Subband CODEC (SBC)

Application Programming Interface Reference Manual

Profile Version: 1.2

Release: 2.1.3

January 19, 2011

Louisville, KY www.stonestreetone.com

Table of Contents

1. Introduction

Bluetopia®, the Bluetooth Protocol Stack by Stonestreet One, provides a software architecture that encapsulates the upper functionality of the Bluetooth Protocol Stack. More specifically, this stack is a software solution that resides above the Physical HCI (Host Controller Interface) Transport Layer and extends through the L2CAP (Logical Link Control and Adaptation Protocol) and the SCO (Synchronous Connection-Oriented) Link layers. In addition to basic functionality at these layers, the Bluetooth Protocol Stack by Stonestreet One provides implementations of the Service Discovery Protocol (SDP), RFCOMM (the Radio Frequency serial COMMunications port emulator), and several of the Bluetooth Profiles. Program access to these layers, services, and profiles is handled via Application Programming Interface (API) calls.

This document focuses on the API reference that contains a description of all programming interfaces required to utilize the Subband CODEC library provided by Bluetopia. Chapter 2 contains a description of the programming interfaces for this library. And, Chapter 3 contains the header file name list for the Subband CODEC library.

1.1 Scope

This reference manual provides information on the Subband CODEC library API. Utilizing this library, in conjunction with the Audio profiles provided by Bluetopia®, the application programmer will be able to fully utilize the functionality of the Subband CODEC library. It should be noted, however, that the Subband CODEC library has no dependence on the Bluetooth Protocol Stack and/or any of its components (profiles). This means that the Subband CODEC library can also be utilized without any other component of the Bluetopia®, libraries, source files, etc. This API is available on the full range of platforms supported by Stonestreet One:

- Windows
- Windows Mobile
- Windows CE
- Linux
- QNX
- Other Embedded OS

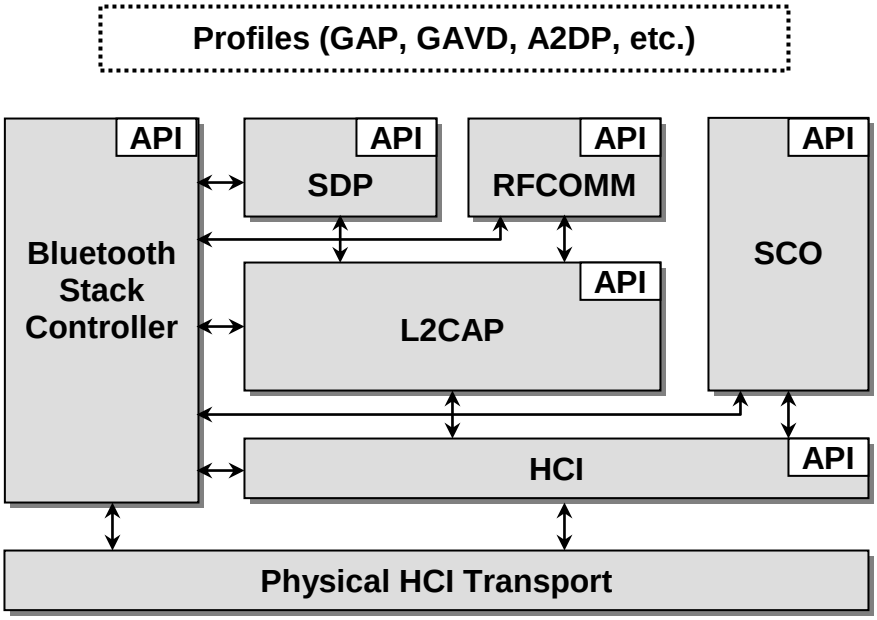

Figure 1-1 The Stonestreet One Bluetooth Protocol Stack

1.2 Applicable Documents

The following documents may be used for additional background and technical depth regarding the Bluetooth technology.

1. *Specification of the Bluetooth System, Volume 1, Core*, version 1.1, February 22, 2001.
2. *Specification of the Bluetooth System, Volume 2, Profiles*, version 1.1, February 22, 2001.

3. *Specification of the Bluetooth System, Volume 1, Architecture and Terminology Overview*, version 2.0 + EDR, November 4, 2004.
4. *Specification of the Bluetooth System, Volume 2, Core System Package*, version 2.0 + EDR, November 4, 2004.
5. *Specification of the Bluetooth System, Volume 3, Core System Package*, version 2.0 + EDR, November 4, 2004.
6. *Bluetooth Assigned Numbers*, version 1.1, February 22, 2001.
7. *Audio/Video Distribution Transport Protocol Specification*, version 1.0, May 22, 2003.
8. *Generic Audio/Video Distribution Profile*, version 1.0, May 22, 2002.
9. *Advanced Audio Distribution Profile*, version 1.0, June 24, 2002.
10. *Bluetopia® Protocol Stack, Application Programming Interface Reference Manual,* version 2.1.3, January 19, 2011.

Possible error returns are listed for each API function call. These are the *most likely* errors, but in fact programmers should allow for the possibility of any error listed in the BTerrors.h header file to occur as the value of a function return.

1.3 Acronyms and Abbreviations

Acronyms and abbreviations used in this document and other Bluetooth specifications are listed in the table below.

Term	Meaning
A2DP	Advance Audio Distribution Profile (Bluetooth Profile)
API	Application Programming Interface
BD_ADDR	Bluetooth Device Address
BT	Bluetooth
GAP	Generic Access Profile (Bluetooth Profile)
GAVD	Generic Audio/Video Distribution (Bluetooth Profile)
LSB	Least Significant Bit
MSB	Most Significant Bit
PCM	Pulse Code Modulation
SBC	Subband CODEC
SDP	Service Discovery Protocol (Bluetooth Protocol)
SPP	Serial Port Protocol (Bluetooth Profile)
UART	Universal Asynchronous Receiver/Transmitter

Term	Meaning
USB	Universal Serial Bus

2. Subband CODEC Programming Interface

The Subband CODEC programming interface defines the library functions that are available to the programmer to encode and decode Subband CODEC streams. The actual prototypes and constants outlined in this section can be found in the **SBCAPI.H** header file in the Bluetopia distribution.

2.1 Subband CODEC

The available Subband CODEC command functions are listed in the table below and are described in the text that follows.

Function	Description
SBC_Initialize_Encoder	Initialize SBC encoder library.
SBC_Cleanup_Encoder	Free resources allocated by the SBC encoder library.
SBC_Change_Encoder_Configuration	Change parameters used to encode SBC stream.
SBC_Encode_Data	Encode PCM audio data into SBC frames.
SBC_Initialize_Decoder	Initialize SBC decoder library.
SBC_Cleanup_Decoder	Free resources allocated to SBC decoder library.
SBC_Decode_Data	Decode SBC bit streams into PCM audio data.
SBC_CalculateEncoderBitPoolSize	Determine bit pool size based on encoder configuration
SBC_CalculateEncoderFrameLength	Determine frame length (in bytes) based on encoder configuration
SBC_CalculateEncoderBitRate	Determine bit rate based on encoder configuration

SBC_Initialize_Encoder

The following function is responsible for initializing an SBC Encoder stream. This function should be called once for every concurrent audio stream (PCM data) that will be encoded as a SBC stream. A successful call to this function will return a NON-NULL Encoder Handle which can then be used as the first argument to the remaining Encoder functions. The parameter to this function specifies the configuration parameters that are used to encode the bit stream. This configuration can be changed via the SBC_Change_Encoder_Configuration() function if required.

It should be noted that the number of bytes required to hold a single SBC encoded frame will be equal to:

Block Size * Number Subbands

The SBC_CalculateEncoderFrameLength() function can also be used to calculate the size (in bytes) required to hold a single SBC frame with specified encoder configurations.

Prototype:

Encoder_t **SBC_Initialize_Encoder**(SBC_Encode_Configuration_t *EncodeConfiguration)

Parameters:

EncodeConfiguration — Pointer to the configuration that will be used for all subsequent bit stream encoding (unless explicitly changed). The encoding parameters should remain constant during the encoding of a single frame of SBC data. If the configuration is changed during the encoding of a frame, the data will be flushed and the encoding of a new frame will begin. This is defined by the following structure:

```
typedef struct
{
  SBC_Sampling_Frequency_t  SamplingFrequency;
  SBC_Block_Size_t          BlockSize;
  SBC_Channel_Mode_t        ChannelMode;
  SBC_Allocation_Method_t   AllocationMethod;
  SBC_Subbands_t            Subbands;
  unsigned long             MaximumBitRate;
} SBC_Encode_Configuration_t;
```

where,

SamplingFrequency defines the sampling frequency of the input audio data using the following enumeration:

```
typedef enum
{
  sf16kHz,
  sf32kHz,
  sf441kHz,
  sf48kHz
} SBC_Sampling_Frequency_t;
```

BlockSize defines the block size to use when encoding the input data using the following enumeration:

```
typedef enum
{
  bsFour,
  bsEight,
  bsTwelve,
  bsSixteen
} SBC_Block_Size_t;
```

ChannelMode defines the encoding mode that should be used when encoding the input data (and also whether the

input data is Mono or Stereo) using the following enumeration:

```
typedef enum
{
  cmMono,
  cmDualChannel,
  cmStereo,
  cmJointStereo
} SBC_Channel_Mode_t;
```

AllocationMethod defines the encoding allocation to use when encoding the input data using the following enumeration:

```
typedef enum
{
  amLoudness,
  amSNR
} SBC_Allocation_Method_t;
```

Subbands defines the number of subbands to use when encoding the data using the following enumeration:

```
typedef enum
{
  sbFour,
  sbEight
} SBC_Subbands_t;
```

MaximumBitRate defines the maximum bit rate to use when encoding the data (bit pool will automatically be calculated so that the resulting SBC bit rate will be less than or equal to this value).

Return:

Valid encoder handle if successful (non-NULL).

Null if an error occurs.

SBC_Cleanup_Encoder

The following function is responsible for freeing all resources that were previously allocated for a SBC Encoder stream.

Prototype:

void **SBC_Cleanup_Encoder**(Encoder_t EncoderHandle)

Parameters:

EncoderHandle	Encoder Handle of the Encoder to cleanup. This value was returned from a successful call to the SBC_Initialize_Encoder() function.

Return:

SBC_Change_Encoder_Configuration

The following function is responsible for changing the parameters that are used to encode a SBC Stream.

Prototype:

int **SBC_Change_Encoder_Configuration**(Encoder_t EncoderHandle,
SBC_Encode_Configuration_t *EncodeConfiguration)

Parameters:

EncoderHandle	Encoder Handle of the Encoder to cleanup. This value was returned from a successful call to the SBC_Initialize_Encoder() function.
EncodeConfiguration	Pointer to the configuration that will be used for all subsequent bit stream encoding (unless explicitly changed). The encoding parameters should remain constant during the encoding of a single frame of SBC data. See EncodeConfiguration parameter description of SBC_Initialize_Encoder for a detailed structure definition.

Return:

Positive, non zero value, if successful that represents the required size of a single SBC Frame (equivalent to the return value from the SBC_CalculateEncoderFrameLength() function).

Negative value error code if an error occurs.

SBC_Encode_Data

The following function is responsible for encoding Audio Data (PCM) into SBC Frames. The parameters that are used to encode the specified data are based on the encoder configuration values that were chosen with either the SBC_Initialize_Encoder() function or the SBC_Change_Encoder_Configuration() function. The encoded SBC frame is stored in the EncodedBitStreamData parameter which (on input) specifies the buffer (and size of the buffer) to store the encoded bit stream into. The SBC_CalculateEncoderFrameLength() utility function can be used to determine the minimum size of the buffer that is required to be passed to this function. It is very important to note that the caller is responsible for correctly specifying the bit stream buffer information so that this function can encode the bit stream.

Prototype:

int **SBC_Encode_Data**(Encoder_t EncoderHandle,
SBC_Encode_Data_t *EncodeData,
SBC_Encode_Bit_Stream_Data_t *EncodedBitStreamData)

Parameters:

EncoderHandle — Encoder Handle of the Encoder stream that is to encode the specified data. This value was returned from a successful call to the SBC_Initialize_Encoder() function.

EncodeData — Pointer to an Encode Data structure, which holds the data to be encoded. The Right Channel Data is not required in the case when encoding data with the Channel Mode in the Encode Configuration Structure set to Mono. This is defined by the following structure:

```
typedef struct
{
  unsigned short     *LeftChannelDataPtr;
  unsigned short     *RightChannelDataPtr;
  unsigned int        ChannelDataLength;
  unsigned int        UnusedChannelDataLength;
} SBC_Encode_Data_t;
```

where,

LeftChannelDataPtr is a pointer to the left audio channel input (PCM) data for Stereo input. When Mono input is used this parameter points to the audio channel input (PCM) data.

RightChannelDataPtr is a pointer to the right audio channel input (PCM) data for Stereo input. When Mono input is used this parameter is ignored.

ChannelDataLength specifies the size (in bytes) of the input data. Note that for Stereo input this specifies the size (in bytes) of each individual channel (which **MUST** be the same size) **NOT** the total size of both channels.

UnusedChannelDataLength is a value that should be set to the same size as ChannelDataLength member when this function is called with data the first time. This member is changed to reflect the number of bytes from the input buffer that have been consumed when this function call is successful. The caller should repeatedly call this function until this member is returned to be zero (or an error occurs).

EncodedBitStreamData — Pointer to the encoded SBC Frame buffer information. On input the size and data pointer member must be valid. If the return value of this function indicates that a SBC Audio Frame was processed then the contents of the data stream buffer (and length member) will specify the encoded SBC bit stream data, otherwise the members of this structure will be undefined. This is defined by the following structure:

```
typedef struct
{
```

```
   unsigned int         BitStreamDataSize;
   unsigned int         BitStreamDataLength;
   unsigned char       *BitStreamDataPtr;
} SBC_Encode_Bit_Stream_Data_t;
```

where,

BitStreamDataSize specifies the size (in bytes) of the actual SBC Encoded Bit Stream Data that is pointed to by the BitStreamDataPtr member. This member is required to be valid on input and *MUST* specify the size (in bytes) of the buffer pointed to by the BitStreamDataPtr member.

BitStreamDataLength specifies the size (in bytes) of the actual SBC Encoded Bit Stream Data that is pointed to by the BitStreamDataPtr member. This member will contain a non-zero value if the return value from this function indicates that a SBC frame has been encoded.

BitStreamDataPtr is a pointer to the actual SBC Encoded Bit Stream Data buffer. This member must point to a buffer of (at least) the size specified by the BitStreamDataSize member.

Return:

SBC_PROCESSING_COMPLETE when a complete frame of SBC data has been encoded. When this value is returned the EncodedBitStreamData structure will be filled with the correct SBC Encoded Bit stream information and the BitStreamDataLength member will contain the length (in bytes) of the data that is present in the bit stream data buffer.

SBC_PROCESSING_DATA if more audio data is required before a complete frame can be produced. When this value is returned, the EncodedBitStreamData structure contents returned from this function will be undefined.

Negative value error code if an error occurs.

SBC_Initialize_Decoder

The following function is responsible for initializing an SBC Decoder stream. This function should be called once for every concurrent SBC Encoded stream that will be decoded as Audio Data (PCM). A successful call to this function will return a NON-NULL Decoder Handle which can then be used as the first argument to the remaining Decoder functions.

Prototype:

Decoder_t **SBC_Initialize_Decoder**(void)

Parameters:

Return:

Valid decoder handle if successful (non-NULL).

Null if an error occurs.

SBC_Cleanup_Decoder

The following function is responsible for freeing all resources that were previously allocated for a SBC Decoder stream.

Prototype:

void **SBC_Cleanup_Decoder**(Decoder_t DecoderHandle)

Parameters:

DecoderHandle	Decoder Handle of the Encoder to cleanup. This value was returned from a successful call to the SBC_Initialize_Decoder() function.

Return:

SBC_Decode_Data

The following function is responsible for decoding SBC encoded Bit Streams into Audio Data (PCM). Upon the completion of processing a frame of audio data, the Decode Configuration parameters and Decoded Data parameters will be set with the information and data decoded from the audio frame.

Prototype:

int **SBC_Decode_Data**(Decoder_t DecoderHandle, unsigned int DataLength, unsigned char *DataPtr, SBC_Decode_Configuration_t *DecodeConfiguration, SBC_Decode_Data_t *DecodedData, unsigned int *UnusedDataLength)

Parameters:

DecoderHandle	Decoder Handle of the Decoder stream that is to decode the specified data. This value was returned from a successful call to the SBC_Initialize_Decoder() function.
DataLength	Bit stream data buffer length specified in bytes.
DataPtr	Pointer to actual bit stream to be decoded. This buffer must be (at least) the size specified by the DataLength parameter.
DecodeConfiguration	Pointer to decode configuration structure. The contents of this structure are valid when this function indicates that it has decoded a SBC frame (SBC_PROCESSING_COMPLETE is returned). The information in this structure specifies the decoded information about the PCM data that was returned. This is defined by the following structure: typedef struct

```
{
  SBC_Sampling_Frequency_t  SamplingFrequency;
  SBC_Block_Size_t          BlockSize;
  SBC_Channel_Mode_t        ChannelMode;
  SBC_Allocation_Method_t   AllocationMethod;
  SBC_Subbands_t            Subbands;
  unsigned long             BitRate;
  unsigned int              BitPool;
  unsigned int              FrameLength;
} SBC_Decode_Configuration_t;
```

where,

SamplingFrequency specifies the sampling frequency of the output audio data using the following enumeration:

```
typedef enum
{
  sf16kHz,
  sf32kHz,
  sf441kHz,
  sf48kHz
} SBC_Sampling_Frequency_t;
```

BlockSize specifies the block size that was used when the data was encode which will be based on the following enumeration:

```
typedef enum
{
  bsFour,
  bsEight,
  bsTwelve,
  bsSixteen
} SBC_Block_Size_t;
```

ChannelMode specifies the encoding mode that was used when the data was encoded using the following enumeration:

```
typedef enum
{
  cmMono,
  cmDualChannel,
  cmStereo,
  cmJointStereo
} SBC_Channel_Mode_t;
```

AllocationMethod specifies the encoding allocation that was used when the input data was encoded using the following enumeration:

```
typedef enum
{
  amLoudness,
  amSNR
```

```
} SBC_Allocation_Method_t;
```

Subbands specifies the number of subbands that were used to encode the data using the following enumeration:

```
typedef enum
{
  sbFour,
  sbEight
} SBC_Subbands_t;
```

BitRate specifies the bit rate that was used to encode the current SBC stream.

BitPool specifies the bit pool that was used to encode the current SBC stream.

FrameLength specifies the size (in bytes) of the current SBC frame that was decoded.

DecodedData — Pointer to a Decode Data structure that will hold the data that is decoded (only when the return value of this function indicates that an SBC Frame was decoded). The Right Channel Data is not used in the case when SBC bit streams that specify Mono as the Channel Mode (specified in the Decode Configuration Structure). This is defined by the following structure:

```
typedef struct
{
  unsigned int        ChannelDataSize;
  unsigned int        LeftChannelDataLength;
  unsigned short     *LeftChannelDataPtr;
  unsigned int        RightChannelDataLength;
  unsigned short     *RightChannelDataPtr;
} SBC_Encode_Data_t;
```

where,

ChannelDataSize specifies the size (in unsigned shorts NOT bytes) of the left and right audio channel buffers. This value is required to be specified on input and specifies the number of samples that can be stored in the channel data buffers.

LeftChannelDataLength specifies the size (in unsigned shorts NOT bytes) of the decoded output data (PCM) for the left audio channel (or a Mono channel).

LeftChannelDataPtr is a pointer to the left audio channel output (PCM) data for Stereo output. When Mono output is used this parameter points to the audio channel output (PCM) data. Note that this buffer is required to be passed in to this function and must be (at least) the size specified by the ChannelDataSize member.

RightChannelDataLength specifies the size (in unsigned shorts NOT bytes) of the decoded output data (PCM). When Mono output is specified this parameter is ignored.

RightChannelDataPtr is a pointer to the right audio channel output (PCM) data for Stereo output. When Mono output is specified this parameter is ignored. Note that this buffer is required to be passed in to this function and must be (at least) the size specified by the ChannelDataSize member.

UnusedDataLength — Pointer to the amount of data that was passed to this function but was not required for the complete processing of the current frame of data. Any unused data should be passed back to the decoder in the next call of this function. This parameter is changed to reflect the number of bytes from the input buffer that have been consumed when this function call is successful. The caller should repeatedly call this function until this parameter is returned as zero (or an error occurs).

Return:

SBC_PROCESSING_COMPLETE when a complete frame of SBC data has been decoded. When this value is returned the DecodeConfiguration and DecodeData structures will be filled with the correct PCM decoded audio information.

SBC_PROCESSING_DATA if more bit stream data is required before the audio data can be completely decoded. When this value is returned the DecodeConfiguration and DecodeData structure contents are undefined.

Negative value error code if an error occurs.

SBC_CalculateEncoderBitPoolSize

The following function is a utility function that is responsible for determining the bit pool size (in number of bits) required to store an entire SBC encoded frame. This value is calculated based on the encoder configuration information that is passed to this function.

Prototype:

int **SBC_CalculateEncoderBitPoolSize**(
SBC_Encode_Configuration_t *EncodeConfiguration)

Parameters:

EncodeConfiguration — Pointer to the configuration that will be used to calculate the required bit pool size (in number of bits). This is defined by the following structure:

```
typedef struct
{
  SBC_Sampling_Frequency_t  SamplingFrequency;
  SBC_Block_Size_t          BlockSize;
```

```
   SBC_Channel_Mode_t        ChannelMode;
   SBC_Allocation_Method_t   AllocationMethod;
   SBC_Subbands_t            Subbands;
   unsigned long             MaximumBitRate;
} SBC_Encode_Configuration_t;
```

where,

SamplingFrequency defines the sampling frequency of the input audio data using the following enumeration:

```
typedef enum
{
  sf16kHz,
  sf32kHz,
  sf441kHz,
  sf48kHz
} SBC_Sampling_Frequency_t;
```

BlockSize defines the block size to use when encoding the input data using the following enumeration:

```
typedef enum
{
  bsFour,
  bsEight,
  bsTwelve,
  bsSixteen
} SBC_Block_Size_t;
```

ChannelMode defines the encoding mode that should be used when encoding the input data (and also whether the input data is Mono or Stereo) using the following enumeration:

```
typedef enum
{
  cmMono,
  cmDualChannel,
  cmStereo,
  cmJointStereo
} SBC_Channel_Mode_t;
```

AllocationMethod defines the encoding allocation to use when encoding the input data using the following enumeration:

```
typedef enum
{
  amLoudness,
  amSNR
} SBC_Allocation_Method_t;
```

Subbands defines the number of subbands to use when encoding the data using the following enumeration:

```
typedef enum
{
```

```
  sbFour,
  sbEight
} SBC_Subbands_t;
```

MaximumBitRate defines the maximum bit rate to use when encoding the data (bit pool will automatically be calculated so that the resulting SBC bit rate will be less than or equal to this value).

Return:

Positive value which represents the size of the calculated bit pool value (in number of bits)

Negative value error code if an error occurs.

SBC_CalculateEncoderFrameLength

The following function is a utility function that is responsible for determining the number of bytes required to store an entire SBC encoded frame. This value is calculated based on the encoder configuration information that is passed to this function.

Prototype:

int **SBC_CalculateEncoderFrameLength** (
SBC_Encode_Configuration_t *EncodeConfiguration)

Parameters:

EncodeConfiguration — Pointer to the configuration that will be used to calculate the required encoded frame size (in number of bytes). This is defined by the following structure:

```
typedef struct
{
  SBC_Sampling_Frequency_t  SamplingFrequency;
  SBC_Block_Size_t          BlockSize;
  SBC_Channel_Mode_t        ChannelMode;
  SBC_Allocation_Method_t   AllocationMethod;
  SBC_Subbands_t            Subbands;
  unsigned long             MaximumBitRate;
} SBC_Encode_Configuration_t;
```

where,

SamplingFrequency defines the sampling frequency of the input audio data using the following enumeration:

```
typedef enum
{
  sf16kHz,
  sf32kHz,
  sf441kHz,
  sf48kHz
} SBC_Sampling_Frequency_t;
```

BlockSize defines the block size to use when encoding the input data using the following enumeration:

```
typedef enum
{
  bsFour,
  bsEight,
  bsTwelve,
  bsSixteen
} SBC_Block_Size_t;
```

ChannelMode defines the encoding mode that should be used when encoding the input data (and also whether the input data is Mono or Stereo) using the following enumeration:

```
typedef enum
{
  cmMono,
  cmDualChannel,
  cmStereo,
  cmJointStereo
} SBC_Channel_Mode_t;
```

AllocationMethod defines the encoding allocation to use when encoding the input data using the following enumeration:

```
typedef enum
{
  amLoudness,
  amSNR
} SBC_Allocation_Method_t;
```

Subbands defines the number of subbands to use when encoding the data using the following enumeration:

```
typedef enum
{
  sbFour,
  sbEight
} SBC_Subbands_t;
```

MaximumBitRate defines the maximum bit rate to use when encoding the data (bit pool will automatically be calculated so that the resulting SBC bit rate will be less than or equal to this value).

Return:

Positive value which represents the size of the calculated encoded frame size value (in number of bytes)

Negative value error code if an error occurs.

SBC_CalculateEncoderBitRate

The following function is a utility function that is responsible for determining the actual bit rate on an encoded SBC frame. This value is calculated based on the encoder configuration information that is passed to this function. This function is useful to determine the actual bit rate rather than the requested maximum bit rate. The actual bit rate can vary from the requested maximum bit rate due to the framing/encoding process.

Prototype:

long **SBC_CalculateEncoderBitRate** (
SBC_Encode_Configuration_t *EncodeConfiguration)

Parameters:

EncodeConfiguration — Pointer to the configuration that will be used to calculate the actual encoded bit rate (in number of bits). This is defined by the following structure:

```
typedef struct
{
  SBC_Sampling_Frequency_t  SamplingFrequency;
  SBC_Block_Size_t          BlockSize;
  SBC_Channel_Mode_t        ChannelMode;
  SBC_Allocation_Method_t   AllocationMethod;
  SBC_Subbands_t            Subbands;
  unsigned long             MaximumBitRate;
} SBC_Encode_Configuration_t;
```

where,

SamplingFrequency defines the sampling frequency of the input audio data using the following enumeration:

```
typedef enum
{
  sf16kHz,
  sf32kHz,
  sf441kHz,
  sf48kHz
} SBC_Sampling_Frequency_t;
```

BlockSize defines the block size to use when encoding the input data using the following enumeration:

```
typedef enum
{
  bsFour,
  bsEight,
  bsTwelve,
  bsSixteen
} SBC_Block_Size_t;
```

ChannelMode defines the encoding mode that should be used when encoding the input data (and also whether the

input data is Mono or Stereo) using the following enumeration:

```
typedef enum
{
  cmMono,
  cmDualChannel,
  cmStereo,
  cmJointStereo
} SBC_Channel_Mode_t;
```

AllocationMethod defines the encoding allocation to use when encoding the input data using the following enumeration:

```
typedef enum
{
  amLoudness,
  amSNR
} SBC_Allocation_Method_t;
```

Subbands defines the number of subbands to use when encoding the data using the following enumeration:

```
typedef enum
{
  sbFour,
  sbEight
} SBC_Subbands_t;
```

MaximumBitRate defines the maximum bit rate to use when encoding the data (bit pool will automatically be calculated so that the resulting SBC bit rate will be less than or equal to this value).

Return:

Positive value which represents the actual bit rate of the encoded stream (in number of bits)

Negative value error code if an error occurs.

3. File Distributions

The header files that are distributed with the Subband CODEC Library are listed in the table below.

File	Contents/Description
SBCAPI.h	Stonestreet One Subband CODEC API definitions
SS1SBC.h	Stonestreet One Subband CODEC include file.