdrplay_api_MasterDllDisappeared   = 5,
    sdrplay_api_SlaveDllDisappeared    = 6,
} sdrplay_api_RspDuoModeCbEventIdT;
```

Events Enumerated Type:

```
typedef enum
{
    sdrplay_api_GainChange             = 0,
    sdrplay_api_PowerOverloadChange    = 1,
    sdrplay_api_DeviceRemoved          = 2,
    sdrplay_api_RspDuoModeChange       = 3,
} sdrplay_api_EventT;
```

2.10.2 Data Structures

Event Callback Structure:

```
typedef struct
{
    unsigned int gRdB;
    unsigned int lnaGRdB;
    double       currGain;
} sdrplay_api_GainCbParamT;
```

Power Overload Structure:

```
typedef struct
{
    sdrplay_api_PowerOverloadCbEventIdT powerOverloadChangeType;
} sdrplay_api_PowerOverloadCbParamT;
```

RSPduo Structure:

```
typedef struct
{
    sdrplay_api_RspDuoModeCbEventIdT modeChangeType;
} sdrplay_api_RspDuoModeCbParamT;
```

Combination of Event Callback Structures:

```
typedef union
{
    sdrplay_api_GainCbParamT          gainParams;
    sdrplay_api_PowerOverloadCbParamT powerOverloadParams;
    sdrplay_api_RspDuoModeCbParamT    rspDuoModeParams;
} sdrplay_api_EventParamsT;
```

Streaming Data Parameter Callback Structure:

```
typedef struct
{
    unsigned int firstSampleNum;
    int          grChanged;
    int          rfChanged;
    int          fsChanged;
    unsigned int numSamples;
} sdrplay_api_StreamCbParamsT;
```

Callback Function Definition Structure:

```
typedef struct
{
    sdrplay_api_StreamCallback_t StreamACbFn;
    sdrplay_api_StreamCallback_t StreamBCbFn;
    sdrplay_api_EventCallback_t  EventCbFn;
} sdrplay_api_CallbackFnsT;
```

2.10.3 Callback Function Prototypes

```
typedef void (*sdrplay_api_StreamCallback_t)(short *xi,
                                             short *xq,
                                             sdrplay_api_StreamCbParamsT *params,
                                             unsigned int numSamples,
                                             unsigned int reset,
                                             void *cbContext);
typedef void (*sdrplay_api_EventCallback_t)(sdrplay_api_EventT eventId,
                                            sdrplay_api_TunerSelectT tuner,
                                            sdrplay_api_EventParamsT *params,
                                            void *cbContext);
```

3 Function Descriptions

3.1 sdrplay_api_Open

```
sdrplay_api_ErrT sdrplay_api_Open(void)
```

Description:

Opens the API and configures the API for use. This function must be called before any other API function.

Parameters:

`void`	No parameters

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	API successfully opened
	`sdrplay_api_Fail`	API failed to open

3.2 sdrplay_api_Close

```
sdrplay_api_ErrT sdrplay_api_Close(void)
```

Description:

Tidies up and closes the API. After calling this function it is no longer possible to access other API functions until sdrplay_api_Open() is successfully called again.

Parameters:

`void`	No parameters

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	API successfully closed

3.3 sdrplay_api_ApiVersion

```
sdrplay_api_ErrT sdrplay_api_ApiVersion(float *apiVer)
```

Description:

This function checks that the version of the include file used to compile the application is consistent with the API version being used.

Parameters:

`apiVer`	Pointer to a float which returns the version of the API

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_InvalidParam`	NULL pointer
	`sdrplay_api_InvalidServiceVersion`	Service version doesn't match
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.4 sdrplay_api_LockDeviceApi

```
sdrplay_api_ErrT sdrplay_api_LockDeviceApi(void)
```

Description:

Attempts to lock the API for exclusive use of the current application. Once locked, no other applications will be able to use the API. Typically used to lock the API prior to calling sdrplay_api_GetDevices() to ensure only one application can select a given device. After completing device selection using sdrplay_api_SelectDevice(), sdrplay_api_UnlockDeviceApi() can be used to release the API. May also be used prior to calling sdrplay_api_ReleaseDevice() if it is necessary to reselect the same device.

Parameters:

`void`	No parameters

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.5 sdrplay_api_UnlockDeviceApi

```
sdrplay_api_ErrT sdrplay_api_UnlockDeviceApi(void)
```

Description:

See description for sdrplay_api_LockDeviceApi().

Parameters:

`none`	No parameters

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.6 sdrplay_api_GetDevices

```
sdrplay_api_ErrT sdrplay_api_GetDevices(sdrplay_api_DeviceT *devices,
                                        unsigned int *numDevs,
                                        unsigned int maxDevs)
```

Description:

This function returns a list of all available devices (up to a maximum defined by maxDev parameter). Once the list has been retrieved, a device can be selected based on the required characteristics.

Parameters:

`devices`	Pointer to an array of device enumeration structures used to return the list of available devices
`numDevs`	Pointer to a variable which on return will indicate the number of available devices
`maxDevs`	Specifies the maximum number of devices that can be returned in the list (size of array of device enumeration structures)

Return:

`sdrplay_api_ErrT` Error code as defined below:

`sdrplay_api_Success`	Successful completion
`sdrplay_api_Fail`	Command failed
`sdrplay_api_InvalidParam`	NULL pointer
`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.7 sdrplay_api_SelectDevice

```
sdrplay_api_ErrT sdrplay_api_SelectDevice(sdrplay_api_DeviceT *device)
```

Description:

Once a device is selected from the list of devices returned in sdrplay_api_GetDevices(), and the additional information for the device configured (see the definitions of sdrplay_api_DeviceT for more information), this function will select the device. Once a device has been selected, it is no longer available for other applications (unless the device is a RSPduo in master/slave mode). On return from this call, the sdrplay_api_DeviceT structure passed in contains a handle that can be used in subsequent calls to the API.

Parameters:

`device`	Pointer to the sdrplay_api_DeviceT structure for the selected device

Return:

`sdrplay_api_ErrT` Error code as defined below:

`sdrplay_api_Success`	Successful completion
`sdrplay_api_Fail`	Command failed
`sdrplay_api_InvalidParam`	NULL pointer
`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.8 sdrplay_api_ReleaseDevice

```
sdrplay_api_ErrT sdrplay_api_ReleaseDevice(sdrplay_api_DeviceT *device)
```

Description:

Releases a device and makes that device available for other applications.

Parameters:

`device`	Pointer to the sdrplay_api_DeviceT structure for the device to be released

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_InvalidParam`	NULL pointer
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.9 sdrplay_api_GetErrorString

```
const char* sdrplay_api_GetErrorString(sdrplay_api_ErrT err)
```

Description:

Upon receipt of an error code, a print friendly error string can be obtained using the function. The returned pointer is a pointer to a static array and does not need to be freed.

Parameters:

`err`	Error code to be converted to a string.

Return:

`const char *`	Pointer to a string containing the error definition

3.10 sdrplay_api_GetLastError

```
sdrplay_api_ErrorInfoT* sdrplay_api_GetLastError(sdrplay_api_DeviceT *device)
```

Description:

Upon receipt of an error code, extended information on the location and reason for the error can be obtained using the function. The returned pointer is a pointer to a static array and does not need to be freed.

Parameters:

`device`	Pointer to the sdrplay_api_DeviceT structure for the device currently used

Return:

`sdrplay_api_ErrorInfoT *`	Pointer to a structure containing the last error information

3.11 sdrplay_api_DisableHeartbeat

```
sdrplay_api_ErrT sdrplay_api_DisableHeartbeat(void)
```

Description:

Debug only function. Allows code to be stepped through without API threads timing out. MUST be called before sdrplay_api_SelectDevice is called.

Parameters:

`void`	No parameters

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Failure to call sdrplay_api_LockDeviceApi

3.12 sdrplay_api_DebugEnable

```
sdrplay_api_ErrT sdrplay_api_DebugEnable(HANDLE dev, sdrplay_api_DbgLvl_t dbgLvl)
```

Description:

Enable or disable debug output logging. This logging can help with debugging issues but will increase the processing load and in some extreme cases, may cause data dropout.

Parameters:

`dev`	Handle of selected device from current device enumeration structure (can be NULL for reduced logging prior to selecting a device)
`dbgLvl`	Specifies the level of debug detail from the sdrplay_api_DbgLvl_t enumerated type

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.13 sdrplay_api_GetDeviceParams

```
sdrplay_api_ErrT sdrplay_api_GetDeviceParams(HANDLE dev,sdrplay_api_DeviceParamsT **deviceParams)
```

Description:

Devices are configured via the parameters contained in the device parameter structure. After selecting a device, the default device parameters are returned and can be modified as required before sdrplay_api_Init() is called. After sdrplay_api_Init() has been called, any changes made to the device parameters must be signalled to the API using sdrplay_api_Update() before they will be applied.

Parameters:

`dev`	Handle of selected device from current device enumeration structure
`deviceParams`	Pointer to a pointer to the device parameters used to setup/control the device

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_NotInitialised`	Device has not been selected
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.14 sdrplay_api_Init

```
sdrplay_api_ErrT sdrplay_api_Init(HANDLE dev,
                                  sdrplay_api_CallbackFnsT *callbackFns,
                                  void *cbContext)
```

Description:

This function will initialise the tuners according to the device parameter structure. After successfully completing initialisation it will set up a thread inside the API which will perform the processing chain. This thread will use the callback function to return the data to the calling application.
Processing chain (in order):

ReadUSBdata	fetch packets of IQ samples from USB interface
DCoffsetCorrection	enabled by default
Agc	enabled by default
DownConvert	enabled in LIF mode when parameters are consistent with down-conversion to baseband
Decimate	disabled by default
IQimbalanceCorrection	enabled by default

Conditions for LIF down-conversion to be enabled for all RSPs in single tuner mode:

```
(fsHz == 8192000) && (bwType == sdrplay_api_BW_1_536) && (ifType == sdrplay_api_IF_2_048)
(fsHz == 8000000) && (bwType == sdrplay_api_BW_1_536) && (ifType == sdrplay_api_IF_2_048)
(fsHz == 8000000) && (bwType == sdrplay_api_BW_5_000) && (ifType == sdrplay_api_IF_2_048)
(fsHz == 2000000) && (bwType <= sdrplay_api_BW_0_300) && (ifType == sdrplay_api_IF_0_450)
(fsHz == 2000000) && (bwType == sdrplay_api_BW_0_600) && (ifType == sdrplay_api_IF_0_450)
(fsHz == 6000000) && (bwType <= sdrplay_api_BW_1_536) && (ifType == sdrplay_api_IF_1_620)
```

In RSPduo master/slave mode, down-conversion is always enabled.

In RSPduo master/slave mode, the slave application cannot be initialised until the master application is running. In this case, a call to sdrplay_api_Init() will return sdrplay_api_StartPending without starting and the call must be repeated after a sdrplay_api_RspDuoModeChange->sdrplay_api_MasterInitialised event has been received.

Conditions for HDR mode to be enabled for the RSPdx with the hardware 500 kHz low pass filter:

```
(rfHz == 135000) || (rfHz == 175000) || (rfHz == 220000) || (rfHz == 250000) ||
(rfHz == 340000) || (rfHz == 475000) && hdrEnable
```

Conditions for HDR mode to be enabled for the RSPdx with the hardware 2 MHz low pass filter:

```
(rfHz == 516000) || (rfHz == 875000) || (rfHz == 1125000) || (rfHz == 1900000) && hdrEnable
```

Parameters:

`dev`	Handle of selected device from current device enumeration structure
`callbackFns`	Pointer to a structure specifying the callback functions to use to send processed data and events
`cbContext`	Pointer to a context passed to the API that will be returned as a parameter in the callback functions

Return:

`sdrplay_api_ErrT` Error code as defined below:

`sdrplay_api_Success`	Successful completion
`sdrplay_api_Fail`	Command failed
`sdrplay_api_NotInitialised`	Device has not been selected
`sdrplay_api_InvalidParam`	NULL pointer
`sdrplay_api_AlreadyInitialised`	There has been a previous call to this function
`sdrplay_api_OutOfRange`	One or more parameters are set incorrectly
`sdrplay_api_HwError`	HW error occured during tuner initialisation
`sdrplay_api_RfUpdateError`	Failed to update Rf frequency
`sdrplay_api_StartPending`	Master device not running
`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.15 sdrplay_api_Uninit

```
sdrplay_api_ErrT sdrplay_api_Uninit(HANDLE dev)
```

Description:

Stops the stream and uninitialises the tuners. In RSPduo master/slave mode, the master application cannot be uninitialised until the slave application is stopped. In this case, a call to sdrplay_api_Uninit() will return sdrplay_api_StopPending without making any changes and the call must be repeated after a sdrplay_api_RspDuoModeChange->sdrplay_api_SlaveUninitialised event has been received.

Parameters:

`dev`	Handle of selected device from current device enumeration structure

Return:

`sdrplay_api_ErrT`	Error code as defined below:	
	`sdrplay_api_Success`	Successful completion
	`sdrplay_api_Fail`	Command failed
	`sdrplay_api_NotInitialised`	Device has not been selected
	`sdrplay_api_StopPending`	Slave device running
	`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.16 sdrplay_api_Update

```
sdrplay_api_ErrT sdrplay_api_Update(HANDLE dev,
                                    sdrplay_api_TunerSelectT tuner,
                                    sdrplay_api_ReasonForUpdateT reasonForUpdate,
                                    sdrplay_api_ReasonForUpdateExtension1T reasonForUpdateExt1)
```

Description:

This function is used to indicate that parameters have been changed and need to be applied. Used to change any combination of values of the parameters. If required it will stop the stream, change the values and then start the stream again, otherwise it will make the changes directly.

The parameters associated with each update type are specified below:

Valid sdrplay_api_ReasonForUpdateT parameters:

```
sdrplay_api_Update_None                       : No changes relating to ReasonForUpdateT
sdrplay_api_Update_Dev_Fs                     : deviceParams->devParams->fsFreq->*
sdrplay_api_Update_Dev_Ppm                    : deviceParams->devParams->ppm
sdrplay_api_Update_Dev_SyncUpdate             : deviceParams->devParams->syncUpdate->*
sdrplay_api_Update_Dev_ResetFlags             : deviceParams->devParams->resetFlags->*
sdrplay_api_Update_Rsp1a_BiasTControl         :
                              deviceParams->rxChannel*->rsp1aTunerParams->biasTEnable
sdrplay_api_Update_Rsp1a_RfNotchControl       :
                              deviceParams->devParams->rsp1aParams->rfNotchEnable
sdrplay_api_Update_Rsp1a_RfDabNotchControl    :
                              deviceParams->devParams->rsp1aParams->rfDabNotchEnable
sdrplay_api_Update_Rsp2_BiasTControl          :
                              deviceParams->rxChannel*->rsp2TunerParams->biasTEnable
sdrplay_api_Update_Rsp2_AmPortSelect          :
                              deviceParams->rxChannel*->rsp2TunerParams->amPortSel
sdrplay_api_Update_Rsp2_AntennaControl        :
                              deviceParams->rxChannel*->rsp2TunerParams->antennaSel
sdrplay_api_Update_Rsp2_RfNotchControl        :
                              deviceParams->rxChannel*->rsp2TunerParams->rfNotchEnable
sdrplay_api_Update_Rsp2_ExtRefControl         :
                              deviceParams->devParams->rsp2Params->extRefOutputEn
sdrplay_api_Update_RspDuo_ExtRefControl       :
                              deviceParams->devParams->rspDuoParams->extRefOutputEn
sdrplay_api_Update_Tuner_Gr                   :
                              deviceParams->rxChannel*->tunerParams->gain->gRdB or
                              deviceParams->rxChannel*->tunerParams->gain->LNAstate
sdrplay_api_Update_Tuner_GrLimits             :
                              deviceParams->rxChannel*->tunerParams->gain->minGr
sdrplay_api_Update_Tuner_Frf                  : deviceParams->rxChannel*->tunerParams->rfFreq->*
sdrplay_api_Update_Tuner_BwType               : deviceParams->rxChannel*->tunerParams->bwType
sdrplay_api_Update_Tuner_IfType               : deviceParams->rxChannel*->tunerParams->ifType
sdrplay_api_Update_Tuner_DcOffset             : deviceParams->rxChannel*->tunerParams->loMode
sdrplay_api_Update_Tuner_LoMode               :
                              deviceParams->rxChannel*->tunerParams->dcOffsetTuner->*
sdrplay_api_Update_Ctrl_DCoffsetIQimbalance   : deviceParams->rxChannel*->ctrlParams->dcOffset->*
sdrplay_api_Update_Ctrl_Decimation            :
                              deviceParams->rxChannel*->ctrlParams->decimation->*
sdrplay_api_Update_Ctrl_Agc                   : deviceParams->rxChannel*->ctrlParams->agc->*
sdrplay_api_Update_Ctrl_AdsbMode              : deviceParams->rxChannel*->ctrlParams->adsbMode
sdrplay_api_Update_Ctrl_OverloadMsgAck        : none (used whenever a power overload event occurs
                                                as an acknowledge signal)
sdrplay_api_Update_RspDuo_BiasTControl        :
                              deviceParams->rxChannel*->rspDuoTunerParams->biasTEnable
sdrplay_api_Update_RspDuo_AmPortSelect        :
                              deviceParams->rxChannel*->rspDuoTunerParams->tuner1AmPortSel
sdrplay_api_Update_RspDuo_Tuner1AmNotchControl:
                              deviceParams->rxChannel*->rspDuoTunerParams->tuner1AmNotchEnable
sdrplay_api_Update_RspDuo_RfNotchControl      :
                              deviceParams->rxChannel*->rspDuoTunerParams->rfNotchEnable
sdrplay_api_Update_RspDuo_RfDabNotchControl   :
                              deviceParams->rxChannel*->rspDuoTunerParams->rfDabNotchEnable
```

Valid sdrplay_api_ReasonForUpdateExtension1T parameters:

```
sdrplay_api_Update_Ext1_None               : No changes relating to ReasonForUpdateExtension1T
sdrplay_api_Update_RspDx_HdrEnable         : deviceParams->devParams->rspDxParams->hdrEnable
sdrplay_api_Update_RspDx_BiasTControl      : deviceParams->devParams->rspDxParams->biasTEnable
sdrplay_api_Update_RspDx_AntennaControl    : deviceParams->devParams->rspDxParams->antennaSel
sdrplay_api_Update_RspDx_RfNotchControl    : deviceParams->devParams->rspDxParams->rfNotchEnable
sdrplay_api_Update_RspDx_RfDabNotchControl :
                           deviceParams->devParams->rspDxParams->rfDabNotchEnable
sdrplay_api_Update_RspDx_HdrBw             : deviceParams->devParams->rspDxTunerParams-> hdrBw
```

Parameters:

`dev`	Handle of selected device from current device enumeration structure
`tuner`	Specifies which tuner(s) to apply the update to
`reasonForUpdate`	Specifies the reason for the call depending on which parameters have been changed in the sdrplay_api_ReasonForUpdateT structure
`reasonForUpdateExt1`	Specifies the reason for the call depending on which parameters have been changed in the sdrplay_api_ ReasonForUpdateExtension1T structure

Return:

`sdrplay_api_ErrT` Error code as defined below:

`sdrplay_api_Success`	Successful completion
`sdrplay_api_Fail`	Command failed
`sdrplay_api_InvalidParam`	NULL pointer or invalid operating mode
`sdrplay_api_OutOfRange`	One or more parameters are set incorrectly
`sdrplay_api_HwError`	HW error occured during tuner initialisation
`sdrplay_api_FsUpdateError`	Failed to update sample rate
`sdrplay_api_RfUpdateError`	Failed to update Rf frequency
`sdrplay_api_GainUpdateError`	Failed to update gain
`sdrplay_api_NotEnabled`	Feature not enabled
`sdrplay_api_ServiceNotResponding`	Communication channel with service broken

3.17 sdrplay_api_SwapRspDuoActiveTuner

```
sdrplay_api_ErrT sdrplay_api_SwapRspDuoActiveTuner(HANDLE dev,
                                                   sdrplay_api_TunerSelectT *currentTuner,
                                                   sdrplay_api_RspDuo_AmPortSelectT tuner1AmPortSel)
```

Description:

After a call to sdrplay_api_Init() for an RSPduo in single tuner mode, this function can be called to change between tuners while maintaining the exact same settings (except in the case when switching from TunerB to TunerA when HiZ is selected by the tuner1AmPortSel parameter). After successful completion, the current device enumeration structure will be updated with the newly selected tuner.

Parameters:

dev	Handle of selected device from current device enumeration structure
currentTimer	Pointer to the selected tuner stored in the current device enumeration structure
tuner1AmPortSel	Specifies whether to use the HiZ port when switching to TunerA when the AM band is selected

Return:

sdrplay_api_ErrT — Error code as defined below:

sdrplay_api_Success	Successful completion
sdrplay_api_Fail	Command failed
sdrplay_api_InvalidParam	NULL pointer or invalid operating mode
sdrplay_api_OutOfRange	One or more parameters are set incorrectly
sdrplay_api_HwError	HW error occured during tuner initialisation
sdrplay_api_RfUpdateError	Failed to update Rf frequency
sdrplay_api_ServiceNotResponding	Communication channel with service broken

3.18 sdrplay_api_SwapRspDuoDualTunerModeSampleRate

```
sdrplay_api_ErrT sdrplay_api_SwapRspDuoDualTunerModeSampleRate(HANDLE dev,
                                                               double *currentSampleRate)
```

Description:

After a call to sdrplay_api_Init() for an RSPduo in master/slave mode, this function can be called to change sample rates between 6MHz and 8MHz. After successful completion, the current device enumeration structure will be updated with the newly selected sample rate. This function can only be called by the master application. As this affects the slave application as well, if it is currently active, the call will return sdrplay_api_StopPending without making any changes and the call must be repeated after a sdrplay_api_RspDuoModeChange->sdrplay_api_SlaveUninitialised event has been received.

Parameters:

dev	Handle of selected device from current device enumeration structure
currentSampleRate	Pointer to the selected sample rate stored in the current device enumeration structure

Return:

sdrplay_api_ErrT — Error code as defined below:

sdrplay_api_Success	Successful completion
sdrplay_api_Fail	Command failed
sdrplay_api_InvalidParam	NULL pointer or invalid operating mode
sdrplay_api_OutOfRange	One or more parameters are set incorrectly
sdrplay_api_HwError	HW error occured during tuner initialisation
sdrplay_api_RfUpdateError	Failed to update Rf frequency
sdrplay_api_StopPending	Slave device running
sdrplay_api_ServiceNotResponding	Communication channel with service broken

3.19 Streaming Data Callback

```
typedef void (*sdrplay_api_StreamCallback_t)(short *xi,
                                             short *xq,
                                             sdrplay_api_StreamCbParamsT *params,
                                             unsigned int numSamples,
                                             unsigned int reset,
                                             void *cbContext)
```

Description:

This callback is triggered when there are samples to be processed.

Parameters:

xi	Pointer to the real data in the buffer
xq	Pointer to the imaginary data in the buffer
params	Pointer to the stream callback parameters structure
numSamples	The number of samples in the current buffer
reset	Indicates if a re-initialisation has occurred within the API and that local buffering should be reset
cbContext	Pointer to context passed into sdrplay_api_Init()

Return:

none

3.20 Event Callback

```
typedef void (*sdrplay_api_EventCallback_t)(sdrplay_api_EventT eventId,
                                            sdrplay_api_TunerSelectT tuner,
                                            sdrplay_api_EventParamsT *params,
                                            void *cbContext)
```

Description:

This callback is triggered whenever an event occurs. The list of events is specified by the sdrplay_api_EventT enumerated type.

Parameters:

eventId	Indicates the type of event that has occurred
tuner	Indicates which tuner(s) the event relates to
params	Pointer to the event callback union (the structure used depends on the eventId)
cbContext	Pointer to context passed into sdrplay_api_Init()

Return:

none

4 API Usage

```
// sdrplay_api_sample_app.c : Simple console application showing the use of the API

#include <Windows.h>
#include <stdio.h>
#include <conio.h>

#include "sdrplay_api.h"

int masterInitialised = 0;
int slaveUninitialised = 0;

sdrplay_api_DeviceT *chosenDevice = NULL;

void StreamACallback(short *xi, short *xq, sdrplay_api_StreamCbParamsT *params, unsigned int
numSamples, unsigned int reset, void *cbContext)
{
    if (reset)
        printf("sdrplay_api_StreamACallback: numSamples=%d\n", numSamples);
    // Process stream callback data here
    return;
}

void StreamBCallback(short *xi, short *xq, sdrplay_api_StreamCbParamsT *params, unsigned int
numSamples, unsigned int reset, void *cbContext)
{
    if (reset)
        printf("sdrplay_api_StreamBCallback: numSamples=%d\n", numSamples);
    // Process stream callback data here - this callback will only be used in dual tuner mode
    return;
}

void EventCallback(sdrplay_api_EventT eventId, sdrplay_api_TunerSelectT tuner,
sdrplay_api_EventParamsT *params, void *cbContext)
{
    switch(eventId)
    {
    case sdrplay_api_GainChange:
        printf("sdrplay_api_EventCb: %s, tuner=%s gRdB=%d lnaGRdB=%d systemGain=%.2f\n",
               "sdrplay_api_GainChange", (tuner == sdrplay_api_Tuner_A)? "sdrplay_api_Tuner_A":
               "sdrplay_api_Tuner_B", params->gainParams.gRdB, params->gainParams.lnaGRdB,
               params->gainParams.currGain);
        break;

    case sdrplay_api_PowerOverloadChange:
        printf("sdrplay_api_PowerOverloadChange: tuner=%s powerOverloadChangeType=%s\n",
               (tuner == sdrplay_api_Tuner_A)? "sdrplay_api_Tuner_A": "sdrplay_api_Tuner_B",
               (params->powerOverloadParams.powerOverloadChangeType ==
               sdrplay_api_Overload_Detected)? "sdrplay_api_Overload_Detected":
               "sdrplay_api_Overload_Corrected");
        // Send update message to acknowledge power overload message received
        sdrplay_api_Update(chosenDevice->dev, tuner, sdrplay_api_Update_Ctrl_OverloadMsgAck,
               sdrplay_api_Update_Ext1_None);
        break;

    case sdrplay_api_RspDuoModeChange:
        printf("sdrplay_api_EventCb: %s, tuner=%s modeChangeType=%s\n",
               "sdrplay_api_RspDuoModeChange", (tuner == sdrplay_api_Tuner_A)?
               "sdrplay_api_Tuner_A": "sdrplay_api_Tuner_B",
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_MasterInitialised)?
               "sdrplay_api_MasterInitialised":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveAttached)?
               "sdrplay_api_SlaveAttached":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveDetached)?
               "sdrplay_api_SlaveDetached":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveInitialised)?
               "sdrplay_api_SlaveInitialised":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveUninitialised)?
               "sdrplay_api_SlaveUninitialised":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_MasterDllDisappeared)?
               "sdrplay_api_MasterDllDisappeared":
               (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveDllDisappeared)?
               "sdrplay_api_SlaveDllDisappeared": "unknown type");
```

```
        if (params->rspDuoModeParams.modeChangeType == sdrplay_api_MasterInitialised)
            masterInitialised = 1;
        if (params->rspDuoModeParams.modeChangeType == sdrplay_api_SlaveUninitialised)
            slaveUninitialised = 1;
        break;

    case sdrplay_api_DeviceRemoved:
        printf("sdrplay_api_EventCb: %s\n", "sdrplay_api_DeviceRemoved");
        break;

    default:
        printf("sdrplay_api_EventCb: %d, unknown event\n", eventId);
        break;
    }
}

void usage(void)
{
    printf("Usage: sample_app.exe [A|B] [ms]\n");
    exit(1);
}

int main(int argc, char *argv[])
{
    sdrplay_api_DeviceT devs[6];
    unsigned int ndev;
    int i;
    float ver = 0.0;
    sdrplay_api_ErrT err;
    sdrplay_api_DeviceParamsT *deviceParams = NULL;
    sdrplay_api_CallbackFnsT cbFns;
    sdrplay_api_RxChannelParamsT *chParams;
    int reqTuner = 0;
    int master_slave = 0;
    char c;
    unsigned int chosenIdx = 0;

    if ((argc > 1) && (argc < 4))
    {
        if (!strcmp(argv[1], "A"))
        {
            reqTuner = 0;
        }
        else if (!strcmp(argv[1], "B"))
        {
            reqTuner = 1;
        }
        else
            usage();
        if (argc == 3)
        {
            if (!strcmp(argv[2], "ms"))
            {
                master_slave = 1;
            }
            else
                usage();
        }
    }
    else if (argc >= 4)
    {
        usage();
    }

    printf("requested Tuner%c Mode=%s\n", (reqTuner == 0)? 'A': 'B', (master_slave == 0)?
                "Single_Tuner": "Master/Slave");

    // Open API
    if ((err = sdrplay_api_Open()) != sdrplay_api_Success)
    {
        printf("sdrplay_api_Open failed %s\n", sdrplay_api_GetErrorString(err));
    }
    else
```

```
{
    // Enable debug logging output
    if ((err = sdrplay_api_DebugEnable(NULL, 1)) != sdrplay_api_Success)
    {
        printf("sdrplay_api_DebugEnable failed %s\n", sdrplay_api_GetErrorString(err));
    }

    // Check API versions match
    if ((err = sdrplay_api_ApiVersion(&ver)) != sdrplay_api_Success)
    {
        printf("sdrplay_api_ApiVersion failed %s\n", sdrplay_api_GetErrorString(err));
    }
    if (ver != SDRPLAY_API_VERSION)
    {
        printf("API version don't match (local=%.2f dll=%.2f)\n", SDRPLAY_API_VERSION, ver);
        goto CloseApi;
    }

    // Lock API while device selection is performed
    sdrplay_api_LockDeviceApi();

    // Fetch list of available devices
    if ((err = sdrplay_api_GetDevices(devs, &ndev, sizeof(devs) /
           sizeof(sdrplay_api_DeviceT))) != sdrplay_api_Success)
    {
        printf("sdrplay_api_GetDevices failed %s\n", sdrplay_api_GetErrorString(err));
        goto UnlockDeviceAndCloseApi;
    }
    printf("MaxDevs=%d NumDevs=%d\n", sizeof(devs) / sizeof(sdrplay_api_DeviceT), ndev);
    if (ndev > 0)
    {
        for (i = 0; i < (int)ndev; i++)
        {
            if (devs[i].hwVer == SDRPLAY_RSPduo_ID)
                printf("Dev%d: SerNo=%s hwVer=%d tuner=0x%.2x rspDuoMode=0x%.2x\n", i,
                   devs[i].SerNo, devs[i].hwVer , devs[i].tuner, devs[i].rspDuoMode);
            else
                printf("Dev%d: SerNo=%s hwVer=%d tuner=0x%.2x\n", i, devs[i].SerNo,
                   devs[i].hwVer, devs[i].tuner);
        }

        // Choose device
        if ((reqTuner == 1) || (master_slave == 1))  // requires RSPduo
        {
            // Pick first RSPduo
            for (i = 0; i < (int)ndev; i++)
            {
                if (devs[i].hwVer == SDRPLAY_RSPduo_ID)
                {
                    chosenIdx = i;
                    break;
                }
            }
        }
        else
        {
            // Pick first device of any type
            for (i = 0; i < (int)ndev; i++)
            {
                chosenIdx = i;
                break;
            }
        }
        if (i == ndev)
        {
            printf("Couldn't find a suitable device to open - exiting\n");
            goto UnlockDeviceAndCloseApi;
        }
        printf("chosenDevice = %d\n", chosenIdx);
        chosenDevice = &devs[chosenIdx];

        // If chosen device is an RSPduo, assign additional fields
        if (chosenDevice->hwVer == SDRPLAY_RSPduo_ID)
        {
```

```
    // If master device is available, select device as master
    if (chosenDevice->rspDuoMode & sdrplay_api_RspDuoMode_Master)
    {
        // Select tuner based on user input (or default to TunerA)
        chosenDevice->tuner = sdrplay_api_Tuner_A;
        if (reqTuner == 1)
            chosenDevice->tuner = sdrplay_api_Tuner_B;

        // Set operating mode
        if (!master_slave)  // Single tuner mode
        {
            chosenDevice->rspDuoMode = sdrplay_api_RspDuoMode_Single_Tuner;
            printf("Dev%d: selected rspDuoMode=0x%.2x tuner=0x%.2x\n", chosenIdx,
                  chosenDevice->rspDuoMode, chosenDevice->tuner);
        }
        else
        {
            chosenDevice->rspDuoMode = sdrplay_api_RspDuoMode_Master;
            // Need to specify sample frequency in master/slave mode
            chosenDevice->rspDuoSampleFreq = 6000000.0;
     printf("Dev%d: selected rspDuoMode=0x%.2x tuner=0x%.2x rspDuoSampleFreq=%.1f\n",
                        chosenIdx, chosenDevice->rspDuoMode,
                        chosenDevice->tuner, chosenDevice->rspDuoSampleFreq);
        }
    }
    else  // Only slave device available
    {
        // Shouldn't change any parameters for slave device
    }
}

// Select chosen device
if ((err = sdrplay_api_SelectDevice(chosenDevice)) != sdrplay_api_Success)
{
    printf("sdrplay_api_SelectDevice failed %s\n", sdrplay_api_GetErrorString(err));
    goto UnlockDeviceAndCloseApi;
}

// Unlock API now that device is selected
sdrplay_api_UnlockDeviceApi();

// Retrieve device parameters so they can be changed if wanted
if ((err = sdrplay_api_GetDeviceParams(chosenDevice->dev, &deviceParams)) !=
   sdrplay_api_Success)
{
    printf("sdrplay_api_GetDeviceParams failed %s\n",
          sdrplay_api_GetErrorString(err));
    goto CloseApi;
}

// Check for NULL pointers before changing settings
if (deviceParams == NULL)
{
    printf("sdrplay_api_GetDeviceParams returned NULL deviceParams pointer\n");
    goto CloseApi;
}

// Configure dev parameters
if (deviceParams->devParams != NULL)
{
   // This will be NULL for slave devices, only the master can change these parameters
    // Only need to update non-default settings
    if (master_slave == 0)
    {
        // Change from default Fs  to 8MHz
        deviceParams->devParams->fsFreq.fsHz = 8000000.0;
    }
    else
    {
        // Can't change Fs in master/slave mode
    }
}

// Configure tuner parameters (depends on selected Tuner which parameters to use)
```

```
            chParams = (chosenDevice->tuner == sdrplay_api_Tuner_B)? deviceParams->rxChannelB:
deviceParams->rxChannelA;
            if (chParams != NULL)
            {
                chParams->tunerParams.rfFreq.rfHz = 220000000.0;
                chParams->tunerParams.bwType = sdrplay_api_BW_1_536;
                if (master_slave == 0) // Change single tuner mode to ZIF
                {
                    chParams->tunerParams.ifType = sdrplay_api_IF_Zero;
                }
                chParams->tunerParams.gain.gRdB = 40;
                chParams->tunerParams.gain.LNAstate = 5;

                // Disable AGC
                chParams->ctrlParams.agc.enable = sdrplay_api_AGC_DISABLE;
            }
            else
            {
                printf("sdrplay_api_GetDeviceParams returned NULL chParams pointer\n");
                goto CloseApi;
            }

            // Assign callback functions to be passed to sdrplay_api_Init()
            cbFns.StreamACbFn = StreamACallback;
            cbFns.StreamBCbFn = StreamBCallback;
            cbFns.EventCbFn = EventCallback;

            // Now we're ready to start by calling the initialisation function
            // This will configure the device and start streaming
            if ((err = sdrplay_api_Init(chosenDevice->dev, &cbFns, NULL)) != sdrplay_api_Success)
            {
                printf("sdrplay_api_Init failed %s\n", sdrplay_api_GetErrorString(err));
                if (err == sdrplay_api_StartPending) // This can happen if we're starting in
master/slave mode as a slave and the master is not yet running
                {
                    while(1)
                    {
                        Sleep(1000);
                        if (masterInitialised) // Keep polling flag set in event callback until
the master is initialised
                        {
                            // Redo call - should succeed this time
                            if ((err = sdrplay_api_Init(chosenDevice->dev, &cbFns, NULL)) !=
sdrplay_api_Success)
                            {
                                printf("sdrplay_api_Init failed %s\n",
sdrplay_api_GetErrorString(err));
                            }
                            goto CloseApi;
                        }
                        printf("Waiting for master to initialise\n");
                    }
                }
                else
                {
                    sdrplay_api_ErrorInfoT *errInfo = sdrplay_api_GetLastError(NULL);
                    if (errInfo != NULL)
                        printf("Error in %s: %s(): line %d: %s\n", errInfo->file, errInfo-
>function, errInfo->line, errInfo->message);
                    goto CloseApi;
                }
            }

            while (1) // Small loop allowing user to control gain reduction in +/-1dB steps using
keyboard keys
            {
                if (_kbhit())
                {
                    c = _getch();
                    if (c == 'q')
                        break;
                    else if (c == 'u')
                    {
                        chParams->tunerParams.gain.gRdB += 1;
```

```
                        // Limit it to a maximum of 59dB
                        if (chParams->tunerParams.gain.gRdB > 59)
                            chParams->tunerParams.gain.gRdB = 20;
                        if ((err = sdrplay_api_Update(chosenDevice->dev, chosenDevice->tuner,
                              sdrplay_api_Update_Tuner_Gr, sdrplay_api_Update_Ext1_None)) !=
                      sdrplay_api_Success)
                        {
                            printf("sdrplay_api_Update sdrplay_api_Update_Tuner_Gr failed %s\n",
                              sdrplay_api_GetErrorString(err));
                            break;
                        }
                    }
                    else if (c == 'd')
                    {
                        chParams->tunerParams.gain.gRdB -= 1;
                        // Limit it to a minimum of 20dB
                        if (chParams->tunerParams.gain.gRdB < 20)
                            chParams->tunerParams.gain.gRdB = 59;
                        if ((err = sdrplay_api_Update(chosenDevice->dev, chosenDevice->tuner,
                              sdrplay_api_Update_Tuner_Gr, sdrplay_api_Update_Ext1_None)) !=
                      sdrplay_api_Success)
                        {
                            printf("sdrplay_api_Update sdrplay_api_Update_Tuner_Gr failed %s\n",
                              sdrplay_api_GetErrorString(err));
                            break;
                        }
                    }
                }
                Sleep(100);
            }

            // Finished with device so uninitialise it
            if ((err = sdrplay_api_Uninit(chosenDevice->dev)) != sdrplay_api_Success)
            {
                printf("sdrplay_api_Uninit failed %s\n", sdrplay_api_GetErrorString(err));
                if (err == sdrplay_api_StopPending)
                {
                // We're stopping in master/slave mode as a master and the slave is still running
                    while(1)
                    {
                        Sleep(1000);
                        if (slaveUninitialised)
                        {
                      // Keep polling flag set in event callback until the slave is uninitialised
                            // Repeat call - should succeed this time
                            if ((err = sdrplay_api_Uninit(chosenDevice->dev)) !=
                              sdrplay_api_Success)
                            {
                                printf("sdrplay_api_Uninit failed %s\n",
                                      sdrplay_api_GetErrorString(err));
                            }
                            slaveUninitialised = 0;
                            goto CloseApi;
                        }
                        printf("Waiting for slave to uninitialise\n");
                    }
                }
                goto CloseApi;
            }

            // Release device (make it available to other applications)
            sdrplay_api_ReleaseDevice(chosenDevice);
        }
UnlockDeviceAndCloseApi:
        // Unlock API
        sdrplay_api_UnlockDeviceApi();

CloseApi:
        // Close API
        sdrplay_api_Close();
    }
    return 0;
}
```

5 Gain Reduction Tables

LNA GR (dB) by Frequency Range and LNAstate for RSP1:

Frequency (MHz)	LNAstate			
	0	1	2	3
0-420	0	24	19[1]	43[2]
420-1000	0	7	19[1]	26[2]
1000-2000	0	5	19[1]	24[2]

LNA GR (dB) by Frequency Range and LNAstate for RSP1A:

Frequency (MHz)	LNAstate									
	0	1	2	3	4	5	6	7	8	9
0-60	0	6	12	18	37	42	61[2]			
60-420	0	6	12	18	20	26	32	38	57	62
420-1000	0	7	13	19	20	27	33	39	45	64[2]
1000-2000	0	6	12	20	26	32	38	43	62[2]	

LNA GR (dB) by Frequency Range and LNAstate for RSP2:

Frequency (MHz)	LNAstate								
	0	1	2	3	4	5	6	7	8
0-420 (Port A or B)	0	10	15	21	24	34	39	45	64[2]
420-1000	0	7	10	17	22	41[2]			
1000-2000	0	5	21	15[3]	15[3]	34[2]			
0-60 (HiZ Port)	0	6	12	18	37[2]				

LNA GR (dB) by Frequency Range and LNAstate for RSPduo:

Frequency (MHz)	LNAstate									
	0	1	2	3	4	5	6	7	8	9
0-60 (50 Ω Ports)	0	6	12	18	37	42	61[2]			
60-420	0	6	12	18	20	26	32	38	57	62
420-1000	0	7	13	19	20	27	33	39	45	64[2]
1000-2000	0	6	12	20	26	32	38	43	62[2]	
0-60 (HiZ Port)	0	6	12	18	37[2]					

LNA GR (dB) by Frequency Range and LNAstate for RSPdx:

Freq (MHz)	LNAstate													
	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0-2 (HDR mode)	0	3	6	9	12	15	18	21	24	25	27	30	33	36
0-12	0	3	6	9	12	15	24	27	30	33	36	39	42	45
12-60	0	3	6	9	12	15	18	24	27	30	33	36	39	42
60-250	0	3	6	9	12	15	24	27	30	33	36	39	42	45
250-420	0	3	6	9	12	15	18	24	27	30	33	36	39	42
420-1000	0	7	10	13	16	19	22	25	31	34	37	40	43	46
1000-2000	0	5	8	11	14	17	20	32	35	38	41	44	47	50

Freq (MHz)	LNAstate													
	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0-2 (HDR mode)	39	42	45	48	51	54	57	60						
0-12	48	51	54	57	60									
12-60	45	48	51	54	57	60								
60-250	48	51	54	57	60	63	66	69	72	75	78	81	84	
250-420	45	48	51	54	57	60	63	66	69	72	75	78	81	84
420-1000	49	52	55	58	61	64	67							
1000-2000	53	56	59	62	65									

[1] Mixer GR only
[2] Includes LNA GR plus mixer GR
[3] In LNAstate 3, external LNA GR only, in LNAstate 4, external plus internal LNA GR

6 Legal Information

For more information, contact: https://www.sdrplay.com/support

Legal Information