

# Nintendo DS

## Speech Recognition Library Specifications

Version 1.0.6

**The contents in this document are highly  
confidential and should be handled accordingly.**

**Confidential**

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd. and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

## Table of Contents

---

1	Introduction .....	6
1.1	Speech Recognition Features .....	6
1.2	Document Overview .....	6
2	System Overview .....	7
2.1	Software Configuration and Operation Overview.....	7
2.2	Basic Specifications.....	8
2.3	The Speech Recognition Library and Data .....	8
2.4	Work Memory .....	9
3	API Specifications .....	10
3.1	Recognition Lexicon Structure Definitions .....	10
3.2	Data Structure Definitions.....	11
3.2.1	Configured Languages (MW and CORE).....	11
3.2.2	Structure Storing the Language-Dependent Data (MW and CORE).....	12
3.2.3	Structure Storing the Configuration Parameters (MW and CORE).....	12
3.2.4	Structure Storing the Recognition Results (MW) .....	13
3.2.5	Callback Function Definitions (MW and CORE).....	13
3.3	The Core Engine API.....	15
3.3.1	Work Memory Size .....	15
3.3.2	Engine Initialization .....	15
3.3.3	Engine Termination Process.....	16
3.3.4	Recognition Dictionary Configuration .....	17
3.3.5	Parameter Configuration .....	17
3.3.6	Parameter Acquisition .....	18
3.3.7	Recognition Start Instruction .....	18
3.3.8	Speech Frame Processing .....	19
3.3.9	Recognition Result Acquisition (Recognition Number) .....	20
3.3.10	Recognition Result Acquisition (Recognition Score) .....	20
3.4	The Middleware API .....	21
3.4.1	Microphone Initialization .....	21
3.4.2	Microphone Termination Process .....	22
3.4.3	Middleware Initialization .....	23
3.4.4	Middleware Termination Process.....	24
3.4.5	Recognition Process .....	24
3.4.6	Recognition Abort Process .....	26
3.4.7	Setting the Priority of the Recognition Library Thread .....	27
3.4.8	Obtaining the Priority of the Recognition Library Thread.....	27
4	Appendix .....	28
4.1	Process Flow when Using Middleware API .....	28

4.2	Callback Functions .....	29
4.3	Phonetic Alphabet Transcription .....	30
4.3.1	US English.....	30
4.3.2	British .....	31
4.3.3	French .....	32
4.3.4	Spanish.....	33
4.3.5	German .....	34
4.3.6	Italian.....	35

## Code

Code 3-1 .....	11
Code 3-2 .....	12
Code 3-3 .....	12
Code 3-4 .....	13
Code 3-5 .....	13

## Figures

Figure 2-1 Software Configuration.....	7
Figure 3-1 Recognition Lexicon Structure .....	10
Figure 4-1 The Speech Recognition Middleware API Process Flowchart.....	28

## Revision History

Version	Revision Date	Description
1.0.6	12/26/2005	• Added a warning about arranging <code>dtb.bin</code> .
1.0.5	11/29/2005	• Added error code and its description to the <code>ASR_GetCandidateWord</code> function.
1.0.4	10/14/2005	• Added explanation about feature for registering phonetic symbols. • Revised 3.1 Recognition Dictionary Structure Definitions. Added to 4.3 About Phonetic Symbols.
1.0.3	07/12/2005	• Changed the specifications as follows to provide notification when the application detects microphone input and performs speech recognition processing. <ul style="list-style-type: none"><li>◦ Added a flag that indicates recognition processing is in progress in <code>ASR_RecogFrame()</code>.</li><li>◦ Added result that is communicated after the execution of <code>ASR_MWRecogStart()</code>.</li></ul> Added set and get functions for middleware recognition thread priority.
1.0.2	06/09/2005	Changed specification of <code>ASR_MWRecogStart()</code> . Added a usage method as “Tips.”
1.0.1	05/24/2005	Modified the explanation of <code>str.bin</code> and some expressions.
1.0.0	03/15/2005	Initial version.

# 1 Introduction

## 1.1 Speech Recognition Features

---

The Nintendo DS Speech Recognition library has the following features.

- **Recognition of up to 100 registered words (phrases)**
  - Any word or phrase registered as text can be recognized when spoken by the user.
  - Registered content can be changed using utterances.
- **Recognition of unspecified speakers**
  - Voices do not need to be recorded or learned ahead of time.
  - Tuned to support children's voices.
- **Multilingual Speech recognition with language-switching**
  - The target language for speech recognition can be changed while the application is running.
- **Real-time recognition processing**
  - The beginning and end of speech are detected automatically, and the recognition result is returned when the end of speech is detected.

See the *Nintendo DS Speech Recognition Library Programming Guide* for examples of applying speech recognition.

## 1.2 Document Overview

---

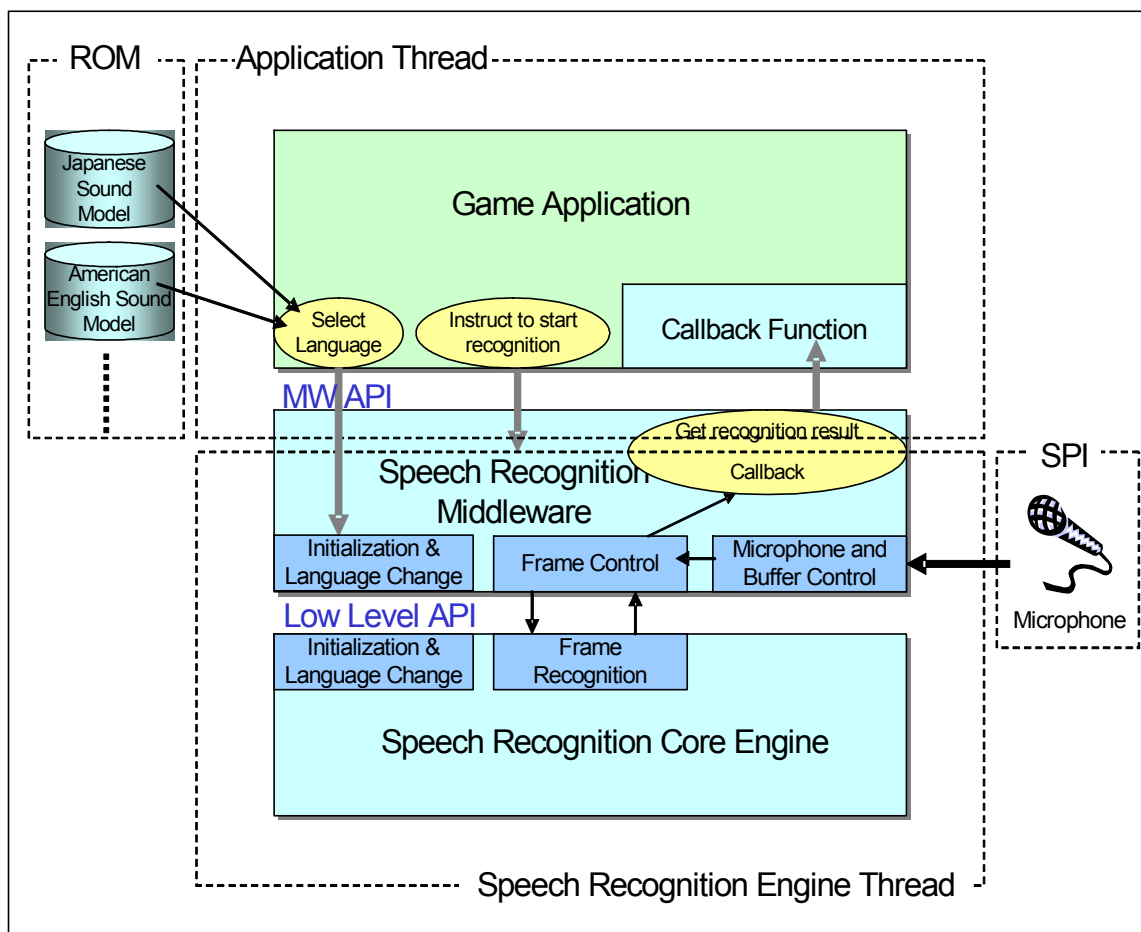
This document describes the speech recognition software module's API specifications (specifications for calling from the application) and assumes that it will be used as a function reference manual during application development. See the *Nintendo DS Speech Recognition Library Programming Guide* for cautions and the general process flow when using speech recognition.

The specifications described in this document correspond to version 1.0.4 of the Speech Recognition SDK. For compatibility with future versions of the Speech Recognition SDK, the language-dependent data released in version 1.0.4 cannot be used with Speech Recognition library version 1.0.3. (However, the language-dependent data released in version 1.0.3 can be used with the Speech Recognition library version 1.0.4.)

## 2 System Overview

### 2.1 Software Configuration and Operation Overview

Figure 2-1 Software Configuration



The speech recognition software module consists of the speech recognition middleware, the speech recognition core engine, and language-dependent sound data. The software module is provided as a library that links to the application and binary data that is placed in ROM. Because the library is not a reentrant library, it cannot be used simultaneously by multiple applications or multiple threads.

The speech recognition middleware controls the built-in DS microphone and passes input sound to the speech recognition core engine to perform speech recognition. The speech recognition process is executed in a separate thread created in the library. The GUI is used to prompt the user to speak. Other processes can be performed from the time the application issues the instruction to begin speech recognition until the callback function is called. The speech recognition core engine API can be used directly for speech recognition when recognizing sounds already stored in memory.

## 2.2 Basic Specifications

<b>Speech Input</b>	11.025 KHz, 16-bit linear PCM <sup>1</sup>	
<b>Multilanguage Support</b>	One engine supports multiple languages by switching language models	
<b>Word Recognition Specifications</b>	Character code	Japanese: Shift-JIS double-byte katakana English: ASCII (ISO 8859-1 for European languages)
	Input restrictions	Maximum of 100 words (phrases), total of 1000 bytes. (See <a href="#">3.1 - Recognition Dictionary Structure Definitions.</a> )
<b>Memory Resources <sup>2</sup></b>	ROM: 50 KB (code & data) RAM: 170 – 200 KB (work memory), 25 KB (sound sampling buffer) Language-dependent data: 70 – 190 KB per language (provided as binary data)	

**Note 1:** Perform sampling configuration with `MIC_SAMPLING_TYPE_SIGNED_12BIT_FILTER_OFF`

**Note 2:** The required resources vary according to the library to be linked and the language to be used.  
For details, see the release notes (`ReleaseNote.txt`) in the SDK.

## 2.3 The Speech Recognition Library and Data

Put the `libASR.h` header file in an `include` statement and link the `libASR.a` library in the application.

The language-dependent data required to change languages is supplied as a set of six binary files for each language. The application should place the required language-dependent data set in the ROM file system, load it into RAM as needed, set a value to the [prescribed structure](#), and then call the language configuration API.

<code>dtb.bin</code>	<b><u>Language-Dependent Data Group</u></b>  Separated into five files so it is easier to upgrade functionality. The application loads the data into a prescribed structure in RAM and passes the data to the recognition engine.
<code>hmm.bin</code>	
<code>phn.bin</code>	
<code>tree.bin</code>	
<code>mdc.bin</code>	
<code>str.bin</code>	<b><u>Initial Image of Environmental History Data</u></b>  This data is required to compensate for individual differences among Nintendo DS devices and is updated with each speech recognition process.  The latest data can be obtained with the API after the speech recognition process completes. So the application should use this initial image to first start up, store the latest data in non-volatile memory, and use it the next time speech recognition begins.

For details about placing the language-dependent data in ROM and using the API, see the *Nintendo DS Speech Recognition Library Programming Guide*.

A number of libraries and language models have been prepared for different applications. For details, see the release notes (`ReleaseNote.txt`) in the SDK.



## 2.4 Work Memory

---

All work memory used by the recognition engine is allocated by the application, and the work memory address is passed when the middleware or the recognition engine is initialized. Secure the required amount of work memory by using the [API that returns the required size for work memory](#). Be careful not to disable the work memory until the processing for recognition engine termination is performed.

## 3 API Specifications

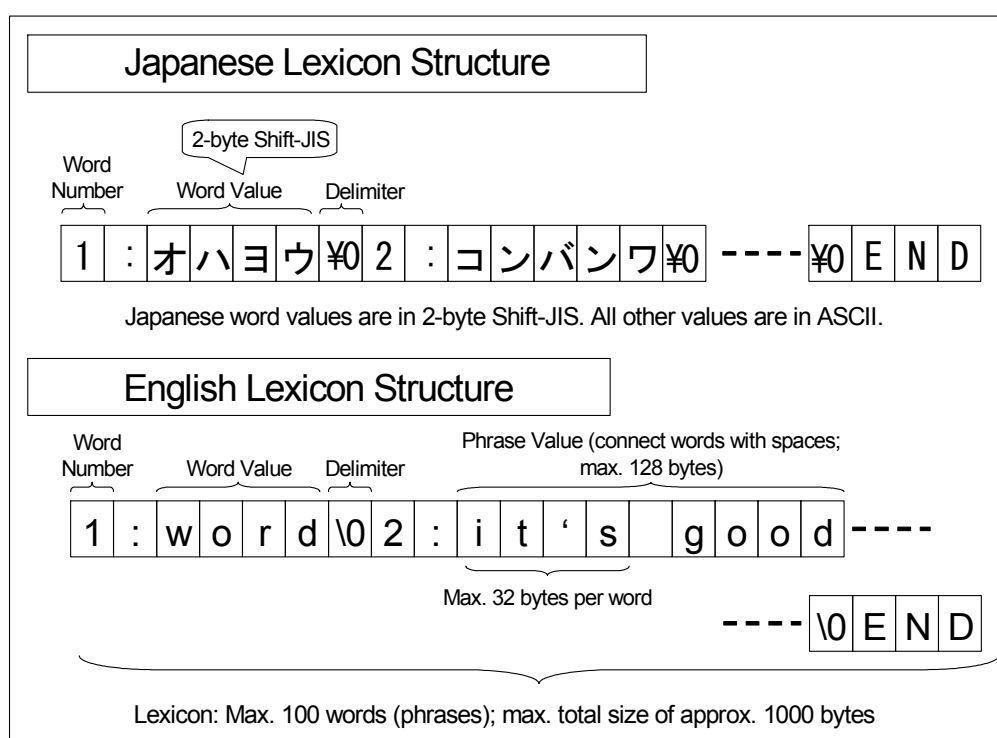
### 3.1 Recognition Lexicon Structure Definitions

Before speech recognition can be used, words (phrases) must be registered in the recognition lexicon. Because the recognition lexicon is text-based, the contents can be freely modified in the application.

- **Format**

Each word consists of a **Word Number** + ":" + **Word Value** + **Delimiter (NULL)**. As many as 100 words can be strung together. The end of the recognition dictionary is indicated with **END**.

Figure 3-1 Recognition Lexicon Structure



- **Phonetic alphabet format**

(Supported in Speech Recognition library versions 1.0.4 and later for non-Japanese languages)

Phonetic alphabet transcription consists of a *word number* + ":" + # + *phonetic alphabet transcription* + *delimiter character* (NULL). A maximum of 100 words can be concatenated. The end of the recognition dictionary is described by END. The phonetic transcription types that can be registered depend on the language. See **Error! Reference source not found. - Error! Reference source not found.** to see an example of this notation and a list of phonetic transcriptions for each language. Because the *word notation* and *phonetic alphabet transcription* format can be simultaneously described, sub-dictionaries can be created for different pronunciations of words in dialects and special pronunciation for character names.

## Recognition Lexicon Specifications and Limitations

	Japanese	English & Other European Languages
<b>Word Number</b>	Single-byte numeric character of up to four digits (1 to 9999) Numerals followed by a colon (":") ASCII 0x3A)	
<b>Word Value</b>	<ul style="list-style-type: none"><li>Shift-JIS double-byte katakana (0x8340 to 0x8394); long-tone symbol "—" (0x815B)</li><li>Up to 128 bytes per word</li></ul>	<ul style="list-style-type: none"><li>Single-byte alphabet (ASCII) and some symbols (apostrophe ('), hyphen (-), and blank space)</li><li>Can use spaces to string words together</li><li>Maximum number of characters in a single word = 32 bytes; 128 bytes per value. (see Fig. 3.1)</li></ul>
<b>Lexicon</b>	100 words (phrases); total of about 1000 bytes for word values <sup>1</sup>	

**Note 1:** Even if word notation is less than 1000 bytes, complicated pronunciations may cause internal resource errors.

## 3.2 Data Structure Definitions

---

The data structures defined in `libASR.h` are explained below. **MW** refers to data used by the middleware API, and **CORE** refers to data used by the core engine API.

### 3.2.1 Configured Languages (MW and CORE)

---

#### Code 3-1

```
typedef enum {  
    ASR_JAPANESE = 0x1,    // Japanese  
    ASR_BRITISH,           // British English  
    ASR_US_ENGLISH,        // American English  
    ASR_FRENCH,            // French  
    ASR_SPANISH,           // Spanish  
    ASR_GERMAN,            // German  
    ASR_ITALIAN,           // Italian  
    ASR_DUTCH,             // Dutch  
    ASR_CHINESE            // Chinese  
} ASR_Language;
```

The API that starts speech recognition (the engine initialization API for the core engine API) specifies the recognition language.

**Note:** The initial release allows only `ASR_JAPANESE` or `ASR_US_ENGLISH` to be specified.

### 3.2.2 Structure Storing the Language-Dependent Data (MW and CORE)

#### Code 3-2

```
typedef enum {
    ASR_DTB_DATA_TYPE = 0x1,
    ASR_HMM_DATA_TYPE,
    ASR_TREE_DATA_TYPE,
    ASR_PHN_DATA_TYPE,
    ASR_MDC_DATA_TYPE,
    ASR_STORAGE_DATA_TYPE
} ASR_DataType;

typedef struct tag_ASR_DataItemType {
    ASR_DataType    type; // Data type
    void            *data; // Pointer to binary data
    long            size; // Size of binary data
} ASR_DataItemType;
```

The application secures a memory region to store the language-dependent data and loads all the binary data placed in ROM. The application also prepares an array to store six `ASR_DataItemType` structures, configures member variables relating to all language-dependent data, and starts the API to initialize the recognition engine.

**Note:** Always arrange data (`dtb.bin`) that corresponds to `ASR_DTB_DATA_TYPE`.

See the *Nintendo DS Speech Recognition Library Programming Guide* for procedures to place language-dependent data in ROM, load data, and initialize the recognition engine.

### 3.2.3 Structure Storing the Configuration Parameters (MW and CORE)

#### Code 3-3

```
typedef struct tag_ASR_ParamType {
    short    maxCandidate; // Number of recognition result candidates (1 - 10)
    short    offTimeout;   // Timeout value for silence (0 - 10000[X10ms])
    short    onTimeout;    // Timeout value during speech (0 - 10000[X10ms])
    short    epdInterval;  // Interval for detecting end of speech (100 - 1000[ms])
} ASR_ParamType;
```

This structure is used by the recognition API (API for configuring and obtaining parameters in the core engine API).

Parameter	Description	Default Value
Number of recognition result candidates	The maximum number of candidates when results are returned by the recognition engine. (If the number of registered words is smaller than this value, results may get cut back so the number of returned candidates is smaller than the set value.)	10
Timeout value for silence	If the period of silence after recognition starts continues for longer than this value, the recognition process is aborted. (The actual duration is this value multiplied by 10 msec.)	1000 (10 sec.)

Timeout value during speech	If speech continues for longer than this value, the recognition process is aborted. (The time is the same as above.)	1000 (10 sec.)
Interval for detecting end of speech	After speech begins, if an interval of silence is longer than this value, it is assumed that speech has ended, and the recognition process begins.	350 ms.

**Note:** The default values are used for the core engine API if no values are set after the recognition engine has been initialized.

### 3.2.4 Structure Storing the Recognition Results (MW)

---

#### Code 3-4

```
typedef struct tag_ASR_ResultType {  
    short result;                // Recognition result (1 or above:  
                                // success & number of candidates, error code)  
    short candiNum[ASR_CANDI_NUM]; // Recognition number array  
    short candiScore[ASR_CANDI_NUM]; // Recognition evaluation score array  
} ASR_ResultType;
```

The application secures this structure and passes the pointer to the API that directs speech recognition to start. When recognition ends (including errors such as timeouts), the recognition engine sets the result in this structure and delivers it as the second argument of the callback function.

### 3.2.5 Callback Function Definitions (MW and CORE)

---

#### Code 3-5

```
typedef void (*ASRCallback) (ASR_RESULT_CODE result, void *addr);
```

The callback function is specified by the microphone initialization API and the start recognition API. The callback function is called in the following cases.

Callback Origin	Status	1st Argument (Result)	2nd Argument (Pointer Destination)
Microphone initialization API (MW and CORE)	Initialization completed	ASR_RESULT_MICINIT_COMPLETED	(short *) 0
API that directs recognition to start (MW only)	Recognition process started	ASR_RESULT_RECOG_STARTED	(ASR_ResultType *) (Address specified when calling the API)
	Recognition process completed	ASR_RESULT_RECOG_COMPLETED	
	Recognition process failed	ASR_RESULT_RECOG_FAILED	

The application performs speech recognition using the following process.

1. Calls the microphone initialization API and waits for the callback function. (The DS hardware specifications require 0.5 second. Instruct the user not to talk during this time by using the GUI or some other means.)
2. After obtaining `ASR_RESULT_MICINIT_COMPLETE`, initializes the middleware, calls the recognition process API, and waits for callback.
3. If `ASR_RESULT_RECOG_STARTED` is returned, use an icon if necessary to indicate to the user that the microphone has received input and speech recognition processing is in progress.
4. After obtaining `ASR_RESULT_RECOG_COMPLETED`, performs processes based on the result obtained from the recognition result from the second argument address.

To learn more details about the application processes, see the *Nintendo DS Speech Recognition Library Programming Guide*.

To learn more details about error codes and recognition results, see 3.4 - The Middleware API.

## 3.3 The Core Engine API

---

Because this API group is platform independent, use the application to control the microphone. Normally, the middleware API is used and is described in **Error! Reference source not found.** - **Error! Reference source not found.**

### 3.3.1 Work Memory Size

---

#### Declaration

```
long ASR_GetWorkMemSize(void)
```

#### Arguments

None.

#### Return Values

Positive value	The needed size of work memory
----------------	--------------------------------

#### Description

This function returns the amount of work memory required by the recognition engine. The application uses this value to allocate a memory region, and then passes the address to the engine initialization API.

### 3.3.2 Engine Initialization

---

#### Declaration

```
short ASR_RecogInit(ASR_Language language, ASR_DataItemType *data,  
                    short itemCount, short *workMem)
```

#### Arguments

ASR_Language language	Specifies the recognition language
ASR_DataItemType *data	Pointer to the language-dependent data group
short itemCount	The number of sets of language-dependent data
short *workMem	The address of work memory

#### Return Values

ASR_NO_ERROR	Successful
ASR_ERROR_INTERNAL	Internal error (e.g., when an invalid language is specified)
ASR_ERROR_PARAM	Argument error

#### Description

This function changes the recognition language and initializes the recognition engine.

### Comments

Have the application load the six sets of language-dependent data located in ROM to RAM, arrange the data as an array of `ASR_DataItemType` structures, and then pass them to this API. Read about the details of this procedure in the Nintendo DS Speech Recognition Library Programming Guide.

Obtain the required work memory size with `ASR_GetWorkMemSize()`, and have the application secure a memory region of that size. Be careful not to disable this memory region until the engine-termination process is performed. Because the recognition engine cannot verify the stability of the memory region, operations can not be guaranteed if the memory region has not been verified.

## 3.3.3 Engine Termination Process

### Declaration

```
short ASR_RecogEnd(void *data)
```

### Arguments

`void *data` Pointer to the buffer storing the Environmental history data (100 bytes)

### Return Values

Positive value greater than 0	Size of the loaded environmental history data
<code>ASR_ERROR_INTERNAL</code>	Recognition engine is not initialized

### Description

This function discards the data related to the recognition engine. When there is environmental history data that can improve recognition accuracy, that data is written to the address of the second argument, and the size of the data is returned.

### Comments

The environmental history data is used to store some characteristics of the speech data passed to the recognition engine and to improve recognition accuracy. When data can be stored in nonvolatile memory, recognition accuracy can be improved by passing the data to the recognition engine the next time the engine is initialized.

After this API has ended, the work memory passed during engine initialization can be discarded.



### 3.3.4 Recognition Dictionary Configuration

---

#### Declaration

```
short ASR_SetDict(unsigned char *dict, short word_count)
```

#### Arguments

unsigned char *dict	Pointer to the recognition dictionary
short word_count	Number of words in dictionary

**Note:** For details about recognition dictionaries, see [3.1 - Recognition Dictionary Structure Definitions](#).

#### Return Values

ASR_NO_ERROR	Successful
ASR_ERROR_INTERNAL	Called before the engine was initialized, or some internal error occurred
ASR_ERROR_PARAM	Argument error
ASR_ERROR_DIC_TOO_LARGE	Too many registered words (more than 100)
ASR_ERROR_DIC_LONG_SPELL	Exceeds length for registered words
ASR_ERROR_DIC_MISSPEL	Word misspelled
ASR_ERROR_DIC_CONV	Word-conversion internal error (when the languages are not the same)

#### Description

This function stores the recognition dictionary in the engine. No recognition dictionary is stored immediately after the engine is first initialized, so call this API at least once. Once a dictionary is stored, it will remain until this API is called again.

### 3.3.5 Parameter Configuration

---

#### Declaration

```
short ASR_SetParam(ASR_ParamType *param)
```

#### Arguments

ASR_ParamType *param	Pointer to the structure storing the configuration parameters
----------------------	---

(For details, see [3.2.3 - Structure Storing the Configuration Parameters](#))

#### Return Values

ASR_NO_ERROR	Successful
ASR_ERROR_PARAM	Argument error
ASR_ERROR_INTERNAL	Called before the engine was initialized, or some internal error occurred

**Description**

This function sets the number of recognition candidates, the silence and speech timeout periods, and the interval of silence to determine the end of speech.

**Comments**

Once the parameters have been set, the values are retained until this function is called again. After using the API that obtains the current values for the parameters, call this function to change only those parameters that need to be changed. If this function is called before the recognition engine has been initialized, the settings will be ignored. (When the engine is initialized, the parameters are set to the default values.)

**3.3.6 Parameter Acquisition****Declaration**

```
short ASR_GetParam(ASR_ParamType *param)
```

**Arguments**

ASR\_ParamType \*param      Pointer to the structure storing the configuration parameters  
(For details, see [3.2.3 - Structure Storing the Configuration Parameters](#))

**Return Values**

ASR_NO_ERROR	Successful
ASR_ERROR_PARAM	Argument error
ASR_ERROR_INTERNAL	Called before the engine was initialized, or some internal error occurred

**Description**

This function writes the current settings for the number of recognition candidates, the silence timeout value, and the speech timeout value to the memory location specified by `param`. Calling this function before the initializing the recognition engine results in an undefined result.

**3.3.7 Recognition Start Instruction****Declaration**

```
short ASR_RecogStart(void)
```

**Arguments**

None.

**Return Values**

ASR_NO_ERROR	Successful
ASR_ERROR_INTERNAL	Called before the engine was initialized, or some internal error occurred

## Description

This function initializes the recognition engine for each speech input. After calling this function, call the frame recognition function, `ASR_RecogFrame`.

## 3.3.8 Speech Frame Processing

---

### Declaration

```
short ASR_RecogFrame(short *framePtr, unsigned short frameSize)
```

### Arguments

`short *framePtr`                      Pointer to one frame of PCM speech data<sup>1</sup>  
`unsigned short frameSize`      Size of speech data (in words)

**Note 1:** Set sampling to `MIC_SAMPLING_TYPE_SIGNED_12BIT_FILTER_OFF` for the DS.

### Return Values

<code>ASR_NO_ERROR</code>	Call successful, No recognition result
<code>ASR_RECOG_STARTED</code>	Call successful, no recognition result, <b>speech input detected (processing in progress)</b>
1 to 10	Call successful, <b>Recognition result</b>
<code>ASR_ERROR_INTERNAL</code>	Internal error occurred (including calls before initialization and illegal arguments)
<code>ASR_ERROR_OFFTIMEOUT</code>	Silence timeout
<code>ASR_ERROR_ONTIMEOUT</code>	Speech timeout
<code>ASR_ERROR_TOO_LOUD</code>	Initial sound too loud
<code>ASR_ERROR_TOO_SOON</code>	Speech began too early

### Description

This function obtains speech frame data (frame size varies<sup>1</sup>: 128 points or greater) and performs the recognition process. When `ASR_RECOG_STARTED` is returned, there is microphone input and speech recognition processing is in progress. If a value between one and ten is returned, recognition has completed and the number of recognition results equals the returned value. Obtain the results using the API described below.

**Note 1:** If one frame is 16.7 msec, then 183 points (366 bytes) of sampling data is passed.

### Comments

When this API is called with the second argument (`frameSize`) set to zero, the recognition result can be obtained without waiting for the end-of-speech detection. (Normally, end-of-speech detection occurs automatically when the speech level declines over 3 to 500 msec.)

This feature can be used to input a zero-size frame when the user presses a button so the recognition process ends without waiting to determine whether speech has ended.

### 3.3.9 Recognition Result Acquisition (Recognition Number)

#### Declaration

```
short ASR_GetCandidateWord(short order)
```

#### Arguments

short order                      Specifies the recognition candidate ranking (1 – 10)

#### Return Values

ASR_ERROR_INTERNAL	No recognition result or invalid order value
ASR_ERROR_PARAM	Recognition rejected (did not match with the registration dictionary entry due to noise input, etc.)
1 to 9999	The recognition result number. The number is set in the library.

#### Description

This function obtains the recognition result (word number) for the candidate ranking specified in the argument after the `ASR_RecogFrame()` function has returned with a value of one or greater. The function returns `ASR_ERROR_INTERNAL` if the argument's candidate ranking value exceeds the number of recognition candidates set by `ASR_SetParam` or if it exceeds the number of candidates returned by `ASR_RecogFrame`. If accidental external noise (such as throat-clearing) was input, `ASR_ERROR_PARAM` may be returned for recognition rejection.

### 3.3.10 Recognition Result Acquisition (Recognition Score)

#### Declaration

```
short ASR_GetCandidateScore(short order)
```

#### Arguments

short order                      Specifies the recognition candidate ranking (1 – 10)

#### Return Values

-32000	No recognition result or invalid order value
Some value larger than -32000	Recognition score

#### Description

This function obtains the recognition result (recognition score) for the candidate ranking specified in the argument after the `ASR_RecogFrame()` function has returned with a value of one or greater. The function returns -32000 if the argument's candidate ranking value exceeds the number of recognition candidates set by `ASR_SetParam` or if it exceeds the number of candidates returned by `ASR_RecogFrame`.

#### Comments

The recognition score can take a negative value according to the sound conditions.

## 3.4 The Middleware API

---

The Middleware API uses features unique to the DS platform, such as microphone control and thread creation, and runs the Core Engine API internally. See the Appendix and the *Nintendo DS Speech Recognition Library Programming Guide* to learn about the use of each API layer and the process flow in the application.

### 3.4.1 Microphone Initialization

---

#### Declaration

```
short ASR_MICInit(ASRCallback callback, short *addr)
```

#### Arguments

ASRCallback callback	Pointer to the callback function
short *addr	Pointer to the argument passed to the callback

#### Return Values

ASR_NO_ERROR	Successful (application enters callback wait state)
ASR_ERROR_PARAM	Illegal argument error
ASR_ERROR_INTERNAL	Called again before calling ASR_MICEnd
ASR_ERROR_DS_MIC	Microphone error (e.g., power operation, gain cannot be set)

#### Description

This function turns microphone power ON and sets the gain for speech recognition. The microphone state when calling the function is stored, and the state is reinstated with ASR\_MICEnd.

Although this function returns immediately, if microphone power is OFF when calling this function, the callback function is called after 500 msec to allow microphone characteristics to stabilize. (If microphone power is ON, the callback function is called immediately.)

This function returns immediately, so if the microphone power is OFF when the function is called, let 500ms pass and call a callback function in order to wait for the microphone features to stabilize. (If the microphone power is ON when the function is called, then call the callback function immediately).

**Callback**

1st argument	2nd argument (value of the <code>short*</code> pointer destination)	Description
<code>ASR_RESULT_MICINIT_COMPLETED</code>	0	Microphone initialization successful (speech can be input)

When an error occurs due to the microphone settings, the callback function cannot be called. The application should verify the callback before moving to the next process (e.g., initializing the middleware or beginning the recognition process).

The memory region specified in the second argument must be valid when calling the callback function. (Do not use to exceed the scope or to specify automatic variables.)

**3.4.2 Microphone Termination Process****Declaration**

```
short ASR_MICEnd(void)
```

**Arguments**

None.

**Return Values**

<code>ASR_NO_ERROR</code>	Successful
<code>ASR_ERROR_INTERNAL</code>	Internal error (e.g., called before initialization)
<code>ASR_ERROR_DS_MIC</code>	Microphone error (e.g., power operation, gain cannot be set)

**Description**

This function returns the microphone to the state it was in when `ASR_MICInit` was called. This function should be called when speech recognition is no longer being used (i.e., after instruction to terminate the middleware is issued).

**Comments**

This function should be used with `ASR_MICInit`. An error is returned if this function is called successively.

### 3.4.3 Middleware Initialization

---

#### Declaration

```
short ASR_MWInit (ASR_Language language, ASR_DataItemType *data,  
                 short itemCount, short *micBuffer, short *coreMem)
```

#### Arguments

ASR_Language language	Specifies the recognition language
ASR_DataItemType *data	Pointer to the language-dependent data group
short itemCount	The number of sets of the language-dependent data
short *micBuffer	The microphone sampling data buffer (32-byte aligned memory of ASR_MIC_BUFFER_SIZE size)
short *coreMem	Pointer to work memory used by the recognition engine <sup>1</sup>

**Note:** For details about these arguments, see [3.2 - Data Structure Definitions](#)

**Note 1:** A memory region of the size obtained with `ASR_GetWorkMemSize` is secured by the application, and the address of that memory region is passed.

#### Return Values

ASR_NO_ERROR	Successful
ASR_ERROR_PARAM	Invalid argument error
ASR_ERROR_INTERNAL	Called again before terminating middleware

#### Description

This function changes the recognition language and initializes the middleware and the recognition engine.

#### Comments

Be careful not to disable the microphone sampling buffer or the recognition engine's work memory until terminating the middleware. Also, operations are not guaranteed if the size of the secured memory region is incorrect.

### 3.4.4 Middleware Termination Process

#### Declaration

```
short ASR_MWEnd(void *data)
```

#### Arguments

`void *data` Pointer to the buffer storing environmental history data (100 bytes)

#### Return Values

Positive value greater than 0	Size of the loaded environmental history data
<code>ASR_ERROR_INTERNAL</code>	Internal error (e.g., called before initialization)

#### Description

This function discards the data related to the middleware and the recognition engine. When there is environmental history data that can improve recognition accuracy, that data is written to the second argument address, and the size of the data is returned. When the argument is NULL, the statistical data is not restored, but the termination process is performed.

#### Comments

The environmental history data is used to store some characteristics of the speech data passed to the recognition engine and to improve recognition accuracy. When data can be stored in nonvolatile memory, recognition accuracy can be improved by passing the data to the recognition engine the next time the engine is initialized.

After this API has ended, the work memory passed during middleware initialization can be discarded.

### 3.4.5 Recognition Process

#### Declaration

```
short ASR_MWRecogStart(unsigned char *dict, short word_count,
                        ASR_ParamType *param, ASR_ResultType *result, ASRCallback
                        callback)
```

#### Arguments

<code>unsigned char *dict</code>	Pointer to the recognition dictionary
<code>short word_count</code>	Number of words in dictionary
<code>ASR_ParamType *param</code>	Pointer to the configuration parameter
<code>ASR_ResultType *result</code>	Pointer to the buffer storing the recognition results
<code>ASRCallback callback</code>	Pointer to the callback function

**Note:** For details about these arguments, see [3.2 - Data Structure Definitions](#).



## Return Values

ASR_NO_ERROR	Successful (application enters callback wait state)
ASR_ERROR_PARAM	Invalid argument error
ASR_ERROR_INTERNAL	Called again before recognition completed
ASR_ERROR_DIC_TOO_LARGE	Too many registered words (more than 100)
ASR_ERROR_DIC_LONG_SPEL	Exceeds length for registered words
ASR_ERROR_DIC_MISSPEL	Word misspelled
ASR_ERROR_DIC_CONV	Word conversion internal error (e.g., when the languages are not the same)

## Description

This function performs the processes from the configuration of the recognition engine (recognition dictionary and parameters settings) to the recognition process in a single pass. This function calls the callback function when recognition starts, terminates, or fails, as well as sets the value in the structure that stores the recognition result.

## Callback

1st argument	2nd argument (Value of result->result)	Description
ASR_RESULT_RECOG_STARTED	0	Indicates that there is microphone input and that recognition processing has started
ASR_RESULT_RECOG_COMPLETED	1 or more (number of recognition candidates)	Sets data for that number of recognition candidates in the recognition result buffer.
ASR_RESULT_RECOG_FAILED	ASR_ERROR_OFFTIMEOUT	Silence timeout
	ASR_ERROR_ONTIMEOUT	Maximum duration of speech input timeout
	ASR_ERROR_TOO_LOUD	Initial sound too loud
	ASR_ERROR_TOO_SOON	Speech began too early
	ASR_ERROR_DS_MIC	Microphone error (sampling impossible)

## Comments

Although this API returns soon after starting, the recognition process continues to execute in a separate thread until either a timeout or some other error arises, or the start and end of a speech is detected and the recognition result is obtained. Notification of the result is made by calling the callback function.

As ASR\_RESULT\_RECOG\_STARTED indicates that there is microphone input and that recognition processing has started, the application can use an icon to indicate to the user that recognition processing is in progress.

After this function executes, have the application perform procedures according to the recognition

result that was obtained when the callback function is called by `ASR_RESULT_RECOG_COMPLETED` (or `ASR_RESULT_RECOG_FAILED`).

This API cannot be called before initializing the middleware or during the recognition process (while in the callback wait state). Attempting to do so will return an error.

**Tips:**

This API takes considerable processing time to set the recognition dictionary<sup>1</sup>, but the processing time can be distributed using the following method:

1. After initializing the middleware, set the recognition dictionary using the `ASR_SetDict` function of the core API.
2. Set the first argument of `ASR_MWRecogStart` to a null pointer. (The second argument can be undefined.)

You can distribute the considerable processing time required to set the dictionary by performing step 1. (above) when there is a slack period, and then perform step 2. when you want to start recognition processing. If the words for recognition do not change, you can perform recognition by repeating step 2. However, if you set the pointer to NULL when no recognition dictionary has been set, it will result in an error.

If you set the parameters in advance using `ASR_SetParam` in the core API, you can repeat recognition using the same parameters by setting the third argument of `ASR_MWRecogStart` to a null pointer.

---

### 3.4.6 Recognition Abort Process

---

**Declaration**

```
void ASR_MWRecogStop(void)
```

**Arguments**

None.

**Return Values**

None.

**Description**

This function aborts the recognition process started by `ASR_MWRecogStart`. After this function is called, the recognition engine halts the recognition process and microphone sampling. If there is a recognition result at this time, the result is returned via the callback function. If there is no result, the callback function is called with `ASR_ERROR_OFFTIMEOUT`.

---

<sup>1</sup> Depending on the number of stored words, this will require from several milliseconds to several tens of milliseconds to process.

### 3.4.7 Setting the Priority of the Recognition Library Thread

---

#### Declaration

```
void ASR_MWSetThreadPriority (u32 prio);
```

#### Arguments

u32 prio      Priority value (0 to 16: However, it must be lower than the main thread priority)

#### Return value

None.

#### Description

Changes the priority (the default is 8) of the thread that the recognition library starts for recognition processing.

#### Comments

**There is no guarantee that this function will operate correctly if the priority is set below that of the main thread (the game thread).** This function is used to set the priority of the recognition library thread to a higher level (a lower value) than the main thread when the main thread is less than 8. The priority setting becomes valid when the `ASR_MWRecogStart` function is called next.

### 3.4.8 Obtaining the Priority of the Recognition Library Thread

---

#### Declaration

```
u32 ASR_MWGetThreadPriority(void);
```

#### Arguments

None.

#### Return Value

The priority of the recognition library thread.

#### Description

Returns the current priority of the recognition library thread.

#### Comments

The default value immediately after the `ASR_MWInit` function is called is `ASR_DEFAULT_THREAD_PRIORITY (=8)`.

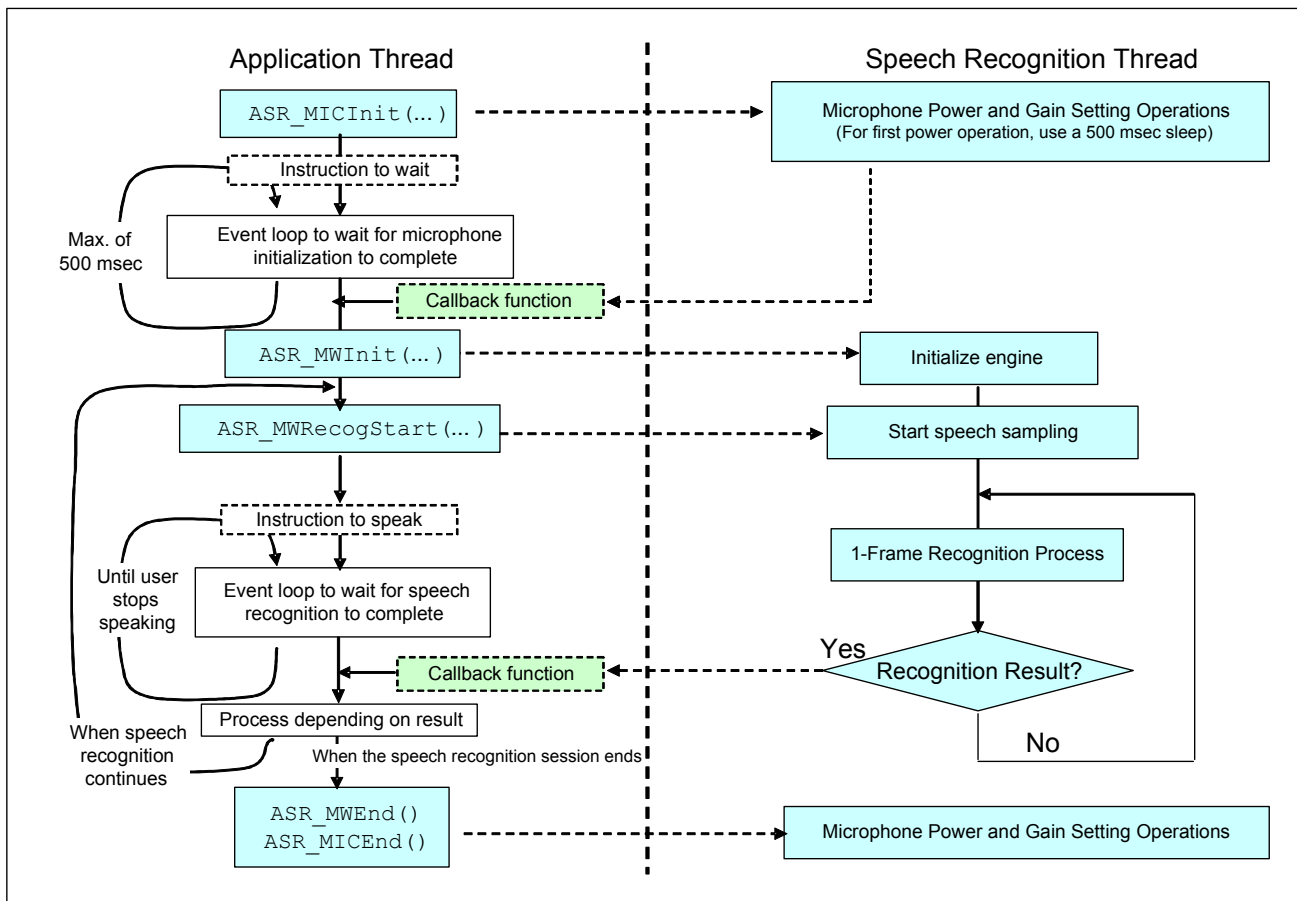
## 4 Appendix

This appendix provides a simple explanation of the process flow for the middleware API. See the *Nintendo DS Speech Recognition Library Programming Guide* for details about the process and precautions, as well as information about the core engine API and useful hints.

### 4.1 Process Flow when Using Middleware API

When using the middleware API, all microphone controls, speech sampling, and speech recognition process are performed in separate threads in the middleware. Consequently, the application can conduct other processes while the recognition process is running. Notification of events, such as when microphone or when speech recognition finishes, are sent to the application via [callback functions](#).

**Figure 4-1 The Speech Recognition Middleware API Process Flowchart**



## 4.2 Callback Functions

---

Do not use the callback functions to perform processes that take time. As shown in the example below, design the callback function to set only flags and error codes, and then check the flag states periodically in the main event loop and implement the appropriate process.

```
volatile short GameASRFlag; // Flag referenced by game thread
volatile short GameASRError; // Error code referenced by game thread

void Callback(ASR_RESULT_CODE result, void *addr)
{
    ASR_ResultType *asrResult;

    GameASRFlag = (short)result;
    switch (result) {
        case ASR_RESULT_MICINIT_COMPLETED:
            GameASRError = (short)(*addr);
            break;

        /* The following is unnecessary if the game thread can reference
           the recognition results buffer specified by ASR_MWRecogStart */
        case ASR_RESULT_RECOG_STARTED:
        case ASR_RESULT_RECOG_COMPLETED:
        case ASR_RESULT_RECOG_FAILED:
            asrResult = (ASR_ResultType *)addr;
            GameASRError = asrResult->result;
            /* In some cases, copies the recognition results */
            break;
        default:
            break;
    }
}
```

The accurate timing of vocalization is important for accurate speech recognition. (If speech begins too soon, it cannot be recognized.) Use a GUI or some other means to make appropriate use of the results from the callback function to clearly alert the user when to speak and when not to speak. In addition, prevent interference from noise input by taking measures such as turning off the speakers while the user is speaking.

## 4.3 Phonetic Alphabet Transcription

Versions 1.0.4 and later of the Speech Recognition library will accept a subset of phonetic alphabet transcriptions as dictionary registrants as defined by SAMPA<sup>2</sup>. Registration is limited to only those phonetic transcriptions that correspond to the language that is configured when the recognition engine is initialized. Attempts to register any other phonetic transcription will generate an error. The tables in the following sections list the symbols and transcriptions that can be registered for each language.

Here are several examples of recognition dictionaries that can be registered with the `ASR_SetDict` or the `ASR_MWRecogStart` function. Phonetic transcriptions are text strings preceded by a pound sign (#).

### 1. Example of registering [1:hello 2:world END] using only the phonetic alphabet transcription

```
- ASR_US_ENGLISH:      "1:#hElO 1:#h@lO 2:#w3`ld END"
- ASR_BRITISH:         "1:#h@l@U 1:#hel@U 2:#w3:ld END"
```

In the above examples, #h@lO is the British pronunciation and #hel@u is the American pronunciation. Here is an example of how they can be used as sub-dictionaries—words registered with the same meaning but with different pronunciations.

### 2. An example registering two phonetic transcriptions using word notation. (In this example, the language has been specified as `ASR_US_ENGLISH`.)

```
"1:hello 1:#hElO 1:#h@lO 2:world 2:#w3`ld END"
```

In this example, the word notation and the phonetic notation yield the same result. But by registering them together, speech recognition has increased precision in situations where the game character name does not follow standard rules for notation and the pronunciation of words varies depending on the dialect.

### 4.3.1 US English

#### Consonants

Symbol	Word	Transcription
<b>p</b>	pin	pIn
<b>b</b>	bin	bIn
<b>t</b>	tin	tIn
<b>d</b>	din	dIn
<b>k</b>	kin	kIn
<b>g</b>	give	gIv
<b>tS</b>	chin	tSIn
<b>dZ</b>	gin	dZIn
<b>f</b>	fin	fIn
<b>v</b>	vim	vIm
<b>T</b>	thin	TIn

<sup>2</sup> The Speech Assessment Methods Phonetic Alphabet (SAMPA) is a type of computer-readable phonetic notation that is based on the International Phonetic Alphabet (IPA) and uses the 7-bit printable ASCII characters.

<b>D</b>	this	DIs
<b>s</b>	sin	sIn
<b>z</b>	zing	zIN
<b>S</b>	shin	SIn
<b>Z</b>	measure	mEZ3`
<b>h</b>	hit	hIt
<b>m</b>	mock	mAk
<b>n</b>	knock	nAk
<b>N</b>	thing	TIN
<b>r</b>	wrong	rON
<b>l</b>	long	lON
<b>w</b>	wasp	wAsp
<b>j</b>	yacht	jAt

### Vowels

Symbol	Word	Transcription
<b>I</b>	pit	pIt
<b>E</b>	pet	pEt
<b>{</b>	pat	p{t
<b>A</b>	pot	pAt
<b>v</b>	cut	kVt
<b>U</b>	put	pUt
<b>i</b>	ease	iz
<b>e</b>	raise	rez
<b>u</b>	lose	luz
<b>o</b>	nose	noz
<b>O</b>	cause	kOz
<b>aI</b>	rise	raIz
<b>OI</b>	noise	nOIz
<b>aU</b>	rouse	raUz
<b>3`</b>	furs	f3`z
<b>@</b>	allow	@laU

## 4.3.2 British ---

### Consonants

Symbol	Word	Transcription
<b>p</b>	pin	pIn
<b>b</b>	bin	bIn
<b>t</b>	tin	tIn
<b>d</b>	din	dIn
<b>k</b>	kin	kIn
<b>g</b>	give	gIv
<b>tS</b>	chin	tSIn
<b>dZ</b>	gin	dZIn
<b>f</b>	fin	fIn
<b>v</b>	vim	vIm
<b>T</b>	thin	TIn

<b>D</b>	this	DIs
<b>s</b>	sin	sIn
<b>z</b>	zing	zIN
<b>S</b>	shin	SIn
<b>Z</b>	measure	mEZ@
<b>h</b>	hit	hIt
<b>m</b>	mock	mQk
<b>n</b>	knock	nQk
<b>N</b>	thing	TIN
<b>r</b>	wrong	rQN
<b>l</b>	long	lQN
<b>w</b>	wasp	wQsp
<b>j</b>	yacht	jQt

### Vowels

Symbol	Word	Transcription
<b>I</b>	pit	pIt
<b>e</b>	pet	pet
<b>{</b>	pat	p{t
<b>Q</b>	pot	pQt
<b>v</b>	cut	kVt
<b>U</b>	put	pUt
<b>@</b>	another	@nVD@
<b>i:</b>	ease	i:z
<b>eI</b>	raise	reIz
<b>aI</b>	rise	raIz
<b>OI</b>	noise	nOIz
<b>u:</b>	lose	lu:z
<b>@U</b>	nose	n@Uz
<b>aU</b>	rouse	raUz
<b>3:</b>	furs	f3:z
<b>A:</b>	stars	stA:z
<b>O:</b>	cause	kO:z
<b>I@</b>	fears	fI@z
<b>U@</b>	cures	kjU@z

## 4.3.3 French

### Consonants

Symbol	Word	Transcription
<b>p</b>	pont	po~
<b>b</b>	bon	bo~
<b>t</b>	temps	ta~
<b>d</b>	dans	da~
<b>k</b>	quand	ka~
<b>g</b>	gant	ga~
<b>f</b>	femme	fam
<b>v</b>	vent	va~
<b>s</b>	sans	sa~
<b>z</b>	zone	zon



<b>S</b>	champ	Sa~
<b>Z</b>	gens	Za~
<b>m</b>	mont	mo~
<b>n</b>	nom	no~
<b>J</b>	oignon	oJo~
<b>l</b>	long	lo~
<b>R</b>	rond	Ro~
<b>w</b>	coin	kwe~
<b>H</b>	juin	ZHe~
<b>j</b>	pierre	pjER

### Vowels

Symbol	Word	Transcription
<b>i</b>	si	si
<b>e</b>	ses	se
<b>E</b>	seize	sEz
<b>a</b>	patte	pat
<b>O</b>	comme	kOm
<b>o</b>	gros	gRo
<b>u</b>	doux	du
<b>y</b>	du	dy
<b>9</b>	neuf	n9f
<b>@</b>	justement	Zyst@ma~
<b>e~</b>	vin	ve~
<b>a~</b>	vent	va~
<b>o~</b>	bon	bo~

## 4.3.4 Spanish

---

### Consonants

Symbol	Word	Transcription
<b>p</b>	padre	padre
<b>b</b>	vino	bino
<b>t</b>	tomo	tomo
<b>d</b>	donde	donde
<b>k</b>	casa	kasa
<b>g</b>	gata	gata
<b>tS</b>	mucho	mutSo
<b>f</b>	fácil	faTil
<b>T</b>	cinco	Tinko
<b>s</b>	sala	sala
<b>x</b>	mujer	muxer
<b>m</b>	mismo	mismo
<b>n</b>	nunca	nunka
<b>J</b>	año	aJo
<b>l</b>	lejos	lexos
<b>r</b>	puro	puro
<b>rr</b>	torre	torre

## Vowels

Symbol	Word	Transcription
j	rei	rrej
i	pico	piko
e	pero	pero
a	valle	baje
o	toro	toro
u	duro	duro

## 4.3.5 German

## Consonants

Symbol	Word	Transcription
p	Pein	paIn
b	Bein	baIn
t	Teich	taIx
d	Deich	daIx
k	Kunst	kUnst
g	Gunst	gUnst
pf	Pfahl	pfa:l
ts	Zahl	tsa:l
tS	deutsch	dOYtS
dZ	Dschungel	dZUNg@l
f	fast	fast
v	was	vas
s	Tasse	tas@
z	Hase	ha:z@
S	waschen	vaS@n
Z	Genie	Zeni:
j	Jahr	ja:R
x	Buch	bu:x
h	Hand	hant
m	mein	main
n	nein	naIn
N	Ding	dIN
l	Leim	laIm
R	Reim	RaIm

## Vowels

Symbol	Word	Transcription
I	Sitz	zIts
E	Gesetz	g@zEts
a	Satz	zats
O	Trotz	tROts
U	Schutz	SUts
Y	hübsch	hYpS
9	plötzlich	pl9tslIx
i:	Lied	li:t
e:	Beet	be:t

<b>a:</b>	Tat	ta:t
<b>o:</b>	rot	Ro:t
<b>u:</b>	Blut	blu:t
<b>y:</b>	süß	zy:s
<b>2:</b>	blöd	bl2:t
<b>aI</b>	Eis	aIs
<b>aU</b>	Haus	haUs
<b>OY</b>	Kreuz	kROYts
<b>@</b>	bitte	bIt@

### Vowels

Symbol	Word	Transcription
<b>i</b>	mite	mite
<b>e</b>	rete	rete
<b>E</b>	meta	mEta
<b>a</b>	rata	rata
<b>O</b>	moto	mOto
<b>o</b>	dove	dove
<b>u</b>	muto	muto

## 4.3.6 Italian

---

### Consonants

Symbol	Word	Transcription
<b>p</b>	pane	pane
<b>b</b>	banco	banko
<b>t</b>	tana	tana
<b>d</b>	danno	danno
<b>k</b>	cane	kane
<b>g</b>	gamba	gamba
<b>pp</b>	coppa	kOppa
<b>bb</b>	gobba	gObba
<b>tt</b>	zitto	tsitto
<b>dd</b>	cadde	kadde
<b>kk</b>	nocca	nOkka
<b>gg</b>	fugga	fugga
<b>ts</b>	zitto	tsitto
<b>dz</b>	zona	dzOna
<b>tS</b>	cena	tSena
<b>dZ</b>	gita	dZita
<b>tts</b>	bozza	bOttsa
<b>ddz</b>	mezzo	mEddzo
<b>ttS</b>	braccio	brattSo
<b>ddZ</b>	oggi	OddZi
<b>f</b>	fame	fame
<b>v</b>	vano	vano
<b>s</b>	sano	sano
<b>z</b>	sbaglio	zbaLLo

<b>S</b>	scendo	Sendo
<b>ff</b>	beffa	bEffa
<b>vv</b>	bevvi	bevvi
<b>ss</b>	cassa	kassa
<b>SS</b>	ascia	aSSa
<b>m</b>	molla	mOlla
<b>n</b>	nocca	nOkka
<b>J</b>	gnocco	JOkko
<b>mm</b>	grammo	grammo
<b>nn</b>	panna	panna
<b>JJ</b>	bagno	baJJo
<b>r</b>	rete	rete
<b>l</b>	lama	lama
<b>rr</b>	ferro	fErro
<b>ll</b>	colla	kOlla
<b>LL</b>	foglia	fOLLa
<b>j</b>	ieri	jEri
<b>w</b>	uomo	wOmo

### Vowels

Symbol	Word	Transcription
<b>i</b>	mite	mite
<b>e</b>	rete	rete
<b>E</b>	meta	mEta
<b>a</b>	rata	rata
<b>O</b>	moto	mOto
<b>o</b>	dove	dove
<b>u</b>	muto	muto

Company names and product names are the trademark or registered trademark of the respective companies.

© 2005 Nintendo

The contents of this document cannot be duplicated, copied, reprinted, transferred, distributed or loaned in whole or in part without the prior approval of Nintendo.