SDRplay Limited.

Software Defined Radio API

Applications

Revision History

Revision	Release Date:	Reason for Change:	Originator
1.0	26 May 2013	Pre-Release 0.0.1	APC
1.1	08 August 2013	Add reset function	APC
1.2	31 July 2014	Extended frequency range	APC
1.3	27 March 2015	Updated error codes & tidy up	APC
1.7	13 August 2015	Added new commands & update mir_sdr_SetDcMode to add missing modes	IMH
1.8	22 December 2015	DC/IQ compensation & new gain map	APC
1.8.1	15 January 2016	Tidy up text	APC
1.94	11th July 2016	Range of new functions	APC
1.97	18th September 2016	Added new AGC + DC offset schemes	APC
2.04	22nd November 2016	Update for RSP2	KA
2.05	30th November 2016	Modified return from mir_sdr_GetDevices	KA
2.06	1th December 2016	Added mir_sdr_ReleaseDeviceIdx, modified gain callback parameters	KA
2.07	2nd December 2016	Modified SetGr functions	KA
2.08	15th December 2016	Fixed issue in IQ compensation	KA

2.09	22nd December 2016	Modification to AGC loop (no external API mods)	KA
2.11	15th November 2017	Added support for RSP1A	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 and RSP1A from SDRplay Limited which make use of the Mirics USB bridge device (MSi2500) and the multi-standard tuner (MSi001).

2 API Data Types

The header file `mir_sdr.h` provides the definitions of the external data types provided by this API.

2.1 Error Code Enumerated Type

```
typedef enum
{
  mir_sdr_Success             = 0,
  mir_sdr_Fail                = 1,
  mir_sdr_InvalidParam        = 2,
  mir_sdr_OutOfRange          = 3,
  mir_sdr_GainUpdateError     = 4,
  mir_sdr_RfUpdateError       = 5,
  mir_sdr_FsUpdateError       = 6,
  mir_sdr_HwError             = 7,
  mir_sdr_AliasingError       = 8,
  mir_sdr_AlreadyInitialised  = 9,
  mir_sdr_NotInitialised      = 10,
  mir_sdr_NotEnabled          = 11,
  mir_sdr_HwVerError          = 12,
  mir_sdr_OutOfMemError       = 13
} mir_sdr_ErrT;
```

2.2 Band Width Enumerated Type

```
typedef enum
{
  mir_sdr_BW_Undefined = 0,
  mir_sdr_BW_0_200     = 200,
  mir_sdr_BW_0_300     = 300,
  mir_sdr_BW_0_600     = 600,
  mir_sdr_BW_1_536     = 1536,
  mir_sdr_BW_5_000     = 5000,
  mir_sdr_BW_6_000     = 6000,
  mir_sdr_BW_7_000     = 7000,
  mir_sdr_BW_8_000     = 8000
} mir_sdr_Bw_MHzT;
```

2.3 IF Enumerated Type

```
typedef enum
{
  Mir_sdr_IF_Undefinded       = -1,
  mir_sdr_IF_Zero             = 0,
  mir_sdr_IF_0_450            = 450,
  mir_sdr_IF_1_620            = 1620,
  mir_sdr_IF_2_048            = 2048
} mir_sdr_If_kHzT;
```

2.4 Transfer Mode Enumerated Type

```
typedef enum
{
  mir_sdr_ISOCH = 0,
  mir_sdr_BULK  = 1
} mir_sdr_TransferModeT;
```

2.5 Reinit Enumerated Type

Used to specify the reason for a reinitialise request - values should be or'd together if there are multiple reasons for the request.

```
typedef enum
{
  mir_sdr_CHANGE_NONE         = 0x00,
  mir_sdr_CHANGE_GR           = 0x01,
  mir_sdr_CHANGE_FS_FREQ      = 0x02,
  mir_sdr_CHANGE_RF_FREQ      = 0x04,
  mir_sdr_CHANGE_BW_TYPE      = 0x08,
  mir_sdr_CHANGE_IF_TYPE      = 0x10,
  mir_sdr_CHANGE_LO_MODE      = 0x20,
  mir_sdr_CHANGE_AM_PORT      = 0x40
} mir_sdr_ReasonForReinitT;
```

2.6 LO Mode Enumerated Type

Used to specify the up-converter (1st) LO frequency for AM for use between frequencies below 60MHz and between 250MHz and 420MHz.

```
typedef enum
{
  mir_sdr_LO_Undefined = 0,
  mir_sdr_LO_Auto      = 1,   // 1st LO is automatically selected to provide appropriate coverage
                              // across all tuner frequency ranges
  mir_sdr_LO_120MHz    = 2,   // 1st LO is set to 120MHz (coverage gap between 370MHZ and 420MHz)
  mir_sdr_LO_144MHz    = 3,   // 1st LO is set to 144MHz (coverage gap between 250MHZ and 255MHz
                              //                                  and between 400MHz and 420MHz)
  mir_sdr_LO_168MHz    = 4    // 1st LO is set to 168MHz (coverage gap between 250MHZ and 265MHz)
} mir_sdr_LoModeT;
```

2.7 Band Enumerated Type

Used to specify the requested band.

```
typedef enum
{
  mir_sdr_BAND_AM_LO          = 0,          //     0 <= Freq <   12 MHz
  mir_sdr_BAND_AM_MID         = 1,          //    12 <= Freq <   30 MHz
  mir_sdr_BAND_AM_HI          = 2,          //    30 <= Freq <   60 MHz
  mir_sdr_BAND_VHF            = 3,          //    60 <= Freq <  120 MHz
  mir_sdr_BAND_3              = 4,          //   120 <= Freq <  250 MHz
  mir_sdr_BAND_X              = 5,          //   250 <= Freq <  420 MHz
  mir_sdr_BAND_4_5            = 6,          //   420 <= Freq < 1000 MHz
  mir_sdr_BAND_L              = 7           //  1000 <= Freq < 2000 MHz
} mir_sdr_BandT;
```

2.8 Gain Reduction Update Mode Enumerated Type

Use to specify which gain reduction update mode to use

```
typedef enum
{
  mir_sdr_USE_SET_GR                = 0,
  mir_sdr_USE_SET_GR_ALT_MODE       = 1,
  mir_sdr_USE_RSP_SET_GR            = 2,
} mir_sdr_SetGrModeT;
```

2.9 RSP2 Band Enumerated Type

```
typedef enum
{
  mir_sdr_RSPII_BAND_UNKNOWN = 0,
  mir_sdr_RSPII_BAND_AM_LO   = 1,
  mir_sdr_RSPII_BAND_AM_MID  = 2,
  mir_sdr_RSPII_BAND_AM_HI   = 3,
  mir_sdr_RSPII_BAND_VHF     = 4,
  mir_sdr_RSPII_BAND_3       = 5,
  mir_sdr_RSPII_BAND_X_LO    = 6,
  mir_sdr_RSPII_BAND_X_MID   = 7,
  mir_sdr_RSPII_BAND_X_HI    = 8,
  mir_sdr_RSPII_BAND_4_5     = 9,
  mir_sdr_RSPII_BAND_L       = 10
} mir_sdr_RSPII_BandT;
```

2.10 RSP2 Antenna Select Enumerated Type

```
typedef enum
{
  mir_sdr_RSPII_ANTENNA_A = 5,
  mir_sdr_RSPII_ANTENNA_B = 6
} mir_sdr_RSPII_AntennaSelectT;
```

2.11 AGC Control Enumerated Type

Used to enable/disable AGC and specify loop filter bandwidth.

```
typedef enum
{
  mir_sdr_AGC_DISABLE  = 0,
  mir_sdr_AGC_100HZ    = 1,
  mir_sdr_AGC_50HZ     = 2,
  mir_sdr_AGC_5HZ      = 3
} mir_sdr_AgcControlT;
```

2.12 Gain Message ID Enumerated Type

Used to identify messages passed back in the gRdB field of the gain callback function.

```
typedef enum
{
  mir_sdr_GAIN_MESSAGE_START_ID  = 0x80000000,
  mir_sdr_ADC_OVERLOAD_DETECTED  = mir_sdr_GAIN_MESSAGE_START_ID + 1,
  mir_sdr_ADC_OVERLOAD_CORRECTED = mir_sdr_GAIN_MESSAGE_START_ID + 2
} mir_sdr_GainMessageIdT;
```

2.13 Minimum IF Gain Reduction Enumerated Type

Used to control the allowable range of IF Gain Reduction values.

```
typedef enum
{
  mir_sdr_EXTENDED_MIN_GR = 0,       //  0 to 59
  mir_sdr_NORMAL_MIN_GR   = 20       // 20 to 59
} mir_sdr_MinGainReductionT;
```

2.14 Java Event Enumerated Type (Android only)

Used to pass events back from the API to the Java application on Android.

```
typedef enum
{
  mir_sdr_GetFd        = 0,
  mir_sdr_FreeFd       = 1,
  mir_sdr_DevNotFound  = 2,
  mir_sdr_DevRemoved   = 3,
  mir_sdr_GetVendorId  = 4,
  mir_sdr_GetProductId = 5,
  mir_sdr_GetRevId     = 6,
  mir_sdr_GetDeviceId  = 7
} mir_sdr_JavaReqT;
```

2.15 Device Enumeration Structure

Used to pass back RSP device information in mir_sdr_GetDevices(). All RSPx devices will be listed, but if a device is already in use, the devAvail flag will be set to 0 (instead of 1 if it is available).

```
typedef struct
{
   char *SerNo;
   char *DevNm;
   unsigned char hwVer;
   unsigned char devAvail;
} mir_sdr_DeviceT;
```

2.16 Gain Values Structure

Used to pass back current gain settings in mir_sdr_GetCurrentGain().

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

3 API Functions

The header mir_sdr.h defines the external function prototypes provided by this API. All functions are blocking.

3.1 Supported Functions

```
mir_sdr_ErrT mir_sdr_StreamInit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                                mir_sdr_If_kHzT ifType, int LNAstate, int *gRdBsystem,
                                mir_sdr_SetGrModeT setGrMode, int *samplesPerPacket,
                                mir_sdr_StreamCallback_t StreamCbFn,
                                mir_sdr_GainChangeCallback_t GainChangeCbFn, void *cbContext);

mir_sdr_ErrT mir_sdr_StreamUninit(void);

mir_sdr_ErrT mir_sdr_SetRf(double drfHz, int abs, int syncUpdate);

mir_sdr_ErrT mir_sdr_SetFs(double dfsHz, int abs, int syncUpdate, int reCal);

mir_sdr_ErrT mir_sdr_SetGr(int gRdB, int abs, int syncUpdate);

mir_sdr_ErrT mir_sdr_SetGrParams(int minimumGr, int lnaGrThreshold);

mir_sdr_ErrT mir_sdr_SetDcMode(int dcCal, int speedUp);

mir_sdr_ErrT mir_sdr_SetDcTrackTime(int trackTime);

mir_sdr_ErrT mir_sdr_SetSyncUpdateSampleNum(unsigned int sampleNum);

mir_sdr_ErrT mir_sdr_SetSyncUpdatePeriod(unsigned int period);

mir_sdr_ErrT mir_sdr_ApiVersion(float *version);

mir_sdr_ErrT mir_sdr_ResetUpdateFlags(int resetGainUpdate, int resetRfUpdate, int resetFsUpdate);

mir_sdr_ErrT mir_sdr_SetTransferMode(mir_sdr_TransferModeT mode);

mir_sdr_ErrT mir_sdr_DownConvert(short *in, short *xi, short *xq, unsigned int samplesPerPacket,
                                 mir_sdr_If_kHzT ifType, unsigned int M, unsigned int preReset);

mir_sdr_ErrT mir_sdr_SetPpm(double ppm);

mir_sdr_ErrT mir_sdr_SetLoMode(mir_sdr_LoModeT loMode);

mir_sdr_ErrT mir_sdr_SetGrAltMode(int *gRidx, int LNAstate, int *gRdBsystem, int abs,
                                  int syncUpdate);

mir_sdr_ErrT mir_sdr_DCoffsetIQimbalanceControl(unsigned int DCenable, unsigned int IQenable);

mir_sdr_ErrT mir_sdr_DecimateControl(unsigned int enable, unsigned int decimationFactor,
                                     unsigned int wideBandSignal);

mir_sdr_ErrT mir_sdr_AgcControl(mir_sdr_AgcControlT enable, int setPoint_dBfs, int knee_dBfs,
                                unsigned int decay_ms, unsigned int hang_ms, int syncAgcUpdate,
                                int lagcLNAstate);

mir_sdr_ErrT mir_sdr_Reinit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                            mir_sdr_If_kHzT ifType, mir_sdr_LoModeT loMode, int LNAstate,
                            int *gRdBsystem, mir_sdr_SetGrModeT setGrMode, int *samplesPerPacket,
                            mir_sdr_ReasonForReinitT reasonForReinit);

mir_sdr_ErrT mir_sdr_DebugEnable(unsigned int enable);

mir_sdr_ErrT mir_sdr_GetCurrentGain(mir_sdr_GainValuesT *gainVals);
```

```
mir_sdr_ErrT mir_sdr_GetDevices(mir_sdr_DeviceT *devices, unsigned int *numDevs,
                                unsigned int maxDevs);

mir_sdr_ErrT mir_sdr_SetDeviceIdx(unsigned int idx);

mir_sdr_ErrT mir_sdr_ReleaseDeviceIdx(void);

mir_sdr_ErrT mir_sdr_GetHwVersion(unsigned char *ver);

mir_sdr_ErrT mir_sdr_RSPII_AntennaControl(mir_sdr_RSPII_AntennaSelectT select);

mir_sdr_ErrT mir_sdr_RSPII_ExternalReferenceControl(unsigned int output_enable);

mir_sdr_ErrT mir_sdr_RSPII_BiasTControl(unsigned int enable);

mir_sdr_ErrT mir_sdr_RSPII_RfNotchEnable(unsigned int enable);

mir_sdr_ErrT mir_sdr_RSP_SetGr(int gRdB, int LNAstate, int abs, syncUpdate);

mir_sdr_ErrT mir_sdr_RSP_SetGrLimits(mir_sdr_MinGainReductionT minGr);

mir_sdr_ErrT mir_sdr_AmPortSelect(int port);

mir_sdr_ErrT mir_sdr_rsp1a_BiasT(int enable);

mir_sdr_ErrT mir_sdr_rsp1a_DabNotch(int enable);

mir_sdr_ErrT mir_sdr_rsp1a_BroadcastNotch(int enable);
```

3.2 Android Specific Functions

```
mir_sdr_ErrT mir_sdr_SetJavaReqCallback(mir_sdr_SendJavaReq_t sendJavaReq);
```

3.3 Deprecated Functions

These functions are no longer supported.

```
mir_sdr_ErrT mir_sdr_Init(int gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                          mir_sdr_If_kHzT ifType, int *samplesPerPacket);

mir_sdr_ErrT mir_sdr_Uninit(void);

mir_sdr_ErrT mir_sdr_ReadPacket(short *xi, short *xq, unsigned int *firstSampleNum,
                                int *grChanged, int *rfChanged, int *fsChanged);

mir_sdr_ErrT mir_sdr_GetGrByFreq(double rfMHz, mir_sdr_BandT *band, int *gRdB, int LNAstate,
                                 int *gRdBsystem, mir_sdr_SetGrModeT setGrMode);

mir_sdr_ErrT mir_sdr_SetParam(int ParamterId, int value);
```

3.4 Callback Function Prototypes

```
typedef void (*mir_sdr_StreamCallback_t)(short *xi, short *xq, unsigned int firstSampleNum,
                                         int grChanged, int rfChanged, int fsChanged,
                                         unsigned int numSamples, unsigned int reset,
                                         void *cbContext);

typedef void (*mir_sdr_GainChangeCallback_t)(unsigned int gRidx, unsigned int gRdB,
                                             unsigned int lnaGRdB, void *cbContext);

typedef int (*mir_sdr_SendJavaReq_t)(mir_sdr_JavaReqT cmd, int param);
```

3.5 mir_sdr_StreamInit

```
mir_sdr_ErrT mir_sdr_StreamInit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                                mir_sdr_If_kHzT ifType, int LNAstate, int *gRdBsystem,
                                mir_sdr_SetGrModeT setGrMode, int *samplesPerPacket,
                                mir_sdr_StreamCallback_t StreamCbFn,
                                mir_sdr_GainChangeCallback_t GainChangeCbFn, void *cbContext)
```

Description:

Replaces `mir_sdr_Init()`. It sets up a thread (or chain of threads) inside the API which will perform the processing chain, and then use the callback function to return the data to the calling application. Processing chain (in order):

`ReadPacket()`	fetch packet of IQ samples from USB interface
`Agc()`	disabled by default
`DCoffsetCorrection()`	enabled by default in LIF mode (otherwise disabled)
`DownConvert()`	automatically enabled if the parameters shown in section 0 are matched
`Decimate()`	disabled by default
`DCoffsetCorrection()`	enabled by default in ZIF mode (otherwise disabled)
`IQimbalanceCorrection()`	enabled by default

This function will set the default tuner based DC correction parameters to Periodic 3 with Speedup disabled – see `mir_sdr_SetDcMode()` for more details.

Parameters:

`gRdB`	Input value is the requested initial gain reduction in dB (see gain reduction tables referenced in section 5 for ranges and mappings), returns IF gain reduction value.
`fsMHz`	Specifies the sample frequency in MHz, values between 2MHz and 10MHz are permitted. Decimation can be used to obtain lower sample rates.
`rfMHz`	Specifies the tuner frequency in MHz, see frequency allocation tables, section 0.
`bwType`	Specifies the bandwidth to be used, see list in enumerated type for supported modes.
`ifType`	Specifies the IF to be used, see list in enumerated type for supported modes.
`LNAstate`	Specifies the LNA state: N/A → if `setGrMode == mir_sdr_USE_SET_GR` [0:1] → if `setGrMode == mir_sdr_USE_SET_GR_ALT_MODE` [0:3] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && RSP1` [0:4] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && AMport1 enabled && RSP2` [0:5] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf >= 420MHz && RSP2` [0:8] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf < 420MHz && RSP2`
`gRdBsystem`	Input value ignored, returns overall system gain reduction value (if `setGrMode != mir_sdr_USE_SET_GR`) .

`setGrMode`	Specifies gain mode to use:
	`mir_sdr_USE_SET_GR` → use `mir_sdr_SetGr()`
	`mir_sdr_USE_SET_GR_ALT_MODE` → use `mir_sdr_SetGrAltMode()`
	`mir_sdr_USE_RSP_SET_GR` → use `mir_sdr_RSP_SetGr()`

`samplesPerPacket`	Pointer to an unsigned integer which returns the number of samples that will be returned in the callback for the current configuration (may be modified by decimation rates). This will be a four times the value of `samplesPerPacket` returned by `mir_sdr_Init()` as there is additional buffering in the processing thread.
`StreamCbFn`	Specifies the callback function to use to send processed data.
`GainChangeCbFn`	Specifies the callback function to use when an AGC gain change happens to notify the application of the current gain reduction settings.
`cbContext`	Pointer to a context passed that will be returned as a parameter in the callbacks.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_AlreadyInitialised`	API has been initialised previously
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device
	`mir_sdr_Fail`	Other failure mechanism (thread/mutex create)

3.6 mir_sdr_StreamUninit

```
mir_sdr_ErrT mir_sdr_Uninit(void)
```

Description:

Stops the stream and uninitialises the API.

Parameters:

`void`	No parameters

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (thread destroy)

3.7 mir_sdr_SetRf

```
mir_sdr_ErrT mir_sdr_SetRf(double drfHz, int abs, int syncUpdate)
```

Description:

Adjusts the nominal tuner frequency maintained in the internal state of the API. Depending on the state of the `abs` parameter, the `drfHz` parameter is either applied as an offset from the internally stored state of the API or is used in an absolute manner to modify the internally stored state. This command will only permit frequency changes that fall within the restrictions of the frequency allocation tables shown in section 5

Parameters:

`drfHz`	Tuner frequency or tuner frequency offset in Hz. Once absolute value has been calculated (if required), it must be within the range of the current frequency band - see frequency allocation tables, section 0.
`abs`	Indicates if `drfHz` is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
`syncUpdate`	Indicates if the tuner frequency update is to be applied immediately or delayed until the next synchronous update point as configured in calls to `mir_sdr_SetSyncUpdateSampleNum()` and `mir_sdr_SetSyncUpdatePeriod()`: 0 → Immediate 1 → Synchronous

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_RfUpdateError`	Previous update has not yet been applied
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.8 mir_sdr_SetFs

```
mir_sdr_ErrT mir_sdr_SetFs(double dfsHz, int abs, int syncUpdate, int reCal)
```

Description:

Adjusts the nominal sample frequency maintained in the internal state of the API. Depending on the state of the `abs` parameter, the `dfsHz` parameter is either applied as an offset from the internally stored state of the API or is used in an absolute manner to modify the internal stored API state. This command will typically permit only small changes in sample frequency in the order of ±1000ppm. For large sample frequency changes a `mir_sdr_Uninit` and `mir_sdr_Init` at the new sample rate must be performed.

Parameters:

`dfsHz`	Sample frequency or sample frequency offset in Hz. Once absolute value has been calculated (if required), it must be in the range 2MHz to 10MHz.
`abs`	Indicates if `dfsHz` is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
`syncUpdate`	Indicates if the sample frequency update is to be applied immediately or delayed until the next synchronous update point as configured in calls to `mir_sdr_SetSyncUpdateSampleNum()` and `mir_sdr_SetSyncUpdatePeriod()`: 0 → Immediate 1 → Synchronous
`reCal`	Recalibration of the PLL. Note: this is normally done only when the nominal sample frequency is set in `mir_sdr_Init()` and should be set to 0 elsewhere: 0 → no recalibration is made 1 → force a recalibration of the PLL

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_FsUpdateError`	Previous update has not yet been applied
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.9 mir_sdr_SetGr

```
mir_sdr_ErrT mir_sdr_SetGr(int gRdB, int abs, int syncUpdate)
```

Description:

Programs the gain reduction required in the tuner. The `abs` parameter is used to determine whether the value specified is absolute gain reduction or an offset from the current gain value. The internal state is updated irrespective of what `abs` parameter is set to.

If the function completes successfully, just before it returns it calls the gain callback function with the updated parameters.

Parameters:

`gRdB`	Gain reduction or gain reduction offset in dB (see gain reduction tables referenced in section 5.1 for ranges and mappings)
`abs`	Indicates if `gRdB` is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
`syncUpdate`	Indicates if the gain reduction is to be applied immediately or delayed until the next synchronous update point as configured in calls to `mir_sdr_SetSyncUpdateSampleNum()` and `mir_sdr_SetSyncUpdatePeriod()`: 0 → Immediate 1 → Synchronous

Return:

`mir_sdr_ErrT` Error code as defined below:

`mir_sdr_Success`	Successful completion
`mir_sdr_NotInitialised`	API has not been initialised
`mir_sdr_GainUpdateError`	Previous update has not yet been applied
`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
`mir_sdr_HwError`	Failed to access device

3.10 mir_sdr_SetGrParams

```
mir_sdr_ErrT mir_sdr_SetGrParams(int minimumGr, int lnaGrThreshold)
```

Description:

Modifies the default gain reduction parameters required in the tuner. These are only applicable when using `mir_sdr_SetGr()`

Parameters:

`minimumGr`	Minimum gain reduction in dB that can be programmed.
`lnaGrThreshold`	Threshold at which the LNA will be switched in.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.11 mir_sdr_SetDcMode

```
mir_sdr_ErrT mir_sdr_SetDcMode(int dcCal, int speedUp)
```

Description:

Sets the DC offset correction mode for the tuner.

Parameters:

`dcCal`	DC offset correction mode: 0 → static 1 → Periodic 1 (Correction applied periodically every 6mS) 2 → Periodic 2 (Correction applied periodically every 12mS) 3 → Periodic 3 (Correction applied periodically every 24mS) 4 → one shot mode (correction applied each time gain update performed) 5 → continuous
`speedUp`	Speed up mode: 0 → disabled 1 → enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.12 mir_sdr_SetDcTrackTime

```
mir_sdr_ErrT mir_sdr_SetDcTrackTime(int trackTime)
```

Description:

Set the time period over which the DC offset is tracked when in one-shot mode.

Parameters:

`trackTime`	Tracking time period – valid range is 1 to 63 and the duration can be calculated as: Duration (us) = `3 * trackTime`

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.13 mir_sdr_SetSyncUpdateSampleNum

```
mir_sdr_ErrT mir_sdr_SetSyncUpdateSampleNum(unsigned int sampleNum)
```

Description:

Configures the sample number of the next synchronous update point. This is typically determined from the use of the `firstSampleNum` parameter returned in the `mir_sdr_ReadPacket()` function call. If the latency incurred over the USB causes this sample number to be set too late, the hardware will adjust automatically to correct for this.

Parameters:

`sampleNum`	Sample number of next synchronous update point.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_HwError`	Failed to access device

3.14 mir_sdr_SetSyncUpdatePeriod

```
mir_sdr_ErrT mir_sdr_SetSyncUpdatePeriod(unsigned int period)
```

Description:

The value set in this call is automatically added to the sample number of the last synchronous update point to determine the next one. Note – this function should be called before `mir_sdr_SetSyncUpdateSampleNum()`.

Parameters:

`period`	Defines the period between synchronous update points can be set between 1 and 1000000 samples.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_HwError`	Failed to access device

3.15 mir_sdr_ApiVersion

```
mir_sdr_ErrT mir_sdr_ApiVersion(float *version)
```

Description:

This function checks that the version in the include file is consistent with the dll version.

Parameters:

`version`	Pointer to a float which returns the version of the dll.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers

3.16 mir_sdr_ResetUpdateFlags

```
mir_sdr_ErrT mir_sdr_ResetUpdateFlags(int resetGainUpdate, int resetRfUpdate, int resetFsUpdate)
```

Description:

If it is detected that an update to one or more of Gain Reduction, Tuner Frequency or Sample Frequency has not completed within some application specific timeout period, the logic prohibiting further updates can be reset using this function. More than one update type can be reset in each call, and once reset, new updates can be scheduled.

Parameters:

`resetGainUpdate`	Reset Gain Reduction update logic: 0 → do not reset 1 → reset
`resetRfUpdate`	Reset Tuner Frequency update logic: 0 → do not reset 1 → reset
`resetFsUpdate`	Reset Sample Frequency update logic: 0 → do not reset 1 → reset

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised

3.17 mir_sdr_SetTransferMode

```
mir_sdr_ErrT mir_sdr_SetTransferMode(mir_sdr_TransferModeT mode);
```

Description:

Used to change USB streaming data transfer mode. Typically, this should not need to be changed and data loss may occur if the mode is changed.

Note: for ARM processor based platforms this function only allows `mir_sdr_BULK` to be used and any other requested value will return an `mir_sdr_OutOfRange` error.

Parameters:

`mode`	Mode selected: `mir_sdr_ISOCH` → Isochronous mode (default on non-ARM CPU platforms) `mir_sdr_BULK` → Bulk mode (default on ARM CPU platforms)	

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.18 mir_sdr_DownConvert

```
mir_sdr_ErrT mir_sdr_DownConvert(short *in, short *xi, short *xq, unsigned int samplesPerPacket,
                                 mir_sdr_If_kHzT ifType, unsigned int M, unsigned int preReset)
```

Description:

A command which converts the sampled IF data obtained from streamed data to I and Q data in a zero IF format. The functions converts from low IF to zero IF by mixing, filtering and decimating the sampled IF data. The function will only operate correctly for the parameters detailed in the table below.

IF Frequency	IF Bandwidth	Input Sample Rate	Output Sample Rate	Decimation Factor
450kHz	200kHz	2MS/s	0.5MS/s	4
450kHz	300kHz	2MS/s	0.5MS/s	4
450kHz	600kHz	2MS/s	1MS/s	2
2048kHz	1536kHz	8.192MS/s	2.048MS/s	4

Parameters:

`in`	Pointer to an array of size (`samplesPerPacket * sizeof(short)`) in which the I samples returned from streamed data are contained. If a non-zero IF mode this array will contain the sampled IF data.
`xi`	Pointer to an array (of minimum size ((`samplesPerPacket/M) * sizeof(short)`) in which the down-converted I samples will be returned.
`xq`	Pointer to an array (of minimum size ((`samplesPerPacket/M) * sizeof(short)`) in which the down-converted Q samples will be returned.
`samplesPerPacket`	An unsigned integer which contains the number of samples that are contained in the input IF sampled data array `in`.
`ifType`	Specifies the IF bandwidth that has been configured, see list in enumerated type for supported modes.
`M`	Desired decimation factor, see table above for list of applicable values.
`preReset`	If `preReset` is equal to 1 then the filtering state will be reset prior to any filtering operation.

Return:

`mir_sdr_ErrT` Error code as defined below:

`mir_sdr_Success`	Successful completion
`mir_sdr_NotInitialised`	API has not been initialised
`mir_sdr_InvalidParam`	NULL pointers
`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.19 mir_sdr_SetPpm

```
mir_sdr_ErrT mir_sdr_SetPpm(double ppm)
```

Description:

To specify a correction factor used to account for offsets from the nominal in the crystal oscillator.

Parameters:

`ppm`	Parts per million offset (e.g. +/- 1 ppm specifies a +/- 24Hz error for a 24MHz crystal).

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_RfUpdateError`	Previous update has not yet been applied
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.20 mir_sdr_SetLoMode

```
mir_sdr_ErrT mir_sdr_SetLoMode(mir_sdr_LoModeT loMode)
```

Description:

Allows a particular up-converter (1st) LO frequency to be specifed or selects automatic mode which allows the API to determine the most appropriate 1st LO frequency across all tuner frequency ranges. This function must be called before the API is initialized – otherwise use mir_sdr_ReInit()

Parameters:

`loMode`	Specifies 1st LO frequency (see `mir_sdr_LoModeT` for possible values).

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_AlreadyInitialised`	API has been initialised previously
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.21 mir_sdr_SetGrAltMode

```
mir_sdr_ErrT mir_sdr_SetGrAltMode(int *gRidx, int LNAstate, int *gRdBsystem, int abs, int syncUpdate)
```

Description:

Alternative method (to `mir_sdr_SetGR()`) to set the gain reduction based on internal gain tables for the current band, requested gain reduction and whether the LNA is enabled or not.

If the function completes successfully, just before it returns it calls the gain callback function with the updated parameters.

Parameters:

`gRidx`	Input value is the requested gain reduction index or gain reduction offset (see gain reduction tables referenced in section 5.2 for ranges and mappings), returns current gain reduction index (which may be different from the input value when not in absolute mode or when `LNAstate` changes).
`LNAstate`	Tuner LNA control: 0 → disable LNA (increases gain reduction) 1 → enable LNA (decreases gain reduction)
`gRdBsystem`	Input value ignored, returns overall system gain reduction.
`abs`	Indicates if `gRdB` is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
`syncUpdate`	Indicates if the gain reduction is to be applied immediately or delayed until the next synchronous update point as configured in calls to `mir_sdr_SetSyncUpdateSampleNum()` and `mir_sdr_SetSyncUpdatePeriod()`: 0 → Immediate 1 → Synchronous

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_GainUpdateError`	Previous update has not yet been applied
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.22 mir_sdr_DCoffsetIQimbalanaceControl

```
mir_sdr_ErrT mir_sdr_DCoffsetIQimbalanceControl(unsigned int DCenable, unsigned int IQenable)
```

Description:

Individual enables for DC offset correction and IQ imbalance correction.

Parameters:

`DCenable`	DC offset correction control: 0 → DC correction disabled 1 → DC correction enabled (default)
`IQenable`	IQ correction control: 0 → IQ correction disabled 1 → IQ correction enabled (default)

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion

3.23 mir_sdr_DecimateControl

```
mir_sdr_ErrT mir_sdr_DecimateControl(unsigned int enable, unsigned int decimationFactor,
                                     unsigned int wideBandSignal)
```

Description:

Used to control whether decimation is enabled or not. Valid decimation factors are 2, 4, 8, 16, 32 or 64 only. If other values are specified then decimation will not be enabled. If wide band mode is selected, the decimation algorithm uses a sequence of half-band filters to achieve the required decimation, otherwise a box-filter is used which is much more efficient but may cause roll-off in the passband of the received signal depending on bandwidth. Otherwise, a simple block averaging is used to reduce the CPU load, but with increased in-band roll-off.

Note: Requires internal stream thread to have been created via `mir_sdr_StreamInit()` for decimation to be enabled. Also for IQ output in Low IF mode, enable must = 0 as the decimation is automatic within the API.

Parameters:

`enable`	Decimation control 0 → Decimation disabled (default) 1 → Decimation enabled
`decimationFactor`	Decimation factor (2, 4, 8, 16 or 32 only).
`widebandSignal`	Filter control: 0 → Use averaging (default) 1 → Use half-band filter

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_AliasingError`	Requested parameters will cause aliasing
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.24 mir_sdr_AgcControl

```
mir_sdr_ErrT mir_sdr_AgcControl(mir_sdr_AgcControlT enable, int setPoint_dBfs, int knee_dBfs,
                                unsigned int decay_ms, int hang_ms, int syncUpdate,
                                int lnaState)
```

Description:

Used to control whether AGC is enabled or not and parameters to allow the AGC to be configured.

Note: Requires internal stream thread to have been created via `mir_sdr_StreamInit()` for AGC to be enabled.

Parameters:

`enable`	Specifies the AGC mode required. See enumerated types for valid values. Default mode is 100Hz (`mir_sdr_100HZ`).
`setPoint_dBfs`	Specifies the required set point in dBfs.
`knee_dBfs`	Not currently used, set to 0.
`decay_ms`	Not currently used, set to 0.
`hang_ms`	Not currently used, set to 0.
`syncAgcUpdate`	Update control: 0 → immediate update 1 → synchronous update (see `mir_sdr_SetGR()` for more details)
`lagcLNAstate`	Indicates the LNA state to use in gain updates when AGC uses `mir_sdr_SetGrAltMode()` or `mir_sdr_RSP_SetGr()` as specified when calling `mir_sdr_StreamInit()`.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.25 mir_sdr_Reinit

```
mir_sdr_ErrT mir_sdr_Reinit(int *gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                            mir_sdr_If_kHzT ifType, mir_sdr_LoModeT loMode, int LNAstate,
                            int *gRdBsystem, mir_sdr_SetGrModeT setGrMode, int *samplesPerPacket,
                            mir_sdr_ReasonForReinitT reasonForReinit)
```

Description:

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. Only those parameters required for the requested reason(s) need to be provided (except pointer parameters whick should be valid pointers irrespective of the `reasonForReinit`) – as specified below:

`mir_sdr_CHANGE_GR: gRdB, LNAstate, gRdBsystem, setGrMode`	does not require Uninit/Init
`mir_sdr_CHANGE_FS_FREQ: fsMHz`	requires Uninit/Init
`mir_sdr_CHANGE_RF_FREQ: rfMHz`	does not require Uninit/Init
`mir_sdr_CHANGE_BW_TYPE: bwType`	requires Uninit/Init
`mir_sdr_CHANGE_IF_TYPE: ifType`	requires Uninit/Init
`mir_sdr_CHANGE_LO_MODE: loMode`	requires Uninit/Init
`mir_sdr_CHANGE_AM_PORT: (preceeded by mir_sdr_AmPortSelect())`	requires Uninit/Init

Note: if a Uninit/Init is required and synchronous updates have previously been configured, then they will need to re-configured using `SetSyncUpdateSampleNum()` and `SetSyncUpdatePeriod()` when `mir_sdr_Reinit()` completes.

Parameters:

`gRdB`	Input value is the request gain reduction or gain reduction offset in dB (see gain reduction tables referenced in section 5 for ranges and mappings), returns IF gain reduction value.
`fsMHz`	Specifies the sample frequency in MHz, typical values between 2MHz and 10MHz. Decimation can be used to obtain lower sample rates.
`rfMHz`	Specifies the tuner frequency in MHz, see frequency allocation tables, section 0.
`bwType`	Specifies the bandwidth to be used, see list in enumerated type for supported modes.
`ifType`	Specifies the IF to be used, see list in enumerated type for supported modes.
`LNAstate`	Specifies the LNA state: N/A → if `setGrMode == mir_sdr_USE_SET_GR` [0:1] → if `setGrMode == mir_sdr_USE_SET_GR_ALT_MODE` [0:3] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && RSP1` [0:4] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && AMport1 enabled && RSP2` [0:5] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf >= 420MHz && RSP2` [0:8] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf < 420MHz && RSP2`
`gRdBsystem`	Input value ignored, returns overall system gain reduction value (if `setGrMode != mir_sdr_USE_SET_GR`) .

`setGrMode`	Specifies gain mode to use:
	`mir_sdr_USE_SET_GR` → use `mir_sdr_SetGr()`
	`mir_sdr_USE_SET_GR_ALT_MODE` → use `mir_sdr_SetGrAltMode()`
	`mir_sdr_USE_RSP_SET_GR` → use `mir_sdr_RSP_SetGr()`

`samplesPerPacket`	Pointer to an unsigned integer which returns the number of samples that will be returned in the callback for the current configuration if `mir_sdr_StreamInit()` was used for initialization, or the number of samples returned by `mir_sdr_ReadPacket()`for `mir_sdr_Init()` (may be modified by decimation rates).
`reasonforReinit`	Used to specify the reason for a reinitialise request - values should be or'd together if there are multiple reasons for the request.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_AliasingError`	Requested parameters can cause aliasing
	`mir_sdr_HwError`	Failed to access device
	`mir_sdr_Fail`	Other failure mechanism (thread create)

3.26 mir_sdr_DebugEnable

```
mir_sdr_ErrT mir_sdr_DebugEnable(unsigned int enable)
```

Description:

Used to enable debug message output.

Parameters:

`enable`	Debug output control: 0 → messages disabled (default) 1 → messages enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion

3.27 mir_sdr_GetCurrentGain

```
mir_sdr_ErrT mir_sdr_GetCurrentGain(mir_sdr_GainValuesT *gainVals);
```

Description:

Used to return the current gain setting based on the current gain reduction value and the total system gain determined for the tuner frequency, IF BW and ZIF/LIF settings. Also, returns the maximum and minimum gain for these settings.

Parameters:

`gainVals`	Pointer to structure to hold gain values.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_Fail`	Other failure mechanism (device not RSP1/2/1A)

3.28 mir_sdr_GetDevices

```
mir_sdr_ErrT mir_sdr_GetDevices(mir_sdr_DeviceT *devices, unsigned int *numDevs,
                                unsigned int maxDevs);
```

Description:

Returns list of RSP1, RSP2 and RSP1A devices along with the serial number and unique reference (`DevNm`) and type. The index in the returned list is used with `mir_sdr_SetDeviceIdx()` to select which device to use. These functions must be called prior to `mir_sdr_StreamInit()` or `mir_sdr_Init()` otherwise the first device in the list will automatically be chosen.

All RSPx devices will be listed in the `devices` structure up to the maximum number of entries in the table as specified by `maxDevs`, with the actual number in the list being returned in numDevs. However, if a device is already in use, the devAvail flag will be set to 0 (instead of 1 if it is available).

Parameters:

`devices`	Pointer to array of `mir_sdr_DeviceT` structures allocated in the application. The array must be able to contain at least `maxDevs` structures.
`numDevs`	Pointer to variable which on return will indicate the number of devices found (or `maxDevs` if there are more than the size of array allocated).
`maxDevs`	Maximum number of devices to be returned.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers

3.29 mir_sdr_SetDeviceIdx

```
mir_sdr_ErrT mir_sdr_SetDeviceIdx(unsigned int idx);
```

Description:

Used to select which device to open. Prior to each call to this function, a call must be made to `mir_sdr_GetDevices()`.

Note: the device must be released using `mir_sdr_ReleaseDeviceIdx()` before exiting the application otherwise the device may not be available for subsequent use.

Parameters:

`idx`	Index of device to be opened. This is the array index of the structures returned in a prior call to `mir_sdr_GetDevices()`.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_HwError`	Failed to access device

3.30 mir_sdr_ReleaseDeviceIdx

```
mir_sdr_ErrT mir_sdr_ReleaseDeviceIdx(void);
```

Description:

Used to release a device previously selected using `mir_sdr_SetDeviceIdx()`. If `mir_sdr_SetDeviceIdx()` has been used, then this function must be called prior to exiting the application.

Parameters:

`void`	No parameters

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_HwError`	Failed to access device

3.31 mir_sdr_GetHwVersion

```
mir_sdr_ErrT mir_sdr_GetHwVersion(unsigned char *ver);
```

Description:

Returns the hardware version of the device currently in use.

Parameters:

`ver`	Pointer to variable that on return will contain the version of the device in use.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers

3.32 mir_sdr_RSPII_AntennaControl

```
mir_sdr_ErrT mir_sdr_RSPII_AntennaControl(mir_sdr_RSPII_AntennaSelectT select);
```

Description:

Selects which antenna to use on an RSP2 device. If the device is already opened, it will change the port immediately, otherwise it will take effect on initialisation (if selected device is an RSP2).

Note, when operating in the AM band, the `mir_sdr_AmPortSelect()` API function may be used to select the Hi-Z input instead of Antenna A or Antenna B.

Parameters:

`select`	Selects antenna:	
	`mir_sdr_RSPII_ANTENNA_A` → Antenna A	
	`mir_sdr_RSPII_ANTENNA_B` → Antenna B	

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_HwVerError`	Incorrect device (not RSP2)

3.33 mir_sdr_RSPII_ExternalReferenceControl

```
mir_sdr_ErrT mir_sdr_RSPII_ExternalReferenceControl(unsigned int output_enable);
```

Description:

Used to enable the external reference output port on a RSP2 device. If this is called before the device is opened (but after a device has been selected with `mir_sdr_SetDeviceIdx()`), then the device will be opened, the external reference enabled and the device closed again.

Parameters:

`output_enable`	External reference output control: 0 → output is disabled (default) 1 → output is enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (mutex create)
	`mir_sdr_HwVerError`	Incorrect device (not RSP2)

3.34 mir_sdr_RSPII_BiasTControl

```
mir_sdr_ErrT mir_sdr_RSPII_BiasTControl(unsigned int enable);
```

Description:

Used to enable the BiasT network on a RSP2 device. If this is called before the device is opened (but after a device has been selected with `mir_sdr_SetDeviceIdx()`), then the device will be opened, the BiasT network will be enabled and the device closed again.

Parameters:

`enable`	BiasT enable control: 0 → BiasT disabled (default) 1 → BiasT enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (mutex create)
	`mir_sdr_HwVerError`	Incorrect device (not RSP2)

3.35 mir_sdr_RSPII_RfNotchEnable

```
mir_sdr_ErrT mir_sdr_RSPII_RfNotchEnable(unsigned int enable);
```

Description:

Enables the RF Notch Filter on a RSP2 device. If the device is already opened, it will change the settings immediately, otherwise it will take effect on initialisation (if selected device is an RSP2).

Parameters:

`enable`	RF Notch Filter control: 0 → filter disabled (default) 1 → filter enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_HwVerError`	Incorrect device (not RSP2)

3.36 mir_sdr_RSP_SetGr

```
mir_sdr_ErrT mir_sdr_RSP_SetGr(int gRdB, int LNAstate, int abs, int syncUpdate);
```

Description:

Alternative method (to `mir_sdr_SetGR()`, for RSP1, RSP2 or RSP1A products) to set the gain reduction based on the requested IF gain reduction and which state the LNA is in.

The total gain reduction applied can be calculated from:

```
gRdb + LNA GR as defined by the current band and the table in section 5.3
```

If the function completes successfully, just before it returns it calls the gain callback function with the updated parameters.

Parameters:

`gRdB`	Requested IF gain reduction or gain reduction offset (see `mir_sdr_RSP_SetGrLimits()` for details of the valid range for this parameter).
`LNAstate`	Specifies the LNA state (see gain reduction tables referenced in section 5.3 for ranges and mappings): [0:3] → if `RSP1` [0:4] → if `AMport1 enabled && RSP2` [0:5] → if `Frf >= 420MHz && RSP2` [0:8] → if `Frf <  420MHz && RSP2`
`abs`	Indicates if `gRdB` is an absolute value or offset from previously set value: 0 → Offset Mode 1 → Absolute Mode
`syncUpdate`	Indicates if the gain reduction is to be applied immediately or delayed until the next synchronous update point as configured in calls to `mir_sdr_SetSyncUpdateSampleNum()` and `mir_sdr_SetSyncUpdatePeriod()`: 0 → Immediate 1 → Synchronous

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_GainUpdateError`	Previous update has not yet been applied
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_HwError`	Failed to access device

3.37 mir_sdr_RSP_SetGrLimits

```
mir_sdr_ErrT mir_sdr_RSP_SetGrLimits(mir_sdr_MinGainReductionT minGr);
```

Description:

This function provides a mechanism to extend the normally useful IF gain reduction range to include values less than 20dB when using `mir_sdr_RSP_SetGr()`. It is not recommended to use the extended range as it is likely to overload to produce signal levels that will overload the ADC input.

Note: that the extended range cannot be used by the internal AGC algorithm and will be disabled when enabling the AGC. If the AGC is subsequently disabled, the extended range will not be re-selected automatically.

Parameters:

`minGr`	Indicates the minimum IF gain reduction to extend the range: `mir_sdr_NORMAL_MIN_GR` → minimum IF GR of 20, range of 20 to 59 `mir_sdr_EXTENDED_MIN_GR` → minimum IF GR of 0, range of 0 to 59

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_NotEnabled`	The requested change has not been applied
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.38 mir_sdr_AmPortSelect

```
mir_sdr_ErrT mir_sdr_AmPortSelect(int port);
```

Description:

Used to select which AM port to use on a RSP2 device, when the requested tuner frequency is within the AM band. This function must be called before the device is initialised, otherwise it will not have any effect.

Parameters:

`port`	AM port control: 0 → Port 0 (Antenna A or Antenna B inputs - default) 1 → Port 1 (Hi-Z input)

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_AlreadyInitialised`	API has been initialised previously
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.39 mir_sdr_rsp1a_BiasT

```
mir_sdr_ErrT mir_sdr_rsp1a_BiasT(int enable);
```

Description:

Used to enable the BiasT network on a RSP1A device. If this is called before the device is opened (but after a device has been selected with `mir_sdr_SetDeviceIdx()`), then the device will be opened, the BiasT network will be enabled and the device closed again.

Parameters:

`enable`	BiasT enable control: 0 → BiasT disabled (default) 1 → BiasT enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (mutex create)
	`mir_sdr_HwVerError`	Incorrect device (not RSP1A)

3.40 mir_sdr_rsp1a_DabNotch

```
mir_sdr_ErrT mir_sdr_rsp1a_DabNotch(int enable);
```

Description:

Used to enable the DAB notch on a RSP1A device. If this is called before the device is opened (but after a device has been selected with `mir_sdr_SetDeviceIdx()`), then the device will be opened, the DAB notch will be enabled and the device closed again.

Parameters:

`enable`	DAB notch enable control: 0 → DAB notch disabled (default) 1 → DAB notch enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (mutex create)
	`mir_sdr_HwVerError`	Incorrect device (not RSP1A)

3.41 mir_sdr_rsp1a_BroadcastNotch

```
mir_sdr_ErrT mir_sdr_rsp1a_BroadcastNotch(int enable);
```

Description:

Used to enable the Broadcast notch on a RSP1A device. If this is called before the device is opened (but after a device has been selected with `mir_sdr_SetDeviceIdx()`), then the device will be opened, the Broadcast notch will be enabled and the device closed again.

Parameters:

`enable`	Broadcast notch enable control: 0 → Broadcast notch disabled (default) 1 → Broadcast notch enabled

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_Fail`	Other failure mechanism (mutex create)
	`mir_sdr_HwVerError`	Incorrect device (not RSP1A)

3.42 mir_sdr_SetJavaReqCallback

```
mir_sdr_ErrT mir_sdr_SetJavaReqCallback(mir_sdr_SendJavaReq_t sendJavaReq);
```

Description:

When the application is running under Java on Android, the callback registered with this function is used to indicate to the application when the events defined by `mir_sdrJavaReq_t` take place that may require information or action from the application.

Parameters:

`sendJavaReq`	Specifies the callback function to use to send the event data.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers

3.43 mir_sdr_Init

```
mir_sdr_ErrT mir_sdr_Init(int gRdB, double fsMHz, double rfMHz, mir_sdr_Bw_MHzT bwType,
                          mir_sdr_If_kHzT ifType, int *samplesPerPacket)
```

Description:

Initiates the API for the specified tuner frequency.

Parameters:

`gRdB`	Inital gain reduction (see gain reduction tables referenced in section 5.1 for ranges and mappings).
`fsMHz`	Specifies the sample frequency in MHz, values between 2MHz and 10MHz are permitted. Decimation can be used to obtain lower sample rates.
`rfMHz`	Specifies the tuner frequency in MHz, see frequency allocation tables, section 0.
`bwType`	Specifies the bandwidth to be used, see list in enumerated type for supported modes.
`ifType`	Specifies the IF to be used, see list in enumerated type for supported modes.
`samplesPerPacket`	Pointer to an unsigned integer which returns the number of samples that will be returned in each call to `mir_sdr_ReadPacket()`.

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_AlreadyInitialised`	API has been initialised previously
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range
	`mir_sdr_AliasingError`	Requested parameters can cause aliasing
	`mir_sdr_HwError`	Failed to access device
	`mir_sdr_Fail`	Other failure mechanism (mutex create)

3.44 mir_sdr_Uninit

```
mir_sdr_ErrT mir_sdr_Uninit(void)
```

Description:

Uninitialises the API.

Parameters:

`void`	No parameters

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised

3.45 mir_sdr_ReadPacket

```
mir_sdr_ErrT mir_sdr_ReadPacket(short *xi, short *xq, unsigned int *firstSampleNum,
                                int *grChanged, int *rfChanged, int *fsChanged)
```

Description:

This function is used to retrieve data from the hardware. The data is returned as 16bit left justified integers

Parameters:

`xi`	Pointer to an array (of minimum size `samplesPerPacket * sizeof(short)`) in which the I samples will be returned. If a non-zero IF is used, this array will contain the sampled IF data.
`xq`	Pointer to an array (of minimum size `samplesPerPacket * sizeof(short)`) in which the Q samples will be returned.
`firstSampleNum`	Pointer to an unsigned integer in which the sample count (modulo 2^32) of the first sample of the retrieved data will be returned.
`grChanged`	Pointer to an integer which indicates if the gain reduction has changed: 0 → no change 1 → changed
`rfChanged`	Pointer to an integer which indicates if the tuner frequency has changed: 0 → no change 1 → changed
`fsChanged`	Pointer to an integer which indicates if the sample frequency has changed: 0 → no change 1 → changed

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_NotInitialised`	API has not been initialised
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_HwError`	Failed to access device

3.46 mir_sdr_GetGrByFreq

```
mir_sdr_ErrT mir_sdr_GetGrByFreq(double rfMHz, mir_sdr_BandT *band, int *gRdB, int LNAstate,
                                 int *gRdBsystem, mir_sdr_SetGrModeT setGrMode)
```

Description:

Get the gain reduction settings for a gain reduction request for a particular tuner frequency. This function does not update the hardware (except when called before `mir_sdr_Init()` or `mir_sdr_StreamInit()`). It also determines the band the tuner frequency is in and returns it.

Parameters:

`rfMHz`	Specifies the tuner frequency in MHz, see frequency allocation tables, section 0.
`band`	Input value ignored, returns band for requested tuner frequency.
`gRdB`	Input value is the requested gain reduction or gain reduction offset in dB (see gain reduction tables referenced in section 5 for ranges and mappings), returns IF gain reduction value.
`LNAstate`	Specifies the LNA state: N/A → if `setGrMode == mir_sdr_USE_SET_GR` [0:1] → if `setGrMode == mir_sdr_USE_SET_GR_ALT_MODE` [0:3] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && RSP1` [0:4] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && AMport1 enabled && RSP2` [0:5] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf >= 420MHz && RSP2` [0:8] → if `setGrMode == mir_sdr_USE_RSP_SET_GR && Frf < 420MHz && RSP2`
`gRdBsystem`	Input value ignored, returns overall system gain reduction value (if `setGrMode != mir_sdr_USE_SET_GR`) .

`setGrMode`	Specifies gain mode to use:
	`mir_sdr_USE_SET_GR` → use `mir_sdr_SetGr()`
	`mir_sdr_USE_SET_GR_ALT_MODE` → use `mir_sdr_SetGrAltMode()`
	`mir_sdr_USE_RSP_SET_GR` → use `mir_sdr_RSP_SetGr()`

Return:

`mir_sdr_ErrT`	Error code as defined below:	
	`mir_sdr_Success`	Successful completion
	`mir_sdr_InvalidParam`	NULL pointers
	`mir_sdr_OutOfRange`	Requested parameters outside of allowed range

3.47 mir_sdr_SetParam

```
mir_sdr_ErrT mir_sdr_SetParam (int ParamterId, int value)
```

Description:

A command designed to allow the setting of various hardware control parameters. Currently the command only accepts one parameter ID which is 101 and this allows the hardware to configure a change to the 1st LO frequency. This command must be executed before mir_sdr_Init to configure the hardware prior to initialization.

Parameters:

ParamterId	101	→ Update 1st LO Frequency
value	19200000	→1st LO Frequency = 168MHz
	22000000	→1st LO Frequency = 144MHz
	24576000	→1st LO Frequency = 120MHz

Return:

mir_sdr_ErrT	Error code as defined below:	
	mir_sdr_Success	Successful completion
	mir_sdr_AlreadyInitialised	API has been initialised previously
	mir_sdr_OutOfRange	Requested parameters outside of allowed range

3.48 Stream callback

```
typedef void (*mir_sdr_StreamCallback_t)(short *xi, short *xq, unsigned int firstSampleNum,
                                         int grChanged, int rfChanged, int fsChanged,
                                         unsigned int numSamples, unsigned int reset,
                                         void *cbContext);
```

Description:

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

Parameters:

`xi`	Pointer to real data in the buffer.
`xq`	Pointer to the imaginary data in the buffer.
`firstSampleNum`	Number of first sample in buffer (used for synchronous updates). Note that this is divided in the API by the `decimationFactor`, to make it relative to the output sample rate. The values specified for sample number and period for synchronous updates should also be relative to the output sample rate.
`grChanged`	Indicates when the gain reduction has changed: Bit0 → Change status of 1st packet: 0=>no change, 1=> change occurred Bit1 → Change status of 2nd packet: 0=>no change, 1=> change occurred Bit2 → Change status of 3rd packet: 0=>no change, 1=> change occurred Bit3 → Change status of 4th packet: 0=>no change, 1=> change occurred
`rfChanged`	Indicates when the tuner frequency has changed: Bit0 → Change status of 1st packet: 0=>no change, 1=> change occurred Bit1 → Change status of 2nd packet: 0=>no change, 1=> change occurred Bit2 → Change status of 3rd packet: 0=>no change, 1=> change occurred Bit3 → Change status of 4th packet: 0=>no change, 1=> change occurred
`fsChanged`	Indicates when the sample frequency has changed: Bit0 → Change status of 1st packet: 0=>no change, 1=> change occurred Bit1 → Change status of 2nd packet: 0=>no change, 1=> change occurred Bit2 → Change status of 3rd packet: 0=>no change, 1=> change occurred Bit3 → Change status of 4th packet: 0=>no change, 1=> change occurred
`numSamples`	The number of samples in the current buffer.
`reset`	Indicates if a re-initialisation has occurred and that the local buffering should be reset.
`cbContext`	Pointer to context passed into `mir_sdr_StreamInit()`.

Return:

`void`	No return value.

3.49 Gain Change callback

```
typedef void (*mir_sdr_GainChangeCallback_t)(unsigned int gRdB, unsigned int lnaGRdB,
                                             void *cbContext);
```

Description:

This callback is triggered whenever a gain update occurs.

Parameters:

`gRdB`	New IF gain reduction value applied by the gain update. This parameter is also used to pass messages from API to the application as defined in section 2.12.
`lnaGRdB`	LNA gain reduction value.
`cbContext`	Pointer to context passed into `mir_sdr_StreamInit()`.

Return:

`void`	No return value.

3.50 Java Event callback

```
typedef int (*mir_sdr_SendJavaReq_t)(mir_sdr_JavaReqT cmd, int param);
```

Description:

This callback is triggered by certain events in the API that require attention from the Java application.

Parameters:

`cmd`	Command/event to be acted upon.
`param`	Data value for event.

Return:

`int`	Data returned from application (value specific to each event).

4 API Usage

The example code (for Windows) below shows how the calls are typically used - note that no error processing is shown.

```
#include "windows.h"
#include "mir_sdr.h"

HANDLE dataAvailSema = NULL;

void callback(short *xi, short *xq, unsigned int firstSampleNum, int grChanged, int rfChanged,
              int fsChanged, unsigned int numSamples, unsigned int reset, void *cbContext)
{
   // put data in a buffer to be used in main loop
   // ...
   // signal semaphore (pseudo code)
   ReleaseSemaphore(dataAvailSema, 1,NULL);
   return;
}

void callbackGC(unsigned int gRdB, unsigned int lnaGRdB, void *cbContext)
{
   // do something with updated gain information if required
   // ...
   return;
}

int main(void)
{
   // data declarations
   mir_sdr_ErrT err;
   int sps;
   float ver;
   int newGr = 40;
   int sysGr = 40;
   int done = 0;
   int syncUpdate = 0;    // initially use asynchronous updates
   int period;
   int sampleNum;
   DWORD ret;
   // ...

   dataAvailSema = CreateSemaphore(NULL, 0, 10, NULL);  // create semaphore

   // check API version
   err = mir_sdr_ApiVersion(&ver);
   if (ver != MIR_SDR_API_VERSION)
   {
      // Error detected, include file does not match dll. Deal with error condition.
   }

   // enable debug output
   mir_sdr_DebugEnable(1);

   // disable DC offset and IQ imbalance correction (default is for these to be enabled - this
   // just show how to disable if required)
   mir_sdr_DCoffsetIQimbalanceControl(1, 0);

   // disable decimation and set decimation factor to 4  (this is for information only as
   // decimation is disabled by default)
   mir_sdr_DecimateControl(0, 4, 0);

   // enable AGC with a setPoint of -30dBfs
   mir_sdr_AgcControl(1, -30, 0, 0, 0, 0, 1);

   // initialise API and hardware for DAB demodulation: initial gain reduction of 40dB, sample
   // rate of 2.048MHz, tuner frequency of 222.064MHz, double sided bandwidth of 1.536MHz and
   // a zero-IF

   // this also configures the API to use the new mir_sdr_SetGrAltMode() to
   // control the gain and disables the LNA. The overall system gain is returned in sysGr

   // used for DAB type signals
```

```
   err = mir_sdr_StreamInit(&newGr, 2.048, 222.064, mir_sdr_BW_1_536, mir_sdr_IF_Zero, 1, &sysGr,
                            mir_sdr_USE_RSP_SET_GR, &sps, callback, callbackGC,
                            (void *)NULL);
   if (err != mir_sdr_Success)
   {
      // Error detected. Deal with error condition.
   }

   // configure DC tracking in tuner
   err = mir_sdr_SetDcMode(4, 1);     // select one-shot tuner DC offset correction with speedup
   err = mir_sdr_SetDcTrackTime(63);  // with maximum tracking time
   // ...

   // processing loop (should be in separate thread probably)
   while(!done)
   {
      // ...
      // wait on semaphore for samples to be received in callback  (pseudo code)
      if ((ret = WaitForSingleObject(dataAvailSema, 100)) != WAIT_OBJECT_0)
      {
         // Error detected. Deal with error condition
         continue;
      }
      // get data from buffer
      // ...
      // tuner frequency tracking
      // ...
      // sample frequency tracking
      // ...
      // demodulate data (pass I and Q data to demod)
      // ...
      // for DAB, it may be required to do all updates during NULL symbol, so once demodulator
      // has established where this is, pass the details to the hardware and switch to using
      // synchronous updates
      mir_sdr_SetSyncUpdatePeriod(period);          // set period first
      mir_sdr_SetSyncUpdateSampleNum(sampleNum);    // then set sample number at start of NULL
      syncUpdate = 1;
          // ...
   }
   // At exit
   err = mir_sdr_StreamUninit();
}
```

5 Gain Reduction Tables

5.1 mir_sdr_SetGr()

See document: Default_Gain_Tables.pdf

5.2 mir_sdr_SetGrAltMode()

See document: RSPI_Gain_Tables.pdf

LNA GR by Band and LNAstate:

Band	LNAstate	
	0	1
AM	24	0
VHF	24	0
Band 3	24	0
Band X	24	0
Band 4/5	7	0
L Band	5	0

5.3 mir_sdr_RSP_SetGr()

LNA GR by Band and LNAstate for RSP1:

Band	LNAstate			
	0	1	2	3
AM	0	24	19[1]	43[2]
VHF	0	24	19[1]	43[2]
Band 3	0	24	19[1]	43[2]
Band X	0	24	19[1]	43[2]
Band 4/5	0	7	19[1]	26[2]
L Band	0	5	19[1]	24[2]

LNA GR by Band and LNAstate for RSP2:

Band	LNAstate								
	0	1	2	3	4	5	6	7	8
AM	0	10	15	21	24	34	39	45	64[2]
VHF	0	10	15	21	24	34	39	45	64[2]
Band 3	0	10	15	21	24	34	39	45	64[2]
Band X	0	10	15	21	24	34	39	45	64[2]
Band 4/5	0	7	10	17	22	41[2]			
L Band	0	5	21	15[3]	15[3]	34[2]			
AM (Port 1)	0	6	12	18	37[2]				

[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 Frequency Allocation Tables

Band	Minimum Frequency (MHz)	Maximum Frequency (MHz)
AM	0	60
VHF	60	120
Band 3	120	250
Band X	250	420
Band 4/5	420	1000
L Band	1000	2000
AM (Port 1) – RSP2 only	0	60

When using the `mir_sdr_StreamInit()` or `mir_sdr_Init()` command any tuner frequency range supported by the hardware can be programmed and this will configure the front end accordingly. Once the desired frequency has been programmed the `mir_sdr_setRf()` command can be used to alter the tuner frequency. It should be noted though, that the `mir_sdr_setRf()` command can only change the frequency within a set of predefined bands. If a frequency is desired that falls outside the current band then a `mir_sdr_Uninit()` command must be issued followed by a `mir_sdr_Init()` command at the new frequency to force reconfiguration of the front end. The table below shows the frequency bands over which the `mir_sdr_setRf()` commands will permit operation.

Alternatively, with this version of the API, the `mir_sdr_Reinit()` command can be used to change the frequency without requiring a `mir_sdr_Uninit()`/`mir_sdr_Init()` when crossing band boundaries.

For more information, contact: support@SDRplay.com

Legal Information