SILK SDK API

Instructions for using the Skype SILK SDK API

The Application Programming Interface (API) is defined in four header files located in the `interface/` folder:

- `SKP_Silk_SDK_API.h` – function declarations for the SILK encoder and decoder.
- `SKP_Silk_control.h` – declarations of structures for controlling the encoder and for controlling and getting status information from the decoder
- `SKP_API_typedef.h` – type definitions
- `SKP_Silk_errors.h` – error code descriptions for the SILK SDK

Encoder Control Struct Description (SKP_Silk_control.h)

The encoder structure (`SKP_SILK_SDK_EncControlStruct`) has the following members:

`SKP_int32 sampleRate;`

(I) Sampling frequency in Hertz of the encoder. Valid values are: 8000, 12000, 16000 and 24000.

`SKP_int packetSize;`

(I) Number of samples per packet. A number of samples corresponding to 20, 40, 60, 80 or 100 ms are supported at any of the above listed sampling frequencies, e.g., 480 samples for a 20 ms packet at 24000 Hz sample frequency.

`SKP_int32 bitRate;`

(I) Target bitrate for active speech in the range 5000 – 100000 bits per second (bps). The value is limited internally if the input value is not within the supported range.

`SKP_int packetLossPercentage;`

(I) Estimated packet loss percentage in the uplink direction (0 – 100). If in-band forward error correction is used, this information determines how much protection the encoder will add.

`SKP_int complexity;`

(I) Complexity setting for choosing one out of the 3 complexity modes. Supported values are 0, 1 and 2, where 0 is the lowest and 2 is the highest complexity.

`SKP_int useInBandFEC;`

(I) Enables / disables use of in-band forward error correction (0 disables and 1 enables).

`SKP_int useDTX;`

(I) Enables / disables use of discontinuous transmission (0 disables and 1 enables).

Decoder Control Struct Description (SKP_Silk_control.h)

The decoder structure (`SKP_SILK_SDK_DecControlStruct`) has the following members:

`SKP_int32 sampleRate;`

(I) Sampling frequency in Hertz of the decoder output signal. Should be at least the input sampling frequency that is used by the encoder. Valid values are 8000, 12000, 16000 and 24000.

`SKP_int frameSize;`

(O) Frames size in samples; always corresponding of 20 ms of data: 160, 240, 320 or 480, depending on `sampleRate`.

`SKP_int framesPerPacket;`

(O) Number of 20 ms frames in the last decoded packet. Possible output values are 1, 2, 3, 4 or 5.

`SKP_int moreInternalDecoderFrames;`

(O) Flag which when set specifies that more frames can be obtained from a multi-frame payload that has been buffered in the decoder (0 or 1).

`SKP_int inBandFECOffset;`

(O) Distance between main payload and redundant payload in packets (0, 1 or 2)

Comments:
The SILK decoder will always decode a 20 ms frame for each function call. If the received packet contains more than one frame, the decoder must be called more than once to fully decode that packet. To indicate when the decoding of the packet has finished the `moreInternalDecoderFrames` flag is used. If `moreInternalDecoderFrames` is 0, the last packet has been fully decoded and the decoder is ready for the next packet. When `moreInternalDecoderFrames` is 1, the decoder is not finished decoding all the frames that were contained in the last packet, and the decoder should be called until `moreInternalDecoderFrames` changes to 0. Also, when `moreInternalDecoderFrames` is 1, the input `inData` to

`SKP_Silk_SDK_Decode` is ignored, as in this case the remaining part of the last received packet is read from an internal buffer.

Functions (SKP_Silk_SDK_API.h)

All functions return an error code, which is 0 if no error was encountered during function execution. A negative value is returned to indicate an error. The list of error codes can be found in `SKP_Silk_errors.h`.

```
SKP_int SKP_Silk_SDK_Get_Encoder_Size(
     SKP_int32 *encSizeBytes
);
```

`*encSizeBytes:`	(O) Size of encoder state in bytes.

Description:
Writes the size of the SILK encoder state in number of bytes to `*encSizeBytes.` Use this function to allocate the right amount of memory space for the encoder state.

```
SKP_int SKP_Silk_SDK_InitEncoder(
     void                             *encState,
     SKP_SILK_SDK_EncControlStruct *encStatus
);
```

`*encState:`	(I/O) Encoder state.
`*encStatus:`	(O) Encoder status struct. Returns default encoder settings.

Description:
Initializes the encoder state, `encState,` and returns the default encoder status, `encStatus`. This function has to be called before the first call to the encoder, and may be called to reset the internal encoder state, for instance when initiating a new voice call.

```
SKP_int SKP_Silk_SDK_QueryEncoder(
     const void                       *encState,
     SKP_SILK_SDK_EncControlStruct *encStatus
);
```

`*encState:`	(I) Encoder state.
`*encStatus:`	(O) Encoder status struct. Returns the current encoder settings.

Description:
Returns the current encoder settings, `encStatus`. This function may be called to check the settings of the encoder.

```
SKP_int SKP_Silk_SDK_Encode(
     void                                    *encState,
     const SKP_SILK_SDK_EncControlStruct     *encControl,
     const SKP_int16                         *samplesIn,
     SKP_int                                 nSamplesIn,
     SKP_uint8                               *outData,
     SKP_int16                               *nBytesOut
);
```

`*encState:`	(I/O) Encoder state.
`*encControl:`	(I) Structure to hold encoder control.
`*samplesIn:`	(I) Input vector with `nSamplesIn` of audio samples.
`nSamplesIn:`	(I) Number of input samples. Must correspond to a multiple of 10 ms, and be no higher than `encControl->packetSize.`
`*outData:`	(O) Output payload.
`*nBytesOut:`	(I/O) Input: Maximum number of bytes allowed in payload. Output: Number of output bytes in the payload.

Description:
This is the main encoder function. The `nSamplesIn` input samples are read from `samplesIn` and buffered internally until at least `encControl->packetSize` samples are available, at which point one payload is encoded and written to `outData`. The parameter pointer `nBytesOut` serves as both input and output. As input `nBytesOut` specifies the maximum number of bytes the payload may consist of; this is (at most) the number of bytes your application has allocated in the `outData` array. As output `nBytesOut` returns the actual size of the payload in `outData`. All members of the `encControl` structure must be set to valid values (see description of `SKP_SILK_SDK_EncControlStruct`), otherwise errors are reported. The input vector `samplesIn` must be sampled at the rate stored in `encControl->sampleRate.`

```
SKP_int SKP_Silk_SDK_Get_Decoder_Size(
     SKP_int32 *decSizeBytes
);
```

`decSizeBytes:`	(O) Size in bytes of the decoder state.

Description:
Writes the size of the SILK decoder state in number of bytes to `decSizeBytes`. Use this function to allocate the right amount memory space for the decoder state.

```
SKP_int SKP_Silk_SDK_InitDecoder(
     void *decState
);
```

`*decState:`	(I/O) Encoder state.

Description:
Initializes the decoder state, `decState`. This function has to be called before first call to the decoder and may be called to reset the internal decoder state, for instance when initiating a new voice call.

```
SKP_int SKP_Silk_SDK_Decode(
     void*                              decState,
     SKP_SILK_SDK_DecControlStruct*     decStatus,
     SKP_int                            lostFlag,
     const SKP_uint8                    *inData,
     const SKP_int                      nBytesIn,
     SKP_int16                          *samplesOut,
     SKP_int16                          *nSamplesOut
);
```

`*decState:`	(I/O) Decoder state.
`*decStatus:`	(I/O) Decoder status struct.
`lostFlag:`	(I) Flag to activate packet loss concealment. 0 not lost 1 lost.
`*inData:`	(I) Packet payload to be decoded.
`nBytesIn:`	(I) Number of bytes in the payload (`inData`).
`*samplesOut:`	(O) Decoded samples.
`*nSamplesOut:`	(O) Number of decoded samples.

Description:
This is the main decoder function. When a payload was lost during transport, the input `lostFlag` should be set to 1, otherwise set it to 0. The `nBytesIn` input parameter must exactly match the number of bytes in the payload as it is used to detect corrupted packets. The sampling rate (in Hertz) of the output signal is set with `decStatus->sampleRate.` It must be ensured that the decoder sampling rate matches or exceeds the encoding sample rate. This can be achieved either by setting the decoder sampling rate to the highest value of 24000 or by indicating a lower maximum sampling rate to the farend sender during call setup.

The decoder updates the decoder status `decStatus`, which should be used to handle multiple frame packets. See description of the decoder status struct.

```
void SKP_Silk_SDK_search_for_LBRR(
     void                    *decState,
     const SKP_uint8         *inData,
     const SKP_int16         nBytesIn,
     SKP_int                 lost_offset,
     SKP_uint8               *LBRRData,
     SKP_int16               *nLBRRBytes
);
```

`*decState:`	(I/O) Decoder state.
`*inData:`	(I) Future packet payload.
`nBytesIn:`	(I) Number of bytes in packet payload.
`lost_offset:`	(I) Distance in packets between lost packet and future packet (1 or 2).
`*LBRRData:`	(O) Extracted redundant LBRR payload.
`*nLBRRBytes:`	(O) Number of bytes of LBRR payload.

Description:
Extracts low-bitrate redundant (LBRR) data. If a packet is lost during transmission and future packets are available on the decoder side this function can be used to extract any available in-band error correction data. If packet n is lost and packet n + 1, and / or packet n + 2 are available on the decoder side, the following should be done: First the payload from packet n + 1 should be input to `SKP_Silk_SDK_search_for_LBRR` together with a `lost_offset` value of 1 which is the relative distance to the lost packet, that is, (n + 1) – n. If after the call `*nLBRRBytes` is larger than zero it means that LBRR data was present in the packet, and the decoder should be called with `LBRRData` as input payload (`inData`) and `nLBRRBytes` as the payload length (`nBytesIn`). If `nLBRRBytes` is zero, the packet n + 2 should be searched, but this time the `lost_offset` should be set to 2, that is, (n+2) – n. If `nLBRRBytes` is zero also for packet n + 2 the decoder should be called with the `lostFlag` set to 1 to activate normal packet loss concealment.

```
void SKP_Silk_SDK_get_TOC(
     void                    *decState,
     const SKP_uint8         *inData,
     const SKP_int16         nBytesIn,
     SKP_Silk_TOC_struct *Silk_TOC
);
```

`*decState:`	(I) Decoder state.
`*inData:`	(I) Packet payload.
`nBytesIn:`	(I) Number of bytes in packet.
`*Silk_TOC:`	(O) Extracted table of contents information about the packet.

Description:
Returns a table of contents (TOC) structure for the packet.

```
SKP_INLINE const char *SKP_Silk_SDK_get_version();
```

Description:
Returns a string containing the Silk SDK version number.

Reference Test Program

The SDK contains source code with implementations of test programs for the SILK encoder (`test/Encoder.c`) and decoder (`test/Decoder.c`). This code serves as a reference implementation of how the API can be used and provides a quick way to compile, run and analyze the performance of SILK in its various modes at any bit-rate.

To compile and generate the test programs for the SDK on Mac or Linux the provided Makefile can be used. The targets for generating the encoder and decoder test programs respectively are 'Encoder' and 'Decoder'. Similarly on Windows the test programs can be generated through the provided Visual Studio 2005 solution and projects. Running either executable without command line arguments prints the command line options.