

# **Cyberon DSpotter SDK**

## **v2.2.x**

### **(32-bit IC Edition)**

#### **Programming Guide**

Version: 1.1.5

Date of issue: July 16, 2020



Leading Speech Solution provider

<http://www.cyberon.com.tw/>

---

No part may be reproduced except as authorized by written permission.  
The copyright and the foregoing restriction extend to reproduction in all media.

Cyberon Corporation, © 2019.

All rights reserved.

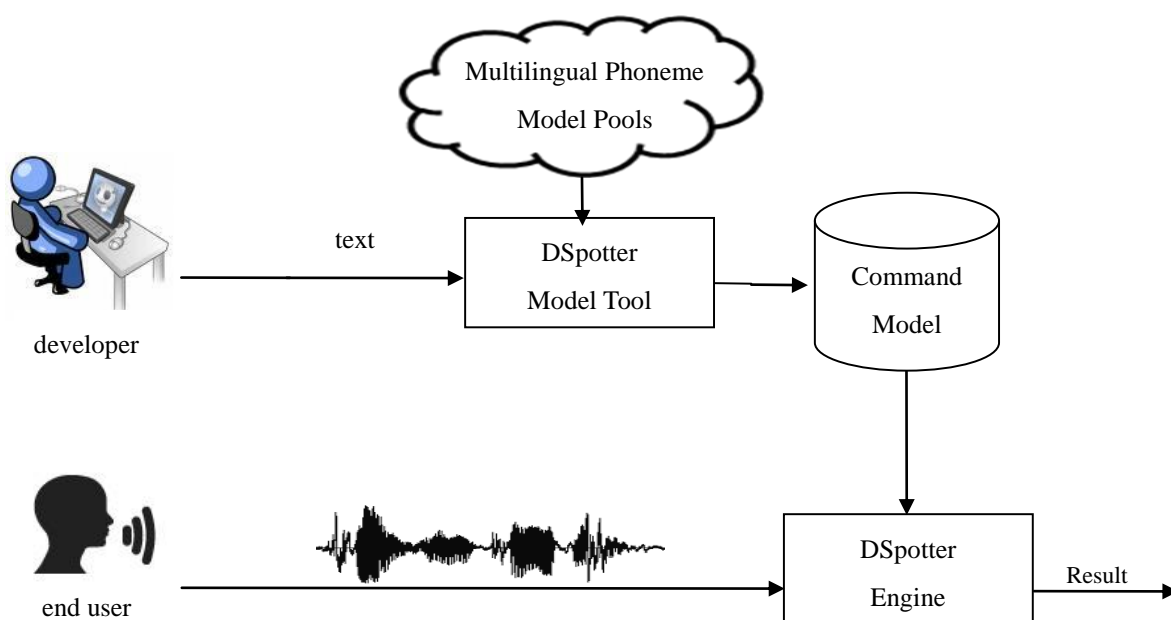
## Table of Contents

<b>1. About Cyberon DSpotter SDK.....</b>	<b>1</b>
<b>2. Release History .....</b>	<b>2</b>
<b>3. DSpotter Specifications, Related Files and Tools .....</b>	<b>4</b>
3.1. Specifications .....	4
3.2. Related Files and Tools .....	5
<b>4. DSpotter SDK API Standard Version.....</b>	<b>7</b>
4.1. Calling Flow Chart of Standard API .....	7
4.1. Initialize, Reset, and Release.....	8
DSpotter_Init_Multi.....	8
DSpotter_Reset .....	9
DSpotter_Release.....	9
DSpotter_GetMemoryUsage_Multi.....	9
4.2. Recognition .....	10
DSpotter_AddSample .....	11
DSpotter_GetResult .....	11
DSpotter_GetResultEPD.....	12
DSpotter_GetResultScore .....	12
DSpotter_GetCmdEnergy .....	13
DSpotter_GetNumWord.....	13
DSpotter_SetEndSil .....	14
DSpotter_SetCmdEndSil .....	15
DSpotter_GetCmdEndSil.....	15
DSpotter_SetConfReward .....	16
DSpotter_GetConfReward.....	16
DSpotter_SetCmdConfReward.....	17
DSpotter_GetCmdConfReward .....	17
DSpotter_SetSGDiffReward.....	18
DSpotter_GetSGDiffReward .....	18
DSpotter_SetEnergyTH .....	19
DSpotter_GetEnergyTH.....	19
DSpotter_SetResultMapID_Sep .....	20
DSpotter_SetResultMapID_Multi .....	20
DSpotter_GetResultMapID.....	21
<b>5. DSpotter SDK API Advanced Version.....</b>	<b>22</b>
5.1. Calling Flow Chart of Advanced API.....	22
5.2. Initialize and Release.....	23
DSpotterSD_Init .....	23
DSpotterSD_GetMemoryUsage .....	24

DSpotterSD_Release.....	24
5.3. Training .....	25
DSpotterSD_AddUtrrStart .....	26
DSpotterSD_AddSample .....	27
DSpotterSD_AddUtrrEnd .....	27
DSpotterSD_GetUtrrEPD .....	28
DSpotterSD_SetEpdLevel .....	28
DSpotterSD_TrainWord.....	29
DSpotterSD_DeleteWord.....	30
DSpotterSD_SetBackgroundEnergyThreshd.....	31
5.4. User-Implemented Flash Operation Functions.....	32
DataFlash_Write .....	32
DataFlash_Erase .....	32
<b>6. DSpotter SDK Error Code Table .....</b>	<b>33</b>
<b>7. DSpotter Supported Languages .....</b>	<b>34</b>

### 1. About Cyberon DSpotter SDK

**DSpotter SDK** is Cyberon's flagship high-performance embedded voice recognition solution specially optimized for mobile phones, automotives, smart home devices, consumer products, and interactive toys. Based on phoneme acoustic models, it enables developers to create applications of speaker-independent (SI) voice recognition capability without requiring costly data collection process for specific commands. With Win32-based DSpotter Model Tool, developers can easily and quickly create their own voice command models simply by text input. Other important features include always-on keyword-spotting capability, highly noise immune, adjustable sensitivity, voice quality assessment, and more than 30 commonly used language versions available.



## 2. Release History

Date	Version	Author	Description
2019/04/11	1.0.0	Roger	<b>Purpose:</b> First release
2019/08/01	1.0.2	Roger	<b>Purpose:</b> Update API
2019/10/14	1.0.3	Roger	<b>Purpose:</b> Update Spec / API
2019/10/24	1.0.4	Roger	<b>Purpose:</b> Update Spec
2019/11/25	1.0.5	Roger	<b>Purpose:</b> Update Spec
2019/12/12	1.0.6	Roger	<b>Purpose:</b> Update Spec
2020/02/20	1.0.7	Roger	<b>Purpose:</b> Update Spec for v2.1.0
2020/03/05	1.0.8	Roger	<b>Purpose:</b> Update Error code table
2020/03/26	1.0.9	Roger	<b>Purpose:</b> Update Spec / API
2020/04/16	1.1.0	Roger	<b>Purpose:</b> Update Error code table/Support Languages
2020/04/29	1.1.1	Roger	<b>Purpose:</b> Update Bin format
2020/05/14	1.1.2	Roger	<b>Purpose:</b> Update Error code table
2020/05/19	1.1.3	Roger	<b>Purpose:</b> Update Spec for v2.2.4
2020/06/02	1.1.4	Roger	<b>Purpose:</b> Update API

2020/07/16	1.1.5	Roger	<b>Purpose:</b> Update Bin format / API
------------	-------	-------	--

### 3. DSpotter Specifications, Related Files and Tools

#### 3.1. Specifications

DSpotter algorithm is available for 32-bit IC platforms. The core engines can be ported to a variety of platforms with architectures. Here lists DSpotter specifications ported to some popular platforms. For 32-bit DSP32, the standard versions of DSpotter algorithms given  $n_c$ , the number voice commands, each of which is 4 syllables in average, the technical specification is listed in the following table:

Algorithm	DSP32
IC Architecture	32-bit, fixed-point ALU
Sample Rate	16kHz
Feature Dimension	23
Code size	26KB
Data size	Level 0: 100KB + 28B * $n_c$ Level 1: 165KB + 28B * $n_c$
RAM size	Level 0: 40KB + 116B * $n_c$ Level 1: 45KB + 116B * $n_c$
Ported Platforms	ARM M3, M4 Tensilica HiFi 3, HiFi Mini
DMIPS request	Level 0: 45MIPS Level 1: 60MIPS

For 32-bit DSP32A, the advanced version of DSpotter algorithm equipped with voice tag training function, given  $n_c$ , the number voice commands, the technical specification is listed below:

Algorithm	DSP32A
IC Architecture Requirement	32-bit with fixed-point ALU
Input Sample Rate	16kHz
Feature Dimension	23
Code size	34KB
Data size	Level 0: 100KB + 340B + 400B * $n_c$ Level 1: 165KB + 340B + 400B * $n_c$
RAM size	Level 0: 75KB + 116B * $n_c$ Level 1: 80KB + 116B * $n_c$
Ported Platforms	ARM M3, M4 Tensilica HiFi 3, HiFi Mini

Note that code size listed in this document is for DSpotter core engine only, and codes for recording, playback, voice compression, data communication, and application main function are not included. Code and RAM sizes listed in the tables of this document are estimated with DSpotter 2.2.4 version.

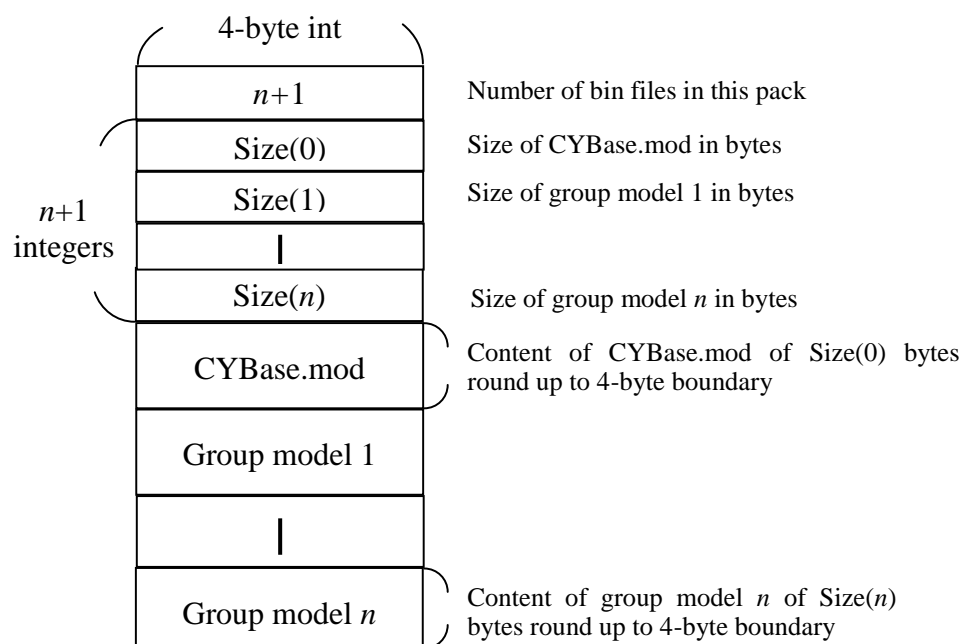
## 3.2. Related Files and Tools

### Library

- **DSpotterSDK\_16k23d\_XXXX.lib**, the library for DSpotter standard version, where XXXX is the name of the IC platform running the DSpotter engine, 16k and 23d stand for 16kHz sampling rate input and 23-dimensional feature vectors respectively.
- **DSpotterSDK\_16k23d\_XXXX\_A.lib**, the library for DSpotter advanced version.

### Data

- **CYBase.mod**: the DNN(Deep Neural Networks) model. The file name CYBase.mod is reserved for DSpotter Model Tool, and should never be changed.
- **XXXX.mod**: the command group model (or called command model). “Group\_ $n$ ” is the default name for the  $n$ -th group of commands in a project when created with DSpotter Model Tool, and can be renamed. All the command group models share the same CYBase.mod in a project.
- **XXXX\_MapID.bin**: the command ID mapping bin for “Group\_ $n$ ”.
- **CYTrimap.mod**: the phoneme map model. The file name CYTrimap.mod is reserved for DSpotter Model Tool, and should never be changed.
- **XXXX\_pack.bin**: the binary file that packs all command group models together with the shared CYBase.mod in a project, where XXXX is the project name assigned by developer when creating it with DSpotter Model Tool. Before using the models in the packed binary file, developers need to unpack it. For a DSpotter project of  $n$  group models, the packing format is shown in the diagram below. Note that this packing file is 4-byte aligned in Little-Endian manner.

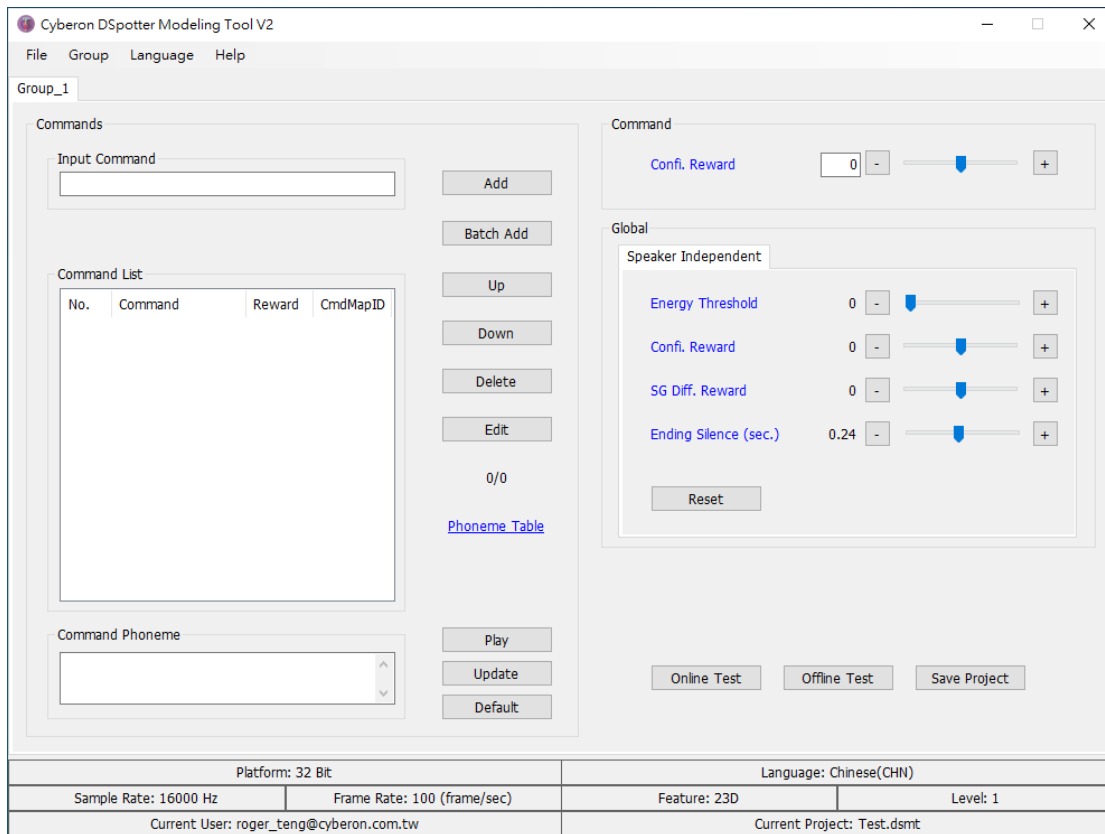




- **AllGroup\_MapID\_pack.bin**: the binary file that packs all command ID mapping bins together.
- **XXXX\_pack\_withTri.bin**: same as XXXX\_pack.bin, append CYTrimap.mod in end of XXXX\_pack.bin
- **XXXX\_pack\_WithTriAndMapID.bin**: same as XXXX\_pack\_withTri.bin, append AllGroup\_MapID\_pack.bin in end of XXXX\_pack\_withTri.bin
- **XXXX\_pack\_withTxt.bin**: additional append .txt for each group, for DSpotter HL.  
CYBase.mod/Group\_1.mod/.../Group\_x.mod/[Group\\_1.txt](#)/.../[Group\\_x.txt](#)
- **XXXX\_pack\_WithTxtAndMapID.bin**: same as XXXX\_pack\_withTxt.bin, append AllGroup\_MapID\_pack.bin in end of XXXX\_pack\_withTxt.bin
- **XXXX\_pack\_withTxtAndTri.bin**: same as XXXX\_pack\_withTxt.bin, append CYTrimap.mod in end of XXXX\_pack\_withTxt.bin
- **XXXX\_pack\_WithTxtAndTriAndMapID.bin**: same as XXXX\_pack\_withTxtAndTri.bin, append AllGroup\_MapID\_pack.bin in end of XXXX\_pack\_withTxtAndTri.bin

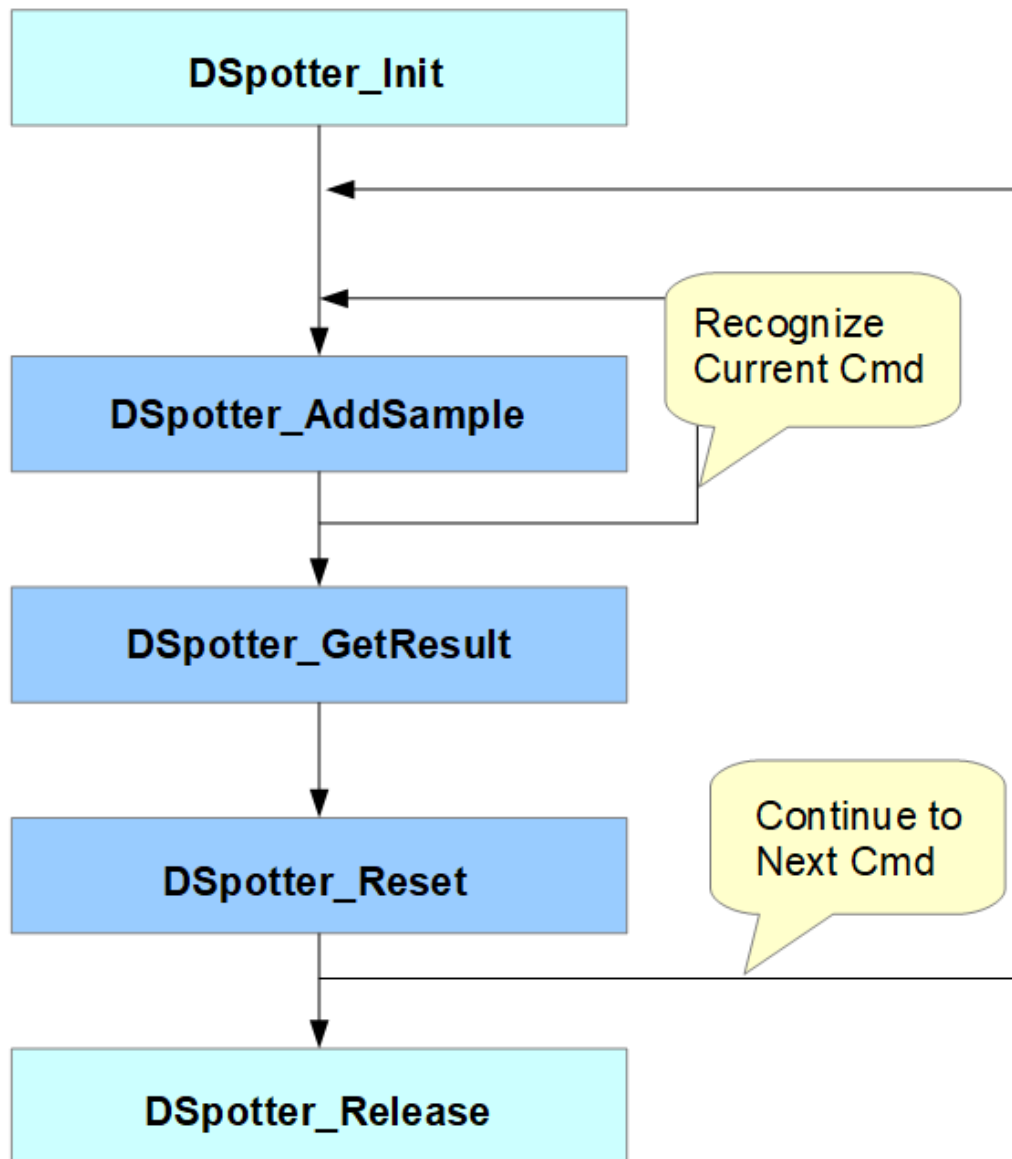
### Tools

- **DSpotter Model Tool**, a Microsoft Win32-based tool for developers to create command models for DSpotter recognition engine. Prior registration is required before developers can use DSpotter Model Tool. Contact with Cyberon if you are new to DSpotter.



## 4. DSpotter SDK API Standard Version

### 4.1. Calling Flow Chart of Standard API



## 4.1. Initialize, Reset, and Release

### *DSpotter\_Init\_Multi*

#### Purpose

Create a recognizer for recognizing multiple groups of commands simultaneously.

#### Prototype

```
HANDLE DSpotter_Init_Multi(BYTE *lpbyCYBase, BYTE *lppbyModel[], INT  
nNumModel, INT nMaxTime, BYTE *lpbyMemPool, INT nMemSize, BYTE  
*lpbyPreserve, INT nPreserve, INT *pnErr);
```

#### Parameters

lpbyCYBase(IN): The background model, contents of CYBase.mod.

lpbyModel(IN): The command model.

nMaxTime(IN): The maximum buffer length in number of frames for keeping the status information of commands.

lpbyMemPool(IN/OUT): Memory buffer for the recognizer.

nMemSize(IN): Size in bytes of the memory buffer *lpbyMemPool*.

lpbyPreserve (IN/OUT): Preserve param, give NULL.

nPreserve (IN): Preserve param, give 0.

pnErr(OUT): The return code.

#### Return value

Return the handle of a recognizer when success or NULL otherwise.

#### Remarks

It is highly recommended that the value of *nMaxTime* should be greater than the maximum duration of all commands, and recognizer could keep the status information of commands during recognition. Note that higher value of *nMaxTime* will increase the memory usage.

A statically reserved buffer of memory pointed by *lpbyMemPool* is required to call this function. Developers can get the memory buffer size *nMemSize* in advance by using the command line tool [DSpotter\\_GetMemoryUsage](#). This memory buffer can be recycled and used by other functions when the recognition task finishes after calling [DSpotter\\_Release\(...\)](#).

Pointer *pnErr* receives the return code after calling this function, *DSPOTTER\_SUCCESS* indicating success, otherwise a negative error code is returned. This pointer can be NULL, The maximum number of command models is 10.

### ***DSpotter\_Reset***

#### **Purpose**

Reset the recognizer before performing recognition.

#### **Prototype**

```
INT DSpotter_Reset(HANDLE hDSpotter);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### ***DSpotter\_Release***

#### **Purpose**

Release a recognizer.

#### **Prototype**

```
INT DSpotter_Release(HANDLE hDSpotter);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### ***DSpotter\_GetMemoryUsage\_Multi***

#### **Purpose**

Get current memory usage.

#### **Prototype**

```
INT DSpotter_GetMemoryUsage_Multi(BYTE *lpbyCYBase, BYTE *lppbyModel[],  
INT nNumModel, INT nMaxTime);
```

#### **Parameters**

lpbyCYBase(IN): The background model, contents of CYBase.mod

lppbyModel(IN): An array of command models to be recognized simultaneously.

nNumModel(IN): Number of models in array *lppbyModel*.

nMaxTime(IN): The maximum buffer length in number of frames for keeping the status of commands.

#### **Return value**

The memory size in bytes or error code.

## 4.2. Recognition

Functions in this section are designed to perform recognition process for the recognizer. For some platforms with relatively limited RAM, the pseudo codes below demonstrate how to use **union** data type of C language to store memory buffers for DSpotter engine, playback, and functions of developer's application in the same location. DSpotter engine retains the memory buffer pointed by *lpbyMemPool* until [\*DSpotter\\_Release\(...\)\*](#) is called, after which the buffer is released and can be recycled and reused by other functions. The pseudo codes below show the calling sequence for always-listening voice recognition:

*// Declare shared memory using data type union in C.*

```
union ShareMem {  
    BYTE lpbyMemPool[N];  
    // N can be obtained with function DSpotter_GetMemoryUsage_Multi(...).  
    SHORT lpsPlayBuffer[...];  
    <Other buffers used by application>  
} ShareMem;
```

DoVR(...)

```
{  
    // Create a recognizer  
    hDSpotter = DSpotter_Init_Multi(..., ShareMem.lpbyMemPool, N, NULL, 0, ...);  
    if (hDSpotter == NULL)  
        goto L_ERROR;
```

<Start Recording>

```
while (1)  
{  
    <Get PCM samples from recording device>  
    if (DSpotter_AddSample(...) == DSPOTTER_SUCCESS)  
    {  
        nID = DSpotter_GetResult(...);  
        DSpotter_GetResultEPD(...); // Optional  
        nScore = DSpotter_GetResultScore(...); // Optional  
        break;  
    }  
}
```

L\_ERROR:

<Stop Recording>

```
DSpotter_Release(...);  
// Share memory ShareMem can be used by other functions after DSpotter_Release(...).
```

<Play Prompt using memory ShareMem.lpsPlayBuffer >

```
}
```

## DSpotter\_AddSample

### Purpose

Add voice samples to the recognizer and perform recognition.

### Prototype

```
INT DSpotter_AddSample(HANDLE hDSpotter, SHORT *lpsSample, INT
nNumSamples);
```

### Parameters

hDSpotter (IN): Handle of the recognizer.

lpsSample(IN): An array of 16kHz, 16-bit, mono-channel PCM raw data.

nNumSamples(IN): Number of samples in *lpsSample*.

### Return value

Result	Comment
DSPOTTER_SUCCESS	A recognition result is concluded, and application can call <a href="#">DSpotterGetResult(...)</a> to retrieve the result.
DSPOTTER_ERR_NeedMoreSample	Recognition result has not been found yet, and need to call this function again to add more samples to the recognizer.
DSPOTTER_ERR_Rejected	A rejected result is concluded, and application can call <a href="#">DSpotterGetResult(...)</a> to retrieve the result.
Other negative error code	

### Remarks

Application should call this function repetitively to add recorded PCM raw data into the recognizer for recognition to proceed until a recognition is found, at which moment this function returns *DSPOTTER\_SUCCESS*, and the application can then call [DSpotter\\_GetResult\(...\)](#) to retrieve the recognized result. The recommended length of the input array of samples *lpsSample* is 480 samples (= 960 bytes).

## DSpotter\_GetResult

### Purpose

Get the recognition result from the recognizer.

### Prototype

```
INT DSpotter_GetResult(HANDLE hDSpotter);
```

### Parameters

hDSpotter (IN): Handle of the recognizer.

### Return value

Return the zero-based command ID when success or negative error code otherwise. If there are more than one command models being recognized simultaneously, the command ID is enumerated in order. For example, if there are 2 models containing  $n_1$  and  $n_2$  commands respectively, the ID for the third command in the second model is  $n_1+2$ .

### ***DSpotter\_GetResultEPD***

#### **Purpose**

Get the boundary information of the current recognition result.

#### **Prototype**

```
INT DSpotter_GetResultEPD(HANDLE hDSpotter, INT *pnWordDura, INT *pnEndSil,  
INT *pnNetworkLatency);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

pnWordDura(OUT): Duration of the result in number of samples.

pnEndSil(OUT): Ending silence length in number of samples.

pnNetworkLatency (OUT): Model delay length in number of samples.

#### **Return value**

Return the command ID when success, or negative error code otherwise.

#### **Remarks**

EPD stands for end-point detection. DSpotter determines the completion of an input voice command by counting the length of the ending silence. This function retrieves the command duration and length of the ending silence. Developers can also calculate the command start time if necessary. In the application, number of added samples is recorded with variable *nTotAddSample*. Then the start time *nStartTime* is

$$nStartTime = nTotAddSample - *pnWordDura - *pnEndSil - *pnNetworkLatency;$$

### ***DSpotter\_GetResultScore***

#### **Purpose**

Get the reliability score of the current recognition result.

#### **Prototype**

```
INT DSpotter_GetResultScore (HANDLE hDSpotter, INT *pnConfi, INT *pnSGDiff,  
INT *pnFIL);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

\*pnConfi(OUT): Score of Confi.

\*pnSGDiff (OUT): Score of SG Diff

\*pnFIL (OUT): Score of Fil

#### **Return value**

Return the non-negative reliability score of the recognition result when success, or negative error code otherwise.

Higher confidence score means voice is more similar to command model.

Higher SG Difference score means voice is more different from Silence/Garbage.

Higher Fil score means voice is more different from Filter model.

***DSpotter\_GetCmdEnergy*****Purpose**

Get the energy of recognition result in RMS value from the recognizer.

**Prototype**

```
INT DSpotter_GetCmdEnergy(HANDLE hDSpotter);
```

**Parameters**

hDSpotter (IN): Handle of the recognizer.

**Return value**

Return the energy of recognition result in RMS value when success, or negative error code otherwise

***DSpotter\_GetNumWord*****Purpose**

Get the number of commands in the input model.

**Prototype**

```
INT DSpotter_GetNumWord(BYTE *lpbyModel);
```

**Parameters**

lpbyModel(IN): The command model.

**Return value**

Return the number of commands when success, or negative error code otherwise.



## ***DSpotter\_SetEndSil***

### **Purpose**

Set group ending silence.

### **Prototype**

**INT DSpotter\_SetEndSil(HANDLE hDSpotter, INT nEndSil);**

### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nEndSil (IN): Ending silence. The range is [0, 16], lower value will make the engine quicker to return a result, Set 1 is 0.03s, The default is 8(0.24s).

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

The ending silence is a global attribute that applies to the entire model. It defines the duration of silence after the voice input for the engine to determine the end of a voice command. Though a longer ending silence makes the engine slower or more "picky" to respond to user's voice input, it can usually give more stable recognition results with less false triggers.

Set value by this API will overwrite all values set by [\*DSpotter\\_SetCmdEndSil\(..\)\*](#).

## ***DSpotter\_SetCmdEndSil***

### **Purpose**

Set command ending silence.

### **Prototype**

**INT DSpotter\_SetCmdEndSil(HANDLE hDSpotter, INT nCmdIdx, INT nEndSil);**

### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nCmdIdx (IN): Command index.

nEndSil (IN): Response time. The range is [0, 16], lower value will make the engine quicker to return a result, Set 1 is 0.03s, The default is 8(0.24s).

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

The ending silence is a attribute that applies to the command. It defines the duration of silence after the voice input for the engine to determine the end of a voice command. Though a longer ending silence makes the engine slower or more "picky" to respond to user's voice input, it can usually give more stable recognition results with less false triggers.

Set value by this API will overwrite value set by [DSpotter\\_SetEndSil\(...\)](#).

## ***DSpotter\_GetCmdEndSil***

### **Purpose**

Get command ending silence.

### **Prototype**

**INT DSpotter\_GetCmdEndSil(HANDLE hDSpotter, INT nCmdIdx);**

### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nCmdIdx (IN): Command index.

### **Return value**

Return value if successful, or negative error code otherwise.

### **Remarks**

### ***DSpotter\_SetConfiReward***

#### **Purpose**

Set group confidence reward.

#### **Prototype**

**INT DSpotter\_SetConfiReward(HANDLE hDSpotter, INT nReward);**

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nReward (IN): Confi Reward. The range is [-100, 100], lower reward will make the engine more "picky" to return a result.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

Confidence score means voice is how much similar to command model.

The group confidence reward is a global threshold that applies to the entire model. A lower reward makes the engine more "picky" to return a result. It is recommended to perform sufficient amount of field tests from different users if the rejection level is changed from its default value.

### ***DSpotter\_GetConfiReward***

#### **Purpose**

Get group confidence reward.

#### **Prototype**

**INT DSpotter\_GetConfiReward(HANDLE hDSpotter, INT \*pnErr);**

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

pnErr (IN/OUT): *DSPOTTER\_SUCCESS* if successful, or negative error code otherwise.

#### **Return value**

Return value.

#### **Remarks**

### ***DSpotter\_SetCmdConfiReward***

#### **Purpose**

Set command confidence reward.

#### **Prototype**

```
INT DSpotter_SetCmdConfiReward(HANDLE hDSpotter, INT nCmdIdx, INT  
nReward);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nCmdIdx (IN): Command index.

nReward (IN): Confi Reward. The range is [-100, 100], lower reward will make the engine more "picky" to return a result.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

Engine will add group and command confidence reward as confidence score offset.

### ***DSpotter\_GetCmdConfiReward***

#### **Purpose**

Get command confidence reward.

#### **Prototype**

```
INT DSpotter_GetCmdConfiReward(HANDLE hDSpotter, INT nCmdIdx, INT *pnErr);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nCmdIdx (IN): Command index.

pnErr (IN/OUT): *DSPOTTER\_SUCCESS* if successful, or negative error code otherwise.

#### **Return value**

Return value.

#### **Remarks**

### ***DSpotter\_SetSGDiffReward***

#### **Purpose**

Set group SG difference reward.

#### **Prototype**

**INT DSpotter\_SetSGDiffReward(HANDLE hDSpotter, INT nReward);**

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

nReward (IN): SG Difference Reward. The range is [-100, 100], lower reward will make the engine more "picky" to return a result.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

SG Difference score means voice is how much different from Silence/Garbage.

The group SG difference reward is a global threshold that applies to the entire model. A lower reward makes the engine more "picky" to return a result. It is recommended to perform sufficient amount of field tests from different users if the rejection level is changed from its default value.

### ***DSpotter\_GetSGDiffReward***

#### **Purpose**

Get group SG difference reward.

#### **Prototype**

**INT DSpotter\_GetSGDiffReward(HANDLE hDSpotter, INT \*pnErr);**

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

pnErr (IN/OUT): *DSPOTTER\_SUCCESS* if successful, or negative error code otherwise.

#### **Return value**

Return value.

#### **Remarks**

***DSpotter\_SetEnergyTH*****Purpose**

Set the energy threshold of recognition result in RMS value.

**Prototype**

```
INT DSpotter_SetEnergyTH(HANDLE hDSpotter, INT nEnergyTH);
```

**Parameters**

hDSpotter (IN): Handle of the recognizer.

nEnergyTH (IN): Command energy in RMS value. The range is [0, 32767].

**Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

**Remarks*****DSpotter\_GetEnergyTH*****Purpose**

Get the energy threshold of recognition result in RMS value.

**Prototype**

```
INT DSpotter_GetEnergyTH(HANDLE hDSpotter);
```

**Parameters**

hDSpotter (IN): Handle of the recognizer.

**Return value**

Return value if successful, or negative error code otherwise.

**Remarks**

### ***DSpotter\_SetResultMapID\_Sep***

#### **Purpose**

Set single command Mapping ID bin to engine.

#### **Prototype**

```
INT DSpotter_SetResultMapID_Sep(HANDLE hDSpotter, BYTE *lpbMapID);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

lpbMapID (IN): The command mapping ID bin.

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

The command mapping ID bin is created by DSMT, user could set multi commands to one index with command mapping ID bin.

### ***DSpotter\_SetResultMapID\_Multi***

#### **Purpose**

Set multi command Mapping ID bins to engine.

#### **Prototype**

```
INT DSpotter_SetResultMapID_Multi(HANDLE hDSpotter, BYTE *lppbMapID[], INT  
nNumMapID);
```

#### **Parameters**

hDSpotter (IN): Handle of the recognizer.

lppbMapID (IN): The command mapping ID bins.

nNumMapID (IN): number of command mapping ID bins

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

nNumMapID must same as nNumModel which used in DSpotter\_Init\_Multi.

***DSpotter\_GetResultMapID*****Purpose**

Get the mapping recognition result from the recognizer.

**Prototype**

```
INT DSpotter_GetResultMapID(HANDLE hDSpotter);
```

**Parameters**

hDSpotter (IN): Handle of the recognizer.

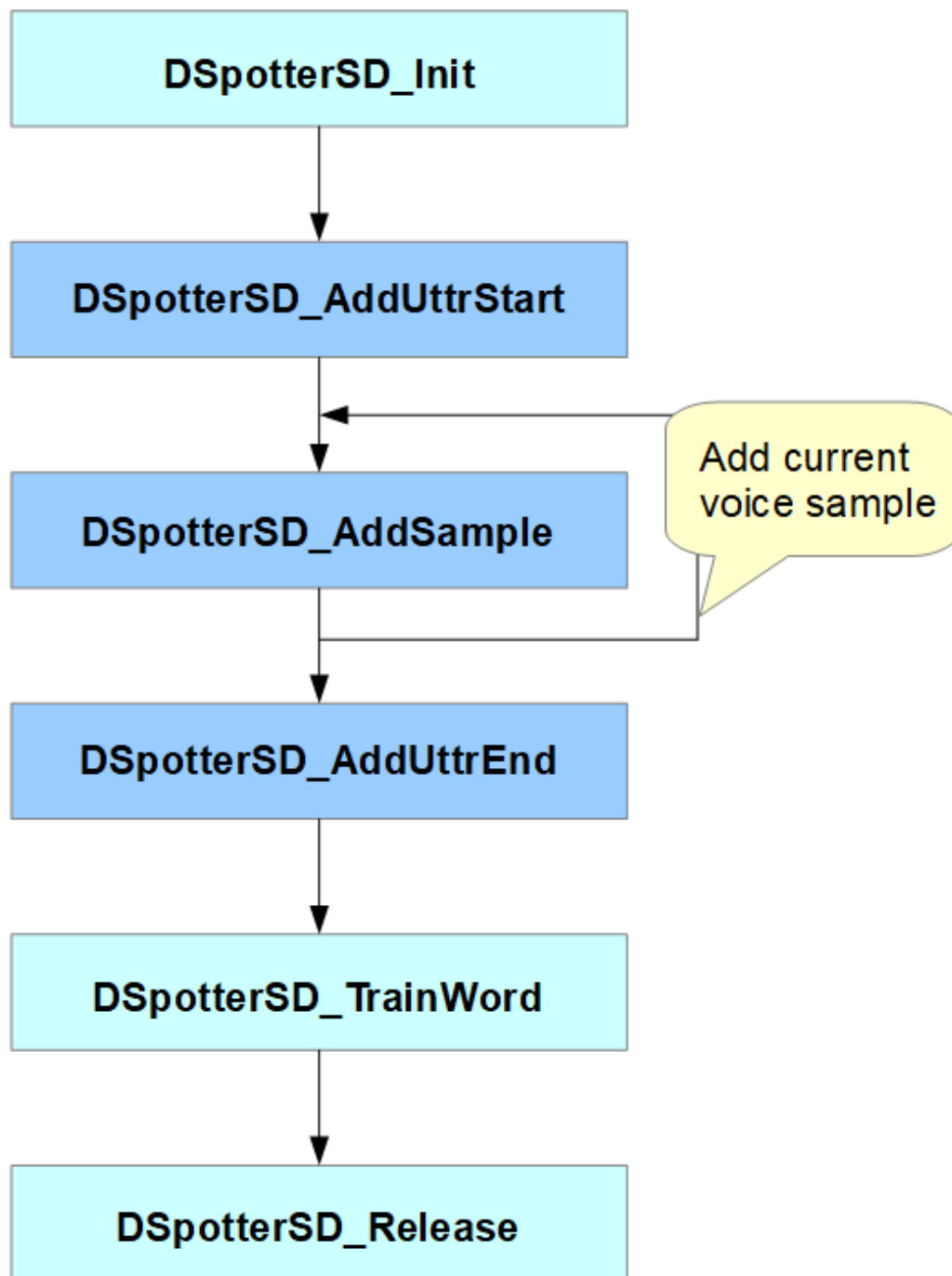
**Return value**

Return the zero-based mapping command ID when success or negative error code otherwise.



## 5. DSpotter SDK API Advanced Version

### 5.1. Calling Flow Chart of Advanced API



## 5.2. Initialize and Release

### *DSpotterSD\_Init*

#### Purpose

Create a DSpotter voice tag trainer.

#### Prototype

```
HANDLE DSpotterSD_Init(BYTE *lpbyCYBase, BYTE *lpbyTrimap, BYTE
*lpbyMemPool, INT nMemSize, INT *pnErr);
```

#### Parameters

lpbyCYBase(IN): The background model for trainer, contents of CYBase.mod

lpbyTrimap (IN): The phoneme map model for trainer, contents of CYTrimap.mod

lpbyMemPool(IN/OUT): Memory buffer for the trainer

nMemSize(IN): Size in bytes for memory buffer *lpbyMemPool*.

pnErr(OUT): The return code.

#### Return value

Return the handle of a trainer when success or NULL otherwise.

#### Remarks

A background model CYBase.mod is required to train a voice tag. The trainer extracts parameters from the input CYBase.mod/Group\_x.mod/CYTrimap.mod, and using the training utterances provided by the user to create new command. Models sharing the same CYBase.mod, including the speaker-independent (SI) ones created from DSpotter Model Tool and the speaker-dependent (SD) voice tags trained here, can be put together and recognized by DSpotter engine simultaneously.

A statically reserved buffer of memory pointed by *lpbyMemPool* is required to call this function. This memory buffer can be recycled and used by other functions when the training task ends after calling [\*DSpotterSD\\_Release\(...\)\*](#).

Pointer *pnErr* receives the return code after calling this function, *DSPOTTER\_SUCCESS* indicating success, otherwise a negative error code is returned. This pointer can be NULL.

***DSpotterSD\_GetMemoryUsage*****Purpose**

Get current memory usage of training.

**Prototype**

```
INT DSpotterSD_GetMemoryUsage(BYTE *lpbyCYBase, BYTE *lpbyTrimap);
```

**Parameters**

lpbyCYBase(IN): The pointer of background model, contents of CYBase.mod.

lpbyTrimap (IN): The phoneme map model for trainer, contents of CYTrimap.mod

**Return value**

The memory size in bytes or error code.

**Remarks**

Function will return memory usage or error code.

***DSpotterSD\_Release*****Purpose**

Release the trainer.

**Prototype**

```
INT DSpotterSD_Release(HANDLE hDSpotter);
```

**Parameters**

hDSpotter(IN): Handle of the trainer.

**Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### 5.3. Training

Functions in this section are designed to perform training process for SD voice tag. The pseudo codes below show the calling sequence for training an SD voice tag:

```
// Declare shared memory using data type union in C.
union ShareMem {
    BYTE lpbyMemPool[N];
    // N can be obtained with function DspotterSD_GetMemoryUsage(...).
    <Other buffers used by application>
} ShareMem;

DoVR_train(...)
{
    <Prepare storage (possibly in flash) for utterance buffer>

    // Create a trainer
    hDspotter = DspotterSD_Init(..., ShareMem.lpbyMemPool, N, ...);
    if (hDspotter == NULL)
        goto L_ERROR;

    // Preparation stage: adding utterances to train a voice tag
    if (DspotterSD_AddUtrStart(...) != DSPOTTER_SUCCESS)
        goto L_ERROR;

    <Start Recording>

    while (1)
    {
        <Get PCM samples from recording device>
        if (DspotterSD_AddSample(...) != DSPOTTER_ERR_NeedMoreSample)
            break;
        // Use DspotterSD_GetUtrEPD(...) to get the starting point of the input
        // utterance, and then start to compress it and write to data flash. (Optional)
        // if (DspotterSD_GetUtrEPD(...) == DSPOTTER_SUCCESS)
        //     <Compress recorded voice data and write it to data flash>
    }

    <Stop Recording>

    if (DspotterSD_AddUtrEnd(...) != DSPOTTER_SUCCESS)
        goto L_ERROR;
    // Use DspotterSD_GetUtrEPD(...) to get the ending point of the input utterance,
    // and move the compressed voice to external flash of larger size. (Optional)
    // if (DspotterSD_GetUtrEPD(...) == DSPOTTER_SUCCESS)
    //     <Move the compressed voice data from data flash to SPI flash>

    // Training stage, and then add voice tag to the model for recognition
    if (DspotterSD_TrainWord(...) != DSPOTTER_SUCCESS)
        <Error Handling ...>

L_ERROR:

    DspotterSD_Release(...);
}
```

## ***DSpotterSD\_AddUttrStart***

### **Purpose**

Prepare to add a new utterance for training.

### **Prototype**

```
INT DSpotterSD_AddUttrStart(HANDLE hDSpotter, SHORT *IpsDataBuf, INT  
nBufSize);
```

### **Parameters**

hDSpotter(IN): Handle of the trainer.

IpsDataBuf (IN/OUT): The pointer of data buffer in **DATA FLASH** to store voice input.

nBufSize(IN): Size in bytes of the data buffer *IpsDataBuf*.

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

The data buffer *IpsDataBuf* is a pointer to internal data flash, or it can be pointing to external flash through SPI bus, as long as the bus is fast enough with address mapping hardware equipped. Internally in this function, user-implemented functions [\*DataFlash\\_Write\(...\)\*](#) and [\*DataFlash\\_Erase\(...\)\*](#), as described in the next section, are employed to access the data flash. If RAM is large enough, developers can also use RAM to simulate data flash when implementing these 2 functions. Note that DSpotter SDK assumes the page size for erasing flash is 4KB. For the consideration of efficiency, pointer *IpsDataBuf* has to be 4KB aligned and *nBufSize* a multiple of 4KB. If *IpsDataBuf* is NULL, this function returns the required size of the data buffer rounded to a multiple of 4KB. Currently the time duration of one voice tag is 3 second, which requires around 16KB data buffer. If given less than 16KB, the maximum length of voice tag shrinks by ratio. ex. 1.5 second voice tag for given 8KB data buffer

## ***DSpotterSD\_AddSample***

### **Purpose**

Add voice samples to the trainer for training.

### **Prototype**

```
INT DSpotterSD_AddSample(HANDLE hDSpotter, SHORT *lpsSample, INT  
nNumSample);
```

### **Parameters**

hDSpotter (IN): Handle of the trainer.

lpsSample(IN): An array of 16kHz, 16-bit, mono-channel PCM raw data.

nNumSamples(IN): Number of samples in *lpsSample*.

### **Return value**

*DSPOTTER\_ERR\_NeedMoreSample* indicates that the caller should call this function again, otherwise *DSPOTTER\_SUCCESS* for successfully obtaining a recognition results, or negative error code otherwise.

## ***DSpotterSD\_AddUttrEnd***

### **Purpose**

Finish the adding process for training a voice tag.

### **Prototype**

```
INT DSpotterSD_AddUttrEnd(HANDLE hDSpotter);
```

### **Parameters**

hDSpotter (IN): Handle of the trainer.

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

Training utterances are added to the trainer by calling [\*DSpotterSD\\_AddUttrStart\(...\)\*](#), [\*DSpotterSD\\_AddSample\(...\)\*](#) repeatedly, and [\*DSpotterSD\\_AddUttrEnd\(...\)\*](#), which constitutes the data preparation stage before training a voice tag.

### ***DSpotterSD\_GetUtrEPD***

#### **Purpose**

Get the boundary information of the currently added training utterance.

#### **Prototype**

```
INT DSpotterSD_GetUtrEPD(HANDLE hDSpotter, INT *pnStart, INT *pnEnd);
```

#### **Parameters**

hDSpotter (IN): Handle of the trainer.

pnStart(OUT): Starting point in samples

pnEnd(OUT): Ending point in samples

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

Usually this function is employed when developers want to store user's voice data for playback purpose. Values of *pnStart* and *pnEnd* are valid only when the function returns *DSPOTTER\_SUCCESS*.

### ***DSpotterSD\_SetEpdLevel***

#### **Purpose**

Set the boundary information of the currently added training utterance.

#### **Prototype**

```
INT DSpotterSD_SetEpdLevel(HANDLE hDSpotter, INT nEpdLevel);
```

#### **Parameters**

hDSpotter (IN): Handle of the trainer.

nEpdLevel (IN): Rejection level. The range is [0, 50]

#### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

#### **Remarks**

Set rejection level of training utterance EPD, if engine can't get EPD correctly, may set after engine init.

## ***DSpotterSD\_TrainWord***

### **Purpose**

Train a voice tag into a command model for recognition.

### **Prototype**

```
INT DSpotterSD_TrainWord(HANDLE hDSpotter, char *lpszModelAddr, INT  
nBufSize, INT *pnUsedSize);
```

### **Parameters**

hDSpotter (IN): Handle of the trainer.

lpszModelAddr(IN/OUT): The pointer of model buffer in **DATA FLASH**.

nBufSize(IN): Size in bytes of the model buffer pointed by *lpszModelAddr*.

pnUsedSize(OUT): Size in bytes of the voice tag pointed by *lpszWordAddr*.

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

This function solely constitutes the training stage. Call this function after the data preparation stage consisting of [DSpotterSD\\_AddUttrStart\(...\)](#), [DSpotterSD\\_AddSample\(...\)](#), and [DSpotterSD\\_AddUttrEnd\(...\)](#) calls.

Model buffer pointed by *lpszModelAddr* is in the format of an acoustic model containing only user trained voice tags. *lpszModelAddr* can be pointing to internal data flash, external SPI flash with address mapping mechanism supported, or RAM simulating flash. For more information, please see remarks for [DSpotterSD\\_AddUttrStart\(...\)](#).

nBufSize contains 340B header(H), and 400B for each voice tag(T) times the maximum number(N) of voice tag, Maximum nBufSize is 16KBytes.

$$\mathbf{nBufSize = H + N \cdot T}$$



## ***DSpotterSD\_DeleteWord***

### **Purpose**

Remove a voice tag from the model for recognition.

### **Prototype**

```
INT DSpotterSD_DeleteWord(HANDLE hDSpotter, char *lpzModelAddr, INT nIdx,  
INT *pnUsedSize);
```

### **Parameters**

hDSpotter (IN): Handle of the trainer.

lpzModelAddr(IN/OUT): The pointer of model buffer in **DATA FLASH**.

nIdx (IN): The command index.

pnUsedSize(OUT): Size in bytes of the model pointed by *lpzModelAddr* after removing the voice tag.

### **Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

### **Remarks**

When a command is successfully removed from a model, the command index for the other survival voice tags may be changed.

Parameter *lpzModelAddr* can be pointing to internal data flash, external SPI flash with address mapping mechanism supported, or RAM simulating flash. For more information, please see remarks for [\*DSpotterSD\\_AddUtrStart\(...\)\*](#).

***DSpotterSD\_SetBackgroundEnergyThreshd*****Purpose**

Set Energy threshold for SD Training.

**Prototype**

```
INT DSpotterSD_SetBackgroundEnergyThreshd(HANDLE hDSpotter, INT  
nThreshold);
```

**Parameters**

hDSpotter(IN): Handle of the trainer.

nThreshold (IN):Base RMS value, default is 1200.

**Return value**

*DSPOTTER\_SUCCESS* for success or negative error code otherwise.

**Remark**

While training voice tag,Engine will check first 10 frames's average RMS value,if RMS greater than 4 times of Base RMS value, [\*DSpotterSD\\_AddSample\(...\)\*](#),will return *DSPOTTER\_ERR\_NoisyEnvironment*.

## 5.4. User-Implemented Flash Operation Functions

DSpotter trainer needs data flash to store the training utterances to train a voice tag. To optimize the resource usage to the most extent, we leave the flexibility to application developers to and manipulate the data flash. It is therefore developers' responsibility to correctly implement the flash access functions listed in this section. Though these functions are intended for accessing flash, developers can actually use RAM to simulate flash in the implementation if RAM is large enough.

### *DataFlash\_Write*

#### **Purpose**

Write data into data flash.

#### **Prototype**

```
INT DataFlash_Write(BYTE *lpbyDest, BYTE *lpbySrc, INT nSize);
```

#### **Parameters**

lpbyDest (OUT): The pointer of destination data buffer in **DATA FLASH**.

lpbySrc (IN): The pointer of source data buffer.

nSize(IN): Size in bytes of the source data buffer *lpbySrc*.

#### **Return value**

0 for success or negative error code otherwise.

### *DataFlash\_Erase*

#### **Purpose**

Erase the flash given the starting address and its size.

#### **Prototype**

```
INT DataFlash_Erase(BYTE *lpbyDest, INT nSize);
```

#### **Parameters**

lpbyDest (OUT): The pointer of destination data buffer in **DATA FLASH**.

nSize(IN): Size in bytes of the destination data buffer *lpbyDest*.

#### **Return value**

0 for success or negative error code otherwise.

#### **Remarks**

Trainer assumed the flash page size is 4KB currently. In other words, the input value of *nSize* is always a multiple of 4KB and pointer *lpbyDest* is 4KB aligned.

## 6. DSpotter SDK Error Code Table

Error Symbol	Error Code
<i>DSPOTTER_SUCCESS</i>	0
<i>DSPOTTER_ERR_IllegalHandle</i>	-2001
<i>DSPOTTER_ERR_IllegalParam</i>	-2002
<i>DSPOTTER_ERR_LeaveNoMemory</i>	-2003
<i>DSPOTTER_ERR_LoadModelFailed</i>	-2005
<i>DSPOTTER_ERR_NeedMoreSample</i>	-2009
<i>DSPOTTER_ERR_BuildUserCommandFailed</i>	-2013
<i>DSPOTTER_ERR_Rejected</i>	-2020
<i>DSPOTTER_ERR_LicenseFailed</i>	-2200
<i>DSPOTTER_ERR_CreateModelFailed</i>	-2500
<i>DSPOTTER_ERR_WriteFailed</i>	-2501
<i>DSPOTTER_ERR_NotEnoughStorage</i>	-2502
<i>DSPOTTER_ERR_NoisyEnvironment</i>	-2503
<i>DSPOTTER_ERR_VoiceTooShort</i>	-2504
<i>DSPOTTER_ERR_VoiceTooLong</i>	-2505

## 7. DSpotter Supported Languages

Chinese (CHN)	English (Worldwide)	English(TWN)
Japanese	Korean	Turkish
Vietnamese	French	Spanish(EU)
German	Italian	Russian
Hindi	Cantonese(HK)	English(IN)
Norwegian	Arabic	Taiwanese
English(SG)	English(SEA)	English(PHI)
Chinese(CHN)/English	Thai	Chinese(TWN)
Portuguese(BRA)	Spanish(LA)	English(US)
English(UK)		