SVOX AG
Baslerstrasse 30
CH-8048 Zürich

phone +41 43 544 06 00
fax +41 43 544 06 01
www.svox.com

SVOX Pico

Speech Output Engine SDK

1.0.0

April 20, 2009

Table of Contents

1 Introduction

SVOX, a European leader for speech technology delivers the enclosed SVOX Pico **software development kit** (**SDK**) intended to be integrated in larger speech applications. SVOX differentiates itself by offering customized speech output solutions with superior sound quality and performance. With SVOX's software architecture customers are offered a speech output engine adaptable to their technical and market needs.

SVOX Pico is a new light-weight **Text-to-Speech** (**TTS**) solution based on Hidden-Markov-Models. It is designed for integration into mobile phones and other mobile devices. Complementing the SVOX Pico SDK s, the SVOX Pico SDK has a new lean API supporting rapid integration and giving full control over the TTS process.

Apart from its very small footprint, SVOX Pico architecture allows yielding using a polling mechanism and enables lingware downloads by using composition of lingware resources at runtime.

The basic SVOX Pico system is platform-independent. Most of the SDK documentation also holds for all supported platforms. Parts that are specific to a product or platform will be marked as such.

The SVOX Pico SDK contains all the necessary libraries, data files, header files, tools, and documentation to integrate the SVOX Pico speech output engine into your application. It also contains sample source code files that demonstrate SVOX Pico API usage. Furthermore, the SDK contains a test program binary with a simple command line. This test program is primarily intended for testing and evaluation purposes. In real-time applications and as a component of larger speech applications, SVOX Pico should be used with the SVOX Pico API of the SDK or via one of the additional standardized APIs provided (platform-dependent).

In addition to the C header files and example source code, the SVOX Pico SDK documentation includes this SVOX Manual. It contains a description of the SVOX Pico SDK installation (in the Appendix C), tools contained in the SDK, and how speech output and the text-to-speech conversion process in SVOX Pico can be controlled by the use of markup tags.

The functionality and content of SDKs delivered within customer-specific projects can vary from the pre-packaged SDKs and the descriptions contained in this manual. Please consult the project documentation for additional information.

2 Pico Test Program

2.1 Introduction

The SVOX Pico SDK contains a test program binary with a simple command line interface. This program needs the support of a console interface from the operating system, and its direct use is possible only on Windows and Unix platforms.

This program is primarily intended for testing and evaluation purposes. Its usage is described in this section.

2.2 Running the Command Line Interface Program

2.2.1 Command Syntax

The general PICOSH command syntax is

```
picosh     [<options>]                                  (interactive mode)
picosh     [<options>] <txtfile> [<sndfile>]            (TTS mode)
picosh -b  [<options>] <cmdfile> [<sndfile>]            (batch file mode)
```

with the following command line options:

```
-d <datapath>
-v <voice>
-h
-V
```

The interactive mode is invoked by simply running the PICOSH command line interface program with just the voice parameter, e.g.:

```
picosh –v de-DE_gl0
```

Currently, the PICOSH program is shipped with a British English voice “en-GB_kh0” and a German voice “de-DE_gl0”.

Running PICOSH without specifying a voice will default to the British English voice.

In TTS mode, this command synthesizes the input text file `<txtfile>` to an output sound file `<sndfile>`. If `<sndfile>` is left empty and direct audio output is supported on your platform, this command synthesizes `<txtfile>` directly to the audio device of the machine where the program is running.

In batch file mode a list of interactive mode commands and texts can be processed which are read from a batch file `<cmdfile>`. After processing the commands and synthesizing the texts the system exits.

2.2.2 Command Line Options and Files

`-d <datapath>` `<datapath>` provides the system data path where the files of the run-time resources (e.g. en-GB_hk0_ta.bin) reside. By default, these files are searched for in the current default directory.

WARNING: <datapath> must include a terminating \ (e.g. Windows) or a terminating / (e.g. Unix), e.g., S:\svox\ or /usr/local/svox/, not only S:\svox or /usr/local/svox.

-v <voice>	<voice> indicates the name of a voice to be used. The voice also defines the language. The availability of voices and languages depends on the individually installed Pico lingware packages.
-h	Displays a summary of the command syntax, options, and operation modes, and then exits.
-V	Displays version information and exits.
<txtfile>	<txtfile> is the name of a text file.
<sndfile>	Output sound file. The format of the output file is the Windows WAVE format . If <sndfile> is omitted, the output is sent to the direct acoustic output device of the machine where picosh is running (platform-dependent).
<cmdfile>	Input batch file. A batch file is a text file that contains a list of strings and interactive mode commands that could also manually be entered in interactive mode. In the batch file mode, the contents of the batch file are read and the strings and commands processed without any command line interaction.

2.2.3 Examples of Usage

Interactive mode:

If the PICOSH command line interface program is started without the <txtfile> and <sndfile> parameters, the interactive mode is invoked. In the interactive mode, the prompt message "picosh-*voice*> " repeatedly asks the user to enter a string to be synthesized or one of the interactive mode commands. The command

```
\help
```

displays all available commands together with a short description. The basic commands are described in the following.

When a text string is entered, it is synthesized directly to the acoustic output device of the machine where the program is running (platform-dependent). A string may contain several sentences.

```
\exit
```

terminates the interactive synthesis.

Text-to-speech mode:

```
picosh –v de-DE_gl0 test.txt test.wav
```

Synthesizes the text file test.txt to the wav file test.wav. In this example, the Pico run-time resource files for the selected language (de-DE*.bin) must reside in the current default directory.

```
picosh –d S:\pico\ test.txt
```

Synthesizes the text file test.txt to the machine audio device (platform-dependent). In this example (on a Windows system using a DOS command prompt), the Pico run-time resource

files for the default language are searched for in the directory `S:\pico\` and the lingware files (the default files “en-GB_ta.bin” and “en-GB_kh0_sg.bin”) are loaded.

Batch file mode:

Creating a text file `batch.txt` containing for example the lines

```
\help
Hello.
```

and then running

```
picosh  -b batch.txt
```

will result in the Pico system starting up, showing the help information on the screen, synthesizing the word "hello", and then exiting.

3 SVOX Pico Application Programming Interface

3.1 Introduction

SVOX Pico offers a proprietary application programming interface (API) that enables speech applications to include SVOX Pico as the speech output component. The SVOX Pico API is defined in the C header file '*picoapi.h*' *and* '*picodefs.h*' which must be included in the application program using the SVOX Pico API. The SVOX Pico API is identical on all supported platforms. The SVOX Pico software itself is provided as dynamic or static link library (e.g. libpico.dll on Windows platforms, libpico.a on Unix platforms).

The SVOX Pico API C header file '*picoapi.h*' contains the documentation of all available functions and their usage. It is distributed together with this manual and as part of all pre-packaged SVOX Pico SDKs. The API delivered within a customer-specific project can vary from the descriptions contained in the header file and in this manual. Please consult the project documentation for additional information.

The following sections are intended to provide a general overview of the basic concepts and the functionality of the SVOX Pico API. Implementation details of the individual functions are given in the header file '*picoapi.h*'.

3.2 Basic Concepts

3.2.1 System, Engine, Voice Definitions and Resources

The SVOX Pico API deals with four different kinds of entities: system, engine, voice definitions and lingware. The API functions managing these entities are organized in two levels, the system and the engine-level.

SVOX Pico Lingware Resource

The term 'lingware' denotes all the language- and speaker-dependent knowledge bases needed to do TTS synthesis. A **resource** is a named collection of such lingware knowledge bases.

Typically, the set of knowledge bases needed for an entire TTS 'voice' are distributed into two separate resources: one containing the speaker-independent ('language') and one containing the speaker-dependent ('speaker') data.

SVOX Pico Voice Definition

A **voice definition** defines a *set of resources* needed to perform synthesis for one 'voice' (language/speaker combination), and maps that set to a *voice name*.

SVOX Pico System

The **system** contains and manages all globally used resources, i.e., it performs loading and unloading of lingware, and creation and deletion of TTS engines. There is only a single, monolithic system block. All API functions on the Pico system level take a `pico_System` handle as the first parameter.

SVOX Pico Engine

An **engine** provides the functions needed to perform actual synthesis. Currently there can be only one engine instance at a time (concurrent engines will be possible in the future). Engines operate in parallel to the system level. (Note, however, that engines must be created and closed via the overall system.) Engines are linked to one voice definition via a voice name. All API functions at the engine-level take a `pico_Engine` handle as the first parameter.

3.2.2 Basic Usage

In its most basic form, an application must call the following functions in order to perform TTS synthesis:

- pico_initialize
- pico_loadResource
- pico_createVoiceDefinition
- pico_addResourceToVoiceDefinition
- pico_newEngine
- pico_putTextUtf8
- pico_getData (several times)
- pico_disposeEngine
- pico_releaseVoiceDefinition
- pico_unloadResource
- pico_terminate

It is possible to repeatedly run the above sequence, i.e., the SVOX Pico system may be initialized and terminated multiple times. This may be useful in applications that need TTS functionality only from time to time.

3.2.3 A First TTS Example

The simple example below demonstrates the basic usage of the SVOX Pico API for doing TTS. The program initializes the system, defines a voice, loads resources, starts an engine for the defined voice, then synthesizes the contents of the '`input`' buffer, and finally closes the engine and terminates the system.

`pico_initialize` initializes the SVOX Pico system and returns the system handle.

`pico_createVoiceDefinition` and `pico_addResourceToVoiceDefinition` introduces to the system a voice name and links this voice name to a set of resource identifiers.

`pico_loadResource` loads the lingware needed to do TTS from a platform-dependent (BIN) file. All resources linked to a voice definition have to be loaded before an engine for that voice can be started.

`pico_newEngine` creates and starts up a new SVOX Pico engine and returns its handle. The set of lingware resources to be used is given by the name of the previously created voice definition.

`pico_putTextUtf8` sends new text to the engine to be synthesized. The function accepts Unicode strings encoded in UTF-8. Sending a NULL character will flush the engine, i.e. tell the engine to process all the text up to the flush regardless of what text might come next.


```
…
memory = malloc(memorySize);
status = pico_initialize(memory,memorySize,&system);
…
myVoiceName = (pico_Char *) "susanne";
status = pico_createVoiceDefinition(system, myVoiceName);
status = pico_addResourceToVoiceDefinition(system, myVoiceName,
                RESOURCE_NAME_DE_SI);
status = pico_addResourceToVoiceDefinition(system, myVoiceName,
                RESOURCE_NAME_DE_SD);
…
status = pico_loadResource(system, siLingFile, &siResource);
status = pico_loadResource(system, sdLingFile, &sdResource);
…
status = pico_newEngine(system,myVoiceName,&engine);
…
textRemaining = strlen(input) + 1; /* includes terminating '\0'*/
inp = (pico_Char *)input;
while (textRemaining) {

     status = pico_putTextUtf8(engine, inp, textRemaining, &bytesSent);
     textRemaining -= bytesSent;
     inp += bytesSent;

     do {
          status = pico_getData(engine, (void *) outBuffer,
          MAX_OUTBUF_SIZE - 1, &bytesReceived, &outDataType);
          if (bytesReceived) {
               /* write to file or forward to audio device */
               printf("received %d bytes\n", bytesReceived);
          }
     } while (PICO_STEP_BUSY == status);
}
…
status = pico_disposeEngine(system,&engine);
…
status = pico_unloadResource(system,&siResource);
status = pico_unloadResource(system,&sdResource);
…
status = pico_terminate(&system);
…
```

`pico_getData` allows the engine to do one TTS processing "step", i.e. a small portion of TTS processing, and returns the processing state. If speech audio data is produced during this step, this data is returned and `bytesReceived` is set to the number of bytes returned. A processing state value of `PICO_STEP_BUSY` means that the engine has still more processing to be done.

`pico_disposeEngine` shuts down the TTS engine and frees memory occupied by the engine.

`pico_unloadResource` removes previously loaded lingware resources.

`pico_terminate` terminates the SVOX Pico system and frees all remaining memory.

It is possible to repeatedly run the above sequence, i.e., the SVOX Pico system may be initialized and terminated several times. This may be useful in applications that need the TTS functionality (and resources) only from time to time.

WARNING: It is not possible, and leads to system failure, if `pico_initialize` is called twice without an intervening `pico_terminate`. Before calling `SVOX pico_terminate`, it is important to close all engines.

3.2.4 SVOX Pico Internal Memory Management

SVOX Pico never dynamically allocates memory on its own (e.g. by using `malloc`). Instead, when initializing the SVOX Pico system with `pico_initialize`, the calling program has to pass SVOX Pico a sufficiently large allocated memory block, which is used for all system and engine operations. SVOX Pico handles this memory block with its own memory management. Whenever possible, SVOX Pico will continue operation with graceful degradation when the limits of the given memory are reached.

3.2.5 Function Return Values and Error Handling

The SVOX Pico API function `pico_getData` returns one of the status codes `PICO_STEP_IDLE`, `PICO_STEP_BUSY`, or `PICO_STEP_ERROR`. All other SVOX Pico API functions return a status code as defined in `picodefs.h`. If no error occurred, `PICO_OK` is returned. (Note that `PICO_OK` is also returned if only warnings occurred during the function call as described below.) In the case of an exception, a value < 0 is returned (one of the `PICO_EXC_xxx` constants defined in `picodefs.h`). The functions `pico_getSystemStatusMessage` and `pico_getEngineStatusMessage` can be used to get a more detailed textual description of the error status. `pico_getSystemStatusMessage` must be used to retrieve the error message of a preceding function call on the system level (functions having the system handle as their first argument) whereas `pico_getEngineStatusMessage` must be used to retrieve the error message of a preceding function call on the engine-level (functions having an engine handle as their first argument).

3.2.5.1 Memory Overflow Handling

As explained in section 3.2.4, Pico operates within allocated memory which is given to it by the calling program at the startup of the system. If any API function requires more memory than available, the API function returns the error code `PICO_EXC_OUT_OF_MEM`. Most internal allocations (handled by the Pico internal memory management) are done during system-level API functions, i.e. system creation, resource loading and engine creation. If `PICO_EXC_OUT_OF_MEM` is returned on a system-level API call, then the initial memory area given to Pico is not sufficient for the desired configuration. The only way to make synthesis possible in this situation is to increase the initial memory area (after terminating/unloading the entities initialized/loaded so far), or to change the configuration, e.g. by loading less or smaller resources. Engine-level API calls never return `PICO_EXC_OUT_OF_MEM`.

3.2.5.2 Other Errors

In case of exceptional events and errors that disrupt the normal flow of operation, PICO_EXC_* or PICO_ERR_* are returned. The safest action to take after such a case is to completely shut down the engine that caused the problem (`pico_disposeEngine`), and to create a new engine (`pico_newEngine`).

3.2.5.3 Warnings

Warnings are issued upon minor errors that do not alter the behavior of the API functions and need no special treatment from an application programming point of view, that is, the synthesis application can proceed as usual. Warnings mainly serve to signal improper TTS input, such as, e.g., wrong parameters in markup tags or inexistent sound files to play.

The number of warnings that occurred during an API function call can be retrieved right after the call by `pico_getNrSystemWarnings` or `pico_getNrEngineWarnings`, and the warning

messages themselves can be retrieved by repeatedly calling `pico_getSystemWarning` or `pico_getEngineWarning`, respectively.

Warnings are treated independently from exceptions and hence do not influence the return code of an API function. That is, even if warnings occurred, the API return code is `PICO_OK`.

3.2.6 Multithreading Issues

Due to the fact that the SVOX Pico system runs on many platforms, including platforms without any parallel-processing or multithreading mechanisms, no system-internal automatic process synchronizations are provided. SVOX Pico offers the possibility to have different operations (e.g. synthesis and resource loading, later also different engines) running in parallel. However, if the SVOX Pico system is applied in a multithreaded environment, the application must ensure mutual exclusion of different threads.

To achieve this, the following simple **protection rule** could be adopted:

All calls to SVOX Pico API functions that use an identical first handle (system or engine handle) must be called in sequence, i.e., in a mutually exclusive way.

All calls to SVOX Pico API functions that use *different* first handles (system or engine handles) can be called in parallel (from different threads) without any further protection.

3.2.7 Stepping Mechanism

`pico_getData` is the API function that actually performs TTS and returns the produced speech audio data. It does so in small processing 'steps', so that each API call returns in less than 200ms. Naturally, this function has to be called repeatedly until the whole input is processed, and in some of the calls, no audio data is returned. If all steps are performed for the given input, `pico_getData` will return `PICO_STEP_IDLE`, otherwise it will return `PICO_STEP_BUSY`.

3.2.7.1 Yielding

The simple stepping mechanism of SVOX Pico gives the calling program full control of when and for how long SVOX Pico is entitled to consume CPU power. It is therefore straight forward to implement a CPU-yield functionality using the SVOX Pico API functions.

3.2.7.2 Engine Flushing

SVOX Pico applies contextual analysis to the input text in order to find the correct pronunciation and intonation of each word. Therefore, it may happen that `pico_getData` returns `PICO_STEP_IDLE` although not all the text input is accounted for in the audio output. This is because the Pico engine may wait for more text input to have a complete picture of the context before taking a final decision. (Typically this is the case if the text input so far does not end with a punctuation that terminates the sentence.) In order to force Pico to output the audio corresponding to all the text, a NULL character has to be send as input using `pico_putText*`. After that, calling `pico_getData` will return `PICO_STEP_IDLE` only when all the text up to the flush was processed and the corresponding audio was output.

3.2.7.3 Abortion of Synthesis

Because of the stepping mechanism, no explicit "abort" API function is necessary. If all pending output should be discarded at some point, simply stopping to call `pico_getData` and reset the engine (`pico_resetEngine`) will do the job. However, if the text input so far should be completely synthesized before abandoning the engine, engine flushing (Section 3.2.7.2) should be applied.

3.2.8 Lingware Resource Loading and Unloading

A lingware resource is loaded by `pico_loadResource`, which returns a handle to the resource. `pico_loadResource` is usually called several times to load several resources. The number of resource files loaded in parallel is limited by the constant `PICO_MAX_NUM_RESOURCES` in `picodefs.h`. Loading of a resource file may be done at any time, even in parallel to a working TTS engine, as long as the general protection rules in the Pico API are obeyed (cf. Section 3.2.6).

`pico_unloadResource` is used to unload a resource and to return the occupied memory space to the Pico `system`. Calling `pico_unloadResource` on a resource that is used by an active engine will return `PICO_EXC_RESOURCE_BUSY` with no further effect.

3.2.9 Lingware Resources and Voice Definitions

When we speak of a TTS 'voice', we mean a full set of lingware knowledge bases sufficient to do Text-To-Speech. Thus one TTS voice corresponds to one language/speaker combination.

A TTS voice could be packaged into one resource file, but typically lingware is distributed into speaker-independent ("language") and speaker-dependent ("speaker") resources, so that one voice corresponds to a combination of (at least) two resources. This division is useful, e.g. in order to have two TTS voices share the same language resource. Optional custom-made lingware (additional lexica, text-preprocessing) are usually packaged into their own resources.

Each resource has its own unique resource name. When lingware is delivered to you, the documentation of that lingware contains the unique name of each of the lingware resource files, and also which resources can be combined together.

Defining which combination of resources should actually be used for a particular TTS voice is only done at runtime:

- `pico_createVoiceDefinition` creates a new 'voice definition' introducing a new *voice name* to the system
- `pico_addResourceToVoiceDefinition`. adds a *resource name* to that voice definition.
- `pico_loadResource` actually loads in memory the resource from a 'BIN' file.
- Note that the 'BIN' file contains the unique resource name corresponding to the documentation, and this information is read during loading. This allows the Pico System to map binary loaded resources with the voice definition.
- Note also that it is not mandatory to call `pico_loadResource` before `pico_createVoiceDefinition`.
- The only mandatory action to be taken is to complete the voice definition AND the memory load of ALL needed resources BEFORE calling `pico_newEngine`.
- With `pico_getResourceName` it is possible to retrieve the resource's unique name, once a resource is loaded. This is useful if resources are managed just by their file names.

When creating an engine with `pico_newEngine`, the TTS voice to be used for this engine is then given by the voice name.

3.3 API Specification

Most of this chapter is also included in the '*picoapi.h*' source file that makes part of the SDK distribution. For more details, please refer to this source file.

3.3.1 Conventions

- *Function arguments*: All arguments that only return values are marked by a leading 'out...' in their name. All arguments that are used as input and output value are marked by a leading 'inout...'. All other arguments are read-only (input) arguments.
- *Error handling:* All API functions return a status code which is one of the status constants defined in picodefs.h. In case of an error, a more detailed description of the status can be retrieved. This is done by calling the function 'pico_getSystemStatusMessage' (or 'pico_getEngineStatusMessage' if the error happened on the SVOX Pico engine-level).
- *Warnings:* Unlike errors, warnings do not prevent an API function from performing its function, but output might not be as intended. Functions 'pico_getNrSystemWarnings' and 'pico_getNrEngineWarnings' respectively can be used to determine whether an API function caused any warnings. Details about warnings can be retrieved by calling 'pico_getSystemWarning' and 'pico_getEngineWarning' respectively.
- *Levels:* The API functions could be grouped in two main classes.
  - *System Level API Functions:* the API functions for initialization and resource loading
  - *Engine-level API Functions:* the API functions for performing the synthesis and getting back the audio samples

3.3.2 pico_initialize

```
PICO_FUNC pico_initialize(
    void *memory,
    const pico_Uint32 size,
    pico_System *outSystem
    );
```

System-level API function that initializes the Pico system and returns its handle in 'outSystem'. Input parameters '*memory*' and '*size*' define the location and maximum size of memory in number of bytes that the Pico system will use.
The '*memory*' location has to be a valid pointer of an already allocated memory area, whose size has to be at least equal to '*size*'. In other words, the application using the Pico library has to provide already allocated memory at location '*memory*' for an amount of '*size*'.
The '*size*' required depends on the number of engines and configurations of lingware to be used. No additional memory will be allocated by the Pico system.
This function must be called before any other API function is called. It may only be called once (e.g. at application startup), unless a call to *'pico_terminate' invalidates the Pico system*.

3.3.3 pico_terminate

```
PICO_FUNC pico_terminate(
```



```
    pico_System *outSystem
    );
```

This is a system-level API function that terminates the Pico system. Lingware resources still being loaded are unloaded automatically. The memory area provided to Pico in *'pico_initialize'* is released, so that it can be de-allocated from the application using the Pico library. The system handle becomes invalid.
It is not allowed to call this function as long as Pico engine instances exist. No API function may be called after this function, except for *'pico_initialize'*, which reinitializes the system.

3.3.4 pico_ getSystemStatusMessage

```
PICO_FUNC pico_getSystemStatusMessage (
    pico_System system,
    pico_Status errCode,
    pico_Retstring outMessage
        );
```

System-level status API function that returns in '*outMessage*' a description of the system status or of an error '*errCode*' that occurred with the most recently called system-level API function..

3.3.5 pico_ getNrSystemWarnings

```
PICO_FUNC pico_ getNrSystemWarnings (
    pico_System system,
    pico_Int32 *outNrOfWarnings
    );
```

System-level status API function that returns in '*outNrOfWarnings*' the number of warnings that occurred with the most recently called system-level API function.

3.3.6 pico_ getSystemWarning

```
PICO_FUNC pico_ getSystemWarning (
    pico_System system,
    const pico_Int32 warningIndex,
    pico_Status *outCode,
    pico_Retstring outMessage
    );
```


System-level status API function that returns in '*outMessage*' a description of a warning that occurred with the most recently called system-level API function.
'*warningIndex*' must be in the range 0..N-1 where N is the number of warnings returned by '*pico_getNrSystemWarnings'*.
'*outCode*' returns the warning as an integer code (cf. PICO_WARN_* in picodefs.h)..

3.3.7 pico_ loadResource

```
PICO_FUNC pico_loadResource (
    pico_System system,
    const pico_Char *resourceFileName,
    pico_Resource *outResource
    );
```

System-level Resource loading API function that loads a resource file into the Pico system.
The number of resource files loaded in parallel is limited by PICO_MAX_NUM_RESOURCES in picodefs.h.
Loading of a resource file may be done at any time (even in parallel to a running engine doing TTS synthesis), but with the general restriction that functions taking a system handle as their first argument must be called in a mutually exclusive fashion.
The loaded resource will be available only to engines started after the resource is fully loaded, i.e., not to engines currently running.

3.3.8 pico_ unloadResource

```
PICO_FUNC pico_unloadResource (
    pico_System system,
    pico_Resource *inoutResource
    );
```

System-level Resource loading API function that unloads a resource file from the Pico system.
If no engine uses the resource file, the resource is removed immediately and its associated internal memory is released, otherwise PICO_EXC_RESOURCE_BUSY is returned.

3.3.9 pico_ getResourceName

```
PICO_FUNC pico_getResourceName (
    pico_System system,
    pico_Resource resource,
    pico_Retstring outName
    );
```

System-level Resource inspection API function that retrieves the unique name of a loaded resource file.

3.3.10 pico_ createVoiceDefinition

```
PICO_FUNC pico_createVoiceDefinition(
    pico_System system,
    const pico_Char *voiceName
    );
```

System-level Voice definition API function that creates a voice definition.
Resources must be added to the created voice with *'pico_addResourceToVoiceDefinition'* before using the voice in *'pico_newEngine'*.
It is an error to create a voice definition with a previously defined voice name. In that case, use *'pico_releaseVoiceName'* first.

3.3.11 pico_addResourceToVoiceDefinition

```
PICO_FUNC pico_addResourceToVoiceDefinition(
    pico_System system,
    const pico_Char *voiceName,
    const pico_Char *resourceName
    );
```

System-level Voice definition API function that adds a mapping pair ('*voiceName*', '*resourceName*') to the voice definition. Multiple mapping pairs can added to a voice definition. When calling *'pico_newEngine'* with '*voiceName*', the corresponding resources from the mappings will be used with that engine.

3.3.12 pico_releaseVoiceDefinition

```
PICO_FUNC pico_releaseVoiceDefinition(
    pico_System system,
    const pico_Char *voiceName
    );
```

System-level Voice definition API function that releases the voice definition 'voiceName'.

3.3.13 pico_newEngine

```
PICO_FUNC pico_newEngine(
    pico_System system,
    const pico_Char *voiceName,
```



```
    pico_Engine *outEngine
    );
```

System-level engine creation API function that creates and initializes a new Pico engine instance and returns its handle in '*outEngine*'. Only one instance per system is currently possible.
The voice definition '*voiceName*', identifies the resources from the mappings that will be used with the created engine.

3.3.14 pico_disposeEngine

```
PICO_FUNC pico_disposeEngine(
    pico_System system,
    pico_Engine *inoutEngine
    );
```

System-level engine creation API function that disposes a Pico engine and releases all memory it occupied. The engine handle becomes invalid.

3.3.15 pico_putTextUtf8

```
PICO_FUNC pico_putTextUtf8(
    pico_Engine engine,
    const pico_Char *text,
    const pico_Int16 textSize,
    pico_Int16 *outBytesPut
    );
```

Engine-level API function that puts text '*text*' encoded in UTF8 into the Pico text input buffer. '*textSize*' is the maximum size in number of bytes accessible in '*text*'.
The input text may also contain text-input commands to change, for example, speed or pitch of the resulting speech output.
The number of bytes actually copied to the Pico text input buffer is returned in '*outBytesPut*'.
Sentence ends are automatically detected.
'\0' characters may be embedded in '*text*' to finish text input or separate independently to be synthesized text parts from each other.
Repeatedly calling *'pico_getData'* will result in the content of the text input buffer to be synthesized (up to the last sentence end or '\0' character detected).
To empty the internal buffers without finishing synthesis, use the function *'pico_resetEngine'*.

3.3.16 pico_getData

```
PICO_FUNC pico_getData(
    pico_Engine engine,
```



```
    void *outBuffer,
    const pico_Int16 bufferSize,
    pico_Int16 *outBytesReceived,
    pico_Int16 *outDataType
    );
```

Engine-level API function that gets speech data from the engine.
Every time this function is called, the engine performs, within a short time slot, a small amount of processing its input text, and then gives control back to the calling application.
I.e. after calling *'pico_putTextUtf8*' (with text including a final embedded '\0'), this function needs to be called repeatedly till '*outBytesReceived*' bytes are returned in '*outBuffer*'.
The type of data returned in '*outBuffer*' (e.g. 8 or 16 bit PCM samples) is returned in '*outDataType*' and depends on the lingware resources.
Possible '*outDataType*' values are listed in picodefs.h (PICO_DATA_*).
This function returns PICO_STEP_BUSY while processing input and producing speech output.
Once all data is returned and there is no more input text available in the Pico text input buffer, PICO_STEP_IDLE is returned.
All other function return values indicate a system error.

3.3.17 pico_resetEngine

```
PICO_FUNC pico_resetEngine(
    pico_Engine engine
    );
```

Engine-level API function that resets the engine and clears all engine-internal buffers, in particular text input and signal data output buffers.
This function should be used to restore the engine initial state, i.e. after an engine-level API function returns an error.

3.3.18 pico_getEngineStatusMessage

```
PICO_FUNC pico_getEngineStatusMessage(
    pico_Engine engine,
    pico_Status errCode,
    pico_Retstring outMessage
    );
```

Engine-level API function that returns in '*outMessage*' a description of the engine status or of an error that occurred with the most recently called engine-level API function.

3.3.19 pico_getNrEngineWarnings

```
PICO_FUNC pico_getNrEngineWarnings(
```



```
    pico_Engine engine,
    pico_Int32 *outNrOfWarnings
    );
```

Engine-level API function that returns in '*outNrOfWarnings*' the number of warnings that occurred with the most recently called engine-level API function.

3.3.20 pico_getEngineWarning

```
PICO_FUNC pico_getEngineWarning(
    pico_Engine engine,
    const pico_Int32 warningIndex,
    pico_Status *outCode,
    pico_Retstring outMessage
    );
```

Engine-level API function that Returns in '*outMessage*' a description of a warning that occurred with the most recently called engine-level API function. 'warningIndex' must be in the range 0..N-1 where N is the number of warnings returned by *'pico_getNrEngineWarnings'*. 'outCode' returns the warning as an integer code (cf. PICO_WARN_*).

4 Input and Output File Formats

In its basic operation, SVOX Pico does not consume or produce files, but inputs and outputs data via API function calls:

Text is input via the `pico_putTextUtf8` API function which accepts Unicode strings encoded in UTF-8. (See Section A.1 for the valid character sets and chapter 7.2 for phonetic input.)

Audio data is fetched via the `pico_getData` API function, which delivers raw *16-bit linear PCM encoded* (linearly quantized 16 bit values) audio samples *with16 kHz sampling frequency*.

If text files are be synthesized or the synthesized audio is to be stored in a file, it is up to the application developer to create the corresponding wrappers.

However, Pico offers **text input commands** (markup tags, cf.Chapter 8) that allow input sound files to be integrated into the output audio stream (`<play>` and `<usesig>`), and to write out (usually small) pieces of synthesized audio to a sound file (`<genfile>`).

In these cases, the format of the files is *wav* format (Windows WAVE format), with the identical encoding and sampling frequency as the audio data output via `pico_getData`.

5 Improving SVOX Pico Text-to-Speech Output

5.1 Introduction

No text-to-speech synthesis system today can claim perfection in the way it converts written text into speech. However, especially in applications that generate the text to be synthesized (e.g., in automatic information systems), the application may help the synthesis process to produce better results. This can be achieved by different means in SVOX Pico. The following sections and the chapters on Markup Language (Chapter 8) provide information on how the synthesis results can be improved for the SVOX Pico system.

5.2 Mixing Voice Prompts with Text-To-Speech

In many applications that include the SVOX Pico text-to-speech component, the resulting voice output quality can be improved by mixing pre-recorded natural-voice prompts for fixed sentence parts with synthesized speech for varying parts. For example, a traffic event information system might try to output a sentence like

> "Vehicles are restricted inbound to *Denver*."

by using a natural-voice prompt for "Vehicles are restricted inbound to" and a synthesized voice for "Denver". It is possible to do so in SVOX Pico by means of the markup tags `<play>` and `<usesig>` (cf. Chapter 7).

In order to avoid the change of voice characteristics when mixing voice prompts with text-to-speech, we recommend having your own **Custom Voice** developed by SVOX. This way, you can have your own text-to-speech system with the same voice characteristics as the speaker you use for creating prompts.

5.3 Insertion of Pauses

A sentence pause is inserted at the end of every sentence. An additional sentence pause is automatically inserted if at least one blank line separates two paragraphs. If a punctuation character is contained in the input text, it often indicates a strong prosodic phrase boundary, and a sentence-internal pause is automatically inserted.

Additional pauses can be inserted and controlled using the markup tag `<break>` which is described in Chapter 7.

5.4 Structured Numbers

Common structured numbers such as telephone numbers are treated properly by the SVOX Pico system. For other structured numbers, or to modify the default behavior, it is recommended to introduce commas between number groups in order to put an appropriate pause between the groups. A '`.`' at the end of the sentence should be separated from a preceding number by a blank, since otherwise the number might be considered an ordinal number in some languages. In most cases though, SVOX Pico text preprocessing will handle this correctly.

For example: "Customer number: 8 6 4, 4 3 3, 9 9 7 ."

6 SVOX Pico Text Preprocessing

The SVOX Pico TTS engine contains an advanced text preprocessor that handles the correct pronunciation of e.g., dates, telephone numbers, and abbreviations.

A detailed description of the predefined text preprocessing capabilities for each of the supported languages is included in Appendix B.

7 SVOX Pico Markup Language

7.1 Introduction

The input text submitted to SVOX Pico can be enriched by text input commands that alter the way in which the text or a part of it is synthesized, and to invoke special actions such as the playback of sound files at certain positions.

Text input commands are input in the form of markup tags. In general, text input commands specify that certain text portions are to be treated in a special way. These **markup sections** are denoted by a **start tag** and a corresponding **end tag**. In some cases, text input commands invoke actions at a specific point in the text rather than modifying the treatment of a text portion. In these cases, the start and the end tag can be melded into a single **combined tag**.

The markup tags interpreted by SVOX Pico are of one of the following forms:

- `<command>` : start tag for a parameterless command
- `<command parameter="value">` : start tag for a command with the specification of a value for a required parameter
- `<command parameter='value'>` : identical to `<command parameter="value">`
- `</command>` : end tag for all kinds of commands
- `<command/>` : combined start/end tag, equivalent to `<command></command>`
- `<command parameter="value"/>` : combined start/end tag, equivalent to `<command parameter="value"></command>`
- `<command parameter='value'/>` : combined start/end tag, equivalent to `<command parameter='value'></command>`

Markup sections of identical type cannot be (directly or indirectly) nested in SVOX Pico, with the sole exception of the markup section `<ignore>`. Markup tags that cannot be interpreted by SVOX Pico are synthesized. In addition, syntax errors detected by the SVOX Pico system are reported as warnings.

In parameter value strings (enclosed in single or double quotes), the backslash character `\` is an escape character. Any character following a backslash, including a second backslash, will be interpreted verbatim. Using this escape character it is possible to have single and double quote characters in the same parameter value string. E.g. the parameter value `"h@\"l@_U"` will result in `h@"l@_U`. The `file` parameter used in several markup tags is an exception: to simplify the specification of file names including path information, the backslash in not an escape character for the `file` parameter.

WARNING: The SVOX Pico markup tags are of the same form as found in the HTML or XML language. Despite this tag form similarity to well-known markup languages, developers familiar with, e.g., HTML should be aware of the simpler markup tag processing capabilities of the SVOX Pico system (e.g. limited nesting). This simpler processing was selected to limit the memory consumption of the SVOX Pico system and keep the performance high, while still fulfilling the needs of text markup for TTS.

7.2 Markup Tags Interpreted by SVOX Pico

Note: All the following tag identifiers may be preceded by the prefix `svox:`, e.g. `<svox:ignore>`

- **<ignore>: Ignoring text**

  `<ignore> ... </ignore>`

  A text portion marked by `<ignore> ... </ignore>` is fully ignored by the synthesis. Ignored sections may be nested.

  Example: `Hello <ignore> any text </ignore> Mister Smith.`

  In this example, the input text is synthesized as if it were only the sentence `"Hello Mister Smith."`

- **<p>: Paragraph structure**

  `<p> ... </p>` or `<paragraph> ... </paragraph>`

  Paragraph structures can be marked by `<p> ... </p>` or its extended form `<paragraph> ... </paragraph>`. In most cases, SVOX Pico automatically detects paragraph structures. The `<p>` tag can be used to enforce the setting of a paragraph structure.

  Example: `<p> This is a paragraph. </p>`

  In this example, the enclosed text is structured as a paragraph.

- **<s>: Sentence structure**

  `<s> ... </s>` or `<sentence> ... </sentence>`

  Sentence structures can be marked by `<s> ... </s>` or its extended form `<sentence> ... </sentence>`. In most cases, SVOX Pico automatically detects sentence structures. The `<s>` tag can be used to enforce the setting of a sentence structure.

  Example: `<s> This is a sentence. </s>`

  In this example, the enclosed text is structured as a sentence.

- **<break>: Break control**

  `<break time="..."/>`

  The markup tag `<break time="..."/>`. controls the pausing or other prosodic boundaries between words. It is most often used to override the typical automatic behavior of SVOX Pico. The overriding effect operates on the boundary to the right of the markup. The duration of a pause can be specified in seconds [s] or milliseconds [ms] as a value of the parameter `time`. Only integer positive values are accepted.

  Examples:

  ```
  This is the first sentence<break time="500ms"/>.  This is the second
  sentence.
  ```

  In this example, a pause of 500 milliseconds is inserted between the two sentences *instead* of the automatic inter-sentence pause.


```
This is the first sentence. <break time="2s"/> This is the second
sentence.
```

In this example, an *additional* pause of 2 seconds is inserted between the two sentences.

- **<pitch>: Setting the pitch level**

  ```
  <pitch level="..."> ... </pitch>
  ```

  The markup tag `<pitch level="...">` changes the general pitch level of the specified text portion to the value given as a value of the parameter `level`. The normal pitch level is 100, the allowed values lie between 50 (one octave lower) and 200 (one octave higher). The end tag `</pitch>` resets the pitch level to 100.

  Example: `Hello, <pitch level="140"> Miss Jones </pitch> arrived.`

  In this example, the section `"Miss Jones"` will be produced at a pitch level of a factor of 1.4 higher than normal.

  The pitch level can be set relative to the current setting by adding a percent character to the level value. E.g. setting the level to `"150%"` will set the pitch level to 150% of the current value. Allowed percentage values lie between 50% and 200%.

  Reducing and increasing the pitch level by a percentage is achievable by preprending a plus or minus character to the level value. When setting for example the level to `"-20%"` the current pitch will be reduced by 20%. Allowed percentage change values lie between -50% and +100%.

- **<speed>: Setting the speed level**

  ```
  <speed level="..."> ... </speed>
  ```

  The markup tag `<speed level="...">` changes the general speed level of the specified text portion to the value given as a value of the parameter `level`. The normal speed level is 100, the allowed values lie between 20 (slowing down by a factor of 5) and 500 (speeding up by a factor of 5). The end tag `</speed>` resets the speed level to 100.

  Example: `Hello, <speed level="300"> Miss Jones </speed> arrived.`

  In this example, the section `"Miss Jones"` will be produced by a factor of 3 faster than normal.

  As for the pitch markup tag, relative speed levels and percentage changes can be specified using the percent, plus, and minus characters in the level value.

- **<volume>: Setting the volume level**

  ```
  <volume level="..."> ... </volume>
  ```

  The markup tag `<volume level="...">` changes the volume level of the specified text portion to the value given as a value of the parameter `level`. The normal volume level is 100. Increasing the volume level (values > 100) may result in degraded signal quality due to saturation effects (clipping) and is not recommended. The allowed volume levels lie between 0 (i.e. no audible output) and 500 (increasing the volume by a factor of 5). The end tag `</volume>` resets the volume level to 100.

  Example: `Hello, Miss <volume level="50"> Jones </volume> arrived.`

In this example, the volume of the section `"Jones"` will be decreased by a factor of 2.

As for the pitch markup tag, relative volume levels and percentage changes can be specified using the percent, plus, and minus characters in the level value.

- **<voice>: Setting the voice**

  ```
  <voice name="..."> ... </voice>
  ```

  The markup tags `<voice name="...">` ... `</voice>` are accepted by SVOX Pico for reasons of compatibility with future versions.

  In the current version, the markup tag pair `<voice name="..."> ... </voice>` behaves like a `<sentence>="..."> ... </sentence>` pair.

- **<preproccontext>: Setting the preprocessing context**

  ```
  <preproccontext name="..."> ... </preproccontext>
  ```

  The SVOX Pico text preprocessor allows several preprocessor contexts to be defined. With the markup tag `<preproccontext name="...">` the currently active context can be changed (if several contexts exist). The SVOX Pico SDK contains a single context named `DEFAULT` that is active by default. Additional application-specific contexts can be implemented by SVOX Pico for specific projects.

  Example: `<preproccontext name="CMD"> advice uturn </preproccontext>`

  In this example it is assumed that an application-specific context named `CMD` exists that preprocesses specific navigation commands in a special way.

- **<phoneme>: Defining the pronunciation form**

  ```
  <phoneme [alphabet="..."] ph="..."/>
  ```

  The tag `<phoneme alphabet="..." ph="..."/>` provides a phonemic or phonetic pronunciation for a word to be inserted into the text in the place of the markup. The markup tag pair `<phoneme alphabet="..." ph="..."/>` is accepted by SVOX Pico for reasons of compatibility with future versions, but has undefined behavior.

  The `ph` parameter is a required parameter that specifies the phoneme or phone string. The phoneme or phone string does not undergo any sort of text normalization or replacement by entries in the lexicon.

  `alphabet` is an optional parameter that specifies the phonemic/phonetic alphabet used for the string `ph`. The valid values for this parameter includes

  ```
  xsampa
  ```

  which corresponds to the Unicode representations of the language-independent phonetic characters developed by the International Phonetic Association. However, for each language, only a language-dependent subset of the phonetic characters and their combinations is accepted, as specified in chapter A.2.1.

  If the parameter `alphabet` is completely omitted then the phoneme string is assumed to be `xsampa`.

  Example:

  ```
  Good evening mister <phoneme alphabet="xsampa" ph="\"br\\a_Un"/> .
  ```


In this example for a British English voice, the pronunciation of the German name `"Braun"` is set to `"br\a_Un`. Note that the xsampa sound "r" is not in the valid subset for English and the xsampa sound r\ (voiced (post-)alveolar approximant) is used instead.

NOTE: In the `ph` argument of the `<phoneme ph='...'> tag`, the double quote (primary stress) and the backslash must be escaped by a backslash ('\').

Further information about the SVOX Pico phone sets can be found in chapter A.2.1.

- **<mark>: Setting position markers**

```
<mark name="..."/>
```

The markup tag `<mark name="..."/>` is accepted by SVOX Pico for reasons of compatibility with future versions. It has currently no effect.

- **<genfile>: Generating sound files**

```
<genfile file="..."> ... </genfile>
```

Using the markup section `<genfile file="..."> ... </genfile>` an arbitrary text portion can be saved to a separate sound file. The format and encoding of the output sound file is determined as described in Chapter 4. The markup tag `<genfile>` can be used especially for creating synthetic voice prompts.

Examples:

```
<genfile file="intro.wav"> Hello, Mister </genfile>
<genfile file="miller.wav"> Miller, </genfile>
<genfile file="rem.wav"> welcome.</genfile>
Hello, Mister <genfile file="smith.wav"> Smith, </genfile> welcome.
```

In this example, three parts of the first sentence `"Hello, Mister Miller, welcome."` are saved in separate sound files of type wav, and the part `"Smith,"` of the second sentence is also saved in a separate file. These generated sound files can, for instance, be used to generate a newly combined utterance using the markup tag `<play>`:

```
<play file="intro.wav"/> <play file="smith.wav"/>
<play file="rem.wav"/>
```

This input "text", which consists only of markup tags, plays the generated sound files as if the (new) sentence `"Hello, Mister Smith, welcome."`.had been synthesized, but, of course, with much less processing time. It is essential to note that the sentence part `"Smith,"` should be synthesized and saved to a sound file in the appropriate sentence environment in order for the newly combined utterance to have a continuous melody and rhythm.

- **<play>: Playing sound and intermediate synthesis units files**

```
<play file="..."/>  or  <play file="..."> ... </play>
```

Using the empty tag `<play file="..."/>` or tag pair `<play file="..."> ... </play>` results in insertion of the specified sound file or intermediate synthesis units in the synthesized signal at the place specified in the input text. In the first variant, the sound file is played at the position where the combined tag is set, in the second variant the sound file is played as a substitute of the text between the start and the end tag, that is, the text between the start and the end tag is ignored. The input file formats of the markup tag `<play>` are determined according to the rules described in Chapter 4. The sampling frequency of the input sound file or the intermediate synthesis units file must be identical to the sampling frequency of the synthesized signal.

Examples:

```
Hello, Miss <play file="miller.wav"/>.
Hello, Miss <play file="miller.wav"> Miller </play>.
```

Remarks:

- If <play> is used as part of a longer sentence, the use of the markup section <usesig> might be more appropriate in order to produce a better sentence melody.
- The volume of the played sound file or intermediate synthesis units file are modified within volume markup sections. The pitch and speed of intermediate synthesis units files are modified within pitch and speed markup sections, but sound files will always be played with their original pitch and speed.

- **<usesig>: Replacing text portions by sound and intermediate synthesis units files**

```
<usesig file="..." [f0beg="..."] [f0end="..."]> ... </usesig>
```

The markup tags <usesig file="..."> and </usesig> are similar to the tags </play file="..."> and </play>, that is, the text enclosed between the start and the end tag is replaced by the specified sound or intermediate synthesis units file. There is, however, an important difference to the use of <play>: the orthographic text enclosed in the <usesig> section is used to interpret the entire sentence as if the <usesig> markup tags were not present. The contextual interpretation of the entire sentence determines the prosody (melody and rhythm) and in some cases even the pronunciation (the choice of phones) of the words.

Example:

```
<usesig file="intro.wav"> Hello, Mister </usesig> Smith,
<usesig file="rem.wav"> welcome. </usesig>
```

In this example, the sound files intro.wav and rem.wav are played instead of synthesizing the text portions "Hello, Mister" and "welcome." The prosody (melody and rhythm) of the word "Smith," is, however, generated as if the entire sentence "Hello, Mister Smith, welcome." were synthesized. By contrast, in the <play>-variant

```
<play file="intro.wav"> Hello, Mister </play> Smith,
<play file="rem.wav"> welcome. </play>
```

the prosody of "Smith" is determined as if the entire sentence were only "Smith." This could lead to unwanted discontinuities in the prosody of the entire utterance.

As a general rule, the markup tag <usesig> should be preferred over <play> if only a part of a sentence is to be replaced by a sound file and the remainder is synthesized. <usesig> is especially suited if fixed parts of a sentence should be taken from pre-synthesized sound files, recorded prompts, or intermediate synthesis units files, and only varying parts (e.g., varying proper names) should be synthesized. In this case, the application of <usesig> may considerably speed up the synthesis, while the variable parts of the sentence still fit in the prosody of the overall sentence.

If recorded voice prompts are mixed with synthesized speech using the <usesig> tag, the perceived overall quality of the output can be increased. It is important though, that the same speaker is used for creating the prompts and to develop the synthetic voice.

A Languages and Voices

A.1 Language Identifiers and Input Character Sets

The following table lists the currently available languages together with:

- ISO 639-2 - ISO 3166 language code (which are combined in rfc 3066)
- Input character set coverage used in the SVOX Pico system. Characters need to be input in UTF-8 format

language	*ISO code <langid>*	*character set coverage*
German (Germany)	`de-DE`	`Latin 1 (ISO 8859-1)`
English (United Kingdom)	`en-GB`	`Latin 1 (ISO 8859-1)`
English (USA)	`en-US`	`Latin 1 (ISO 8859-1)`
French (France)	`fr-FR`	`Latin 1 (ISO 8859-1)`
Italian (Italy)	`it-IT`	`Latin 1 (ISO 8859-1)`
Spanish (Spain)	`es-ES`	`Latin 1 (ISO 8859-1)`

The language components contained in a lingware resource are identified in the unique resource name by the ISO 639-2 – ISO 3166 code. This 5 or 6-character identification `<langid>` is used as initial part of the resource name. Language Resource files are usually shipped with the file name `<unique resource name>.bin`.

A.2 Lingware Resources

Currently, the following lingware resources are available:

A.2.1 English (United Kingdom)

en-GB_ta_1.0.0.3-0-0	British English language resource
en-GB_kh0_sg_1.0.0.3-0-0	British English female voice resource

A.2.2 German (Germany)

de-DE_ta_1.0.0.3-0-0	German language resource
de-DE_gl0_sg_1.0.3.0-0-0	German female voice resource

A.2.3 English (USA)

en-US_ta_1.0.0.3-0-0	American English language resource
en-US_lh0_sg_1.0.0.3-0-0	American English female voice resource

A.2.4 Spanish (Spain)

es-ES_ta_1.0.0.3-0-0	French language resource
es-ES_zl0_sg_1.0.0.3-0-0	French female voice resource

A.2.5 French (France)

fr-FR_ta_1.0.0.3-0-1	French language resource
fr-FR_nk0_sg_1.0.0.3-0-1	French female voice resource

A.2.6 Italian (Italy)

it-IT_ta_1.0.0.3-0-0	Italian language resource
it-IT_cm0_sg_1.0.0.3-0-0	Italian female voice resource

A.3 Phonetic Alphabets

A.3.1 Introduction

X-SAMPA (Extended Speech Assessment Methods Phonetic Alphabet), a variant of the SAMPA alphabet (SAM Phonetic Alphabet, resulting from the ESPRIT project SAM/"Speech Assessment Methods"), is a computer-readable phonetic alphabet. It was designed to unify the individual language SAMPA alphabets, and extend SAMPA to cover the entire range of characters in the International Phonetic Alphabet (IPA). SVOX Pico X-SAMPA is the base alphabet of which *language-specific subsets* are used to represent phonetic entries in the markup tag `<phoneme>`.

The following sections list the complete list of X-SAMPA symbols used as a base alphabet and then the language-specific subsets available for SVOX Pico.

A.3.2 List of valid X-Sampa Symbols (Base Alphabet)

The following is a complete list of the symbols defined in the official X-SAMPA documents.[1] The only purpose of this list is to give information about the phonetic value of the symbols used in the language specific descriptions in the sections to follow and to provide a systematic overview of the whole inventory. It is not intended as a specification for phonetic input; the user should refer for this purpose to the descriptions available for each language.

A.3.3 Consonants

A.3.3.1 Pulmonic

4	alveolar tap
<\	voiced epiglottal fricative
>\	epiglottal plosive
?	glottal stop
?\	pharyngeal fricative, voiced
B	bilabial fricative, voiced
B\	bilabial trill
C	palatal fricative, voiceless
D	dental fricative, voiced
F	labiodental nasal
G	velar fricative, voiced
G\	voiced uvular plosive
H	voiced labial-palatal approx.

[1] *Computer-coding the IPA: a proposed extension of SAMPA (http://www.phon.ucl.ac.uk/home/sampa/x-sampa.htm)*


```
H\      voiceless epiglottal fricative
J       palatal nasal
J\      voiced palatal plosive
K       alveolar lateral fricative, vl.
K\      alveolar lateral fricative, vd.
L       palatal lateral approximant
L\      velar lateral approximant
M\      velar approximant
N       velar nasal
N\      uvular nasal
P       labiodental approximant
R       uvular fricative, voiced
R\      uvular trill
S       postalveolar fricative, voiceless
T       dental fricative, voiceless
W       voiceless labial-velar fricative
X       uvular fricative, voiceless
X\      pharyngeal fricative, voiceless
Z       postalveolar fricative, voiced
b       voiced bilabial plosive
c       voiceless palatal plosive
d       voiced alveolar plosive
d`      retroflex plosive, voiced
f       voiceless labiodental fricative
g       voiced velar plosive
h       voiceless glottal fricative
h\      glottal fricative, voiced
j       palatal approximant
j\      palatal fricative, voiced
k       voiceless velar plosive
l       alveolar lateral approximant
l\      alveolar lateral flap
l`      retroflex lateral approximant
m       bilabial nasal
n       alveolar nasal
n`      retroflex nasal
p       voiceless bilabial plosive
p\      bilabial fricative, voiceless
q       voiceless uvular plosive
r       alveolar trill
r\      alveolar approximant
r\`     retroflex approximant
r`      retroflex flap
s       voiceless alveolar fricative
```



```
s\      alveolo-palatal fricative, vl.
s`      retroflex fricative, voiceless
t       voiceless alveolar plosive
t`      retroflex plosive, voiceless
v       voiced labiodental fricative
v\      labiodental approximant
w       labial-velar approximant
x       voiceless velar fricative
x\      simultaneous S and x
z       voiced alveolar fricative
z\      voiced alveolo-palatal fricative
z`      retroflex fricative, voiced
```

A.3.3.2 Non-pulmonic

A.3.3.2.1 Clicks

```
!\      (post)alveolar
=\      palatoalveolar
O\      bilabial
|\      dental
|\|\    alveolar lateral
```

A.3.3.2.2 Ejectives, Implosives[2]

```
_<      implosive
_>      ejective
```

A.3.4 Vowels

```
&       open front rounded
1       close central unrounded
2       close-mid front rounded
3       open-mid central unrounded
3\      open-mid central rounded
6       open schwa (turned a)
7       close-mid back unrounded
8       close-mid central rounded
9       open-mid front rounded
@       schwa
@\      close-mid central unrounded
A       open back unrounded
E       open-mid front unrounded
I       near-close near-front
M       close back unrounded
O       open-mid back rounded
```

[2] Diacritics only.

Q	open back rounded
U	rounded near-close near-back
V	open-mid back unrounded
Y	rounded near-close near-front
a	open front unrounded
e	close-mid front unrounded
i	close front unrounded
o	close-mid back rounded
u	close back rounded
y	close front rounded
{	ash (ae digraph)
}	close central rounded
I\	central lax close unrounded vowel[3]
U\	central lax close rounded vowel[4]

A.3.5 Suprasegmentals

A.3.5.1 General

#	word separator
%	secondary stress
-\	linking mark
.	syllable break
:	long
:\	half-long
_X	extra-short
"	primary stress

A.3.5.2 Level Tones

!	downstep
^	upstep
_/	rising tone
_B	extra low
_H	level high
_L	level low
_M	level mid
_T	level extra high
_\	falling tone

A.3.5.3 Contour Tones

_B_L	low rising
_F	falling
_H_T	high rising
_R	rising

[3] Symbol not in IPA

[4] Symbol not in IPA


```
_R_F    rising-falling
```

A.3.5.4 Prosodic, Paralinguistic or other Non-segmental Notation

```
<       begin nonsegmental notation
>       end nonsegmental notation
<F>     global fall
<R>     global rise
|       minor (foot) group
||      major (intonation) group
```

A.3.5.5 Language Specific Tones[5]

```
_1
_2
_3
_4
_5
_6
```

A.3.6 Diacritics

```
'       palatalized
5       velarized l
=       syllabic
_+      advanced
_-      retracted
_0      voiceless
_=      syllabic
_?\     pharyngealized
_A      advanced tongue root
_G      velarized
_N      linguolabial
_O      more rounded
_^      non-syllabic
_a      apical
_c      less rounded
_d      dental
_e      velarized or pharyngealized
_h      aspirated
_j      palatalized
_k      creaky voiced
_l      lateral release
_m      laminal
_n      nasal release
_o      lowered
```

[5] The meaning of these symbols is language specific and will be documented separately.

_q	retracted tongue root
_r	raised
_t	breathy voiced
_v	voiced
_w	labialized
_x	mid-centralized
_}	no audible release
_~	nasalized
`	rhoticity
~	nasalized
_"	centralized

A.3.7 Other Symbols

*	undefined escape character
-	separator
/	indeterminacy in French vowels
_	tie bar

A.4 Language Specific Descriptions

A.4.1 German (de-DE)

A.4.1.1 Consonants

A.4.1.1.1 Plosives

?	Abend	"?a:.b@nt
b	**B**all	"bal
d	**D**ach	"dax
g	**G**abe	"ga:.b@
k	**K**amm	"kam
p	**P**ost	"pOst
t	**T**ritt	"tRIt

A.4.1.1.2 Nasals

N	la**ng**	"laN
m	**M**atte	"ma.t@
n	**N**est	"nEst

A.4.1.1.3 Fricatives

C	Mil**ch**	"mIlC
R	**R**abe	"Ra:.b@
S	Ma**sch**e	"ma.S@
Z	**G**enie	Ze."ni:
f	A**ff**e	"?a.f@

h	**H**und	"hUnt
s	Ma**ss**e	"ma.s@
v	**W**ahl	"va:l
x	Ba**ch**	"bax
z	Ro**s**e	"Ro:.z@

A.4.1.1.4 Approximants

j	**J**agd	"ja:kt
l	**L**uft	"lUft

A.4.1.1.5 Syllabic Consonants

l=	Nab**el**	"na:.b=l
m=	hart**em**	"haR.t=m
n=	Latt**en**	"la.t=n

A.4.1.1.6 Affricates

d_Z	**Dsch**ungel	"d_ZU.N@l
p_f	Dam**pf**	"damp_f
t_S	Kla**tsch**	"klat_S
t_s	**Z**unge	"t_sU.N@

A.4.1.2 Monophthongs

A.4.1.2.1 Short

2	**Ö**konom	?2.ko."no:m
6	Halt**er**	"hal.t6
9	g**ö**ttlich	"g9t.lIC
@	Lag**e**	"la:.g@
E	K**e**tte	"kE.t@
I	K**i**ste	"kIs.t@
O	K**o**pf	"kOp_f
U	K**u**nst	"kUnst
Y	f**ü**llen	"fY.l@n
a	K**a**ppe	"ka.p@
e	Th**e**orie	te.o."Ri:
i	Pol**i**tik	po.li."ti:k
o	pol**ie**ren	po."li:.R@n
u	K**u**rier	ku."Ri:6_^
y	Mythologie	my.to.lo."gi:

A.4.1.2.2 Long

2:	**Ö**l	"?2:l
E:	B**ä**r	"bE:6_^
a:	K**a**ter	"ka:.t6
e:	B**ee**t	"be:t

i:	B**ie**ne	"bi:.n@
o:	B**oo**t	"bo:t
u:	H**u**t	"hu:t
y:	k**üh**l	"ky:l

A.4.1.2.3 Nasalized

9~:	parf**um**	par."f9~:
E~:	T**im**bre	"tE~:.bR@
a~:	Ch**an**ce	"Sa~:.s@
o~:	f**on**d	"fo~:

A.4.1.2.4 Non-syllabic

6_^	Bie**r**	"bi:6_^
i_^	Mum**i**e	"mu:.mi_^@"
o_^	f**o**yer	fo_^a."je:
u_^	akt**u**ell	?ak."tu_^El

A.4.1.3 Diphthongs

O_Y	D**eu**tsch	"dO_Yt_S
a_I	B**ei**n	"ba_In
a_U	K**au**f	"ka_Uf

A.4.1.4 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

A.4.2 English (en-GB)

A.4.2.1 Consonants

A.4.2.1.1 Plosives

b	**b**in	"bIn
d	**d**in	"dIn
g	**g**ive	"gIv
k	**k**in	"kIn
p	**p**in	"pIn
t	**t**in	"tIn

A.4.2.1.2 Nasals

N	thi**ng**	"TIN
m	**m**ock	"mQk
n	**kn**ock	"nQk

A.4.2.1.3 Fricatives

D	**th**is	"DIs
S	**sh**in	"SIn
T	**th**in	"TIn
Z	mea**s**ure	"mE.Z@
f	**f**it	"fIt
h	**h**it	"hIt
s	**s**in	"sIn
v	**v**im	"vIm
z	zing	"zIN

A.4.2.1.4 Approximants

j	**y**acht	"jQt
l	**l**ong	"lQN
l=	litt**le**	"lI.tl=
r\	**r**ight	"r\a_It
w	**w**asp	"wQsp

A.4.2.1.5 Affricates

d_Z	**g**in	"d_ZIn
t_S	**ch**in	"t_SIn

A.4.2.2 Monophthongs

A.4.2.2.1 Short

@	**a**llow	@."la_U
E	p**e**t	"pEt
I	p**i**t	"pIt
Q	p**o**t	"pQt
U	p**u**t	"pUt
V	c**u**t	"kVt
{	p**a**t	"p{t

A.4.2.2.2 Long

3:	f**ur**s	"f3:z
A:	st**ar**s	"stA:z
i:	**ea**se	"i:z
o:	c**au**se	"ko:z
u:	l**o**se	"lu:z

A.4.2.3 Diphthongs

@_U	n**o**se	"n@_Uz
I_@	f**ea**rs	"fI_@z
O_I	n**oi**se	"nO_Iz
U_@	c**u**res	"kjU_@z

a_I	r**i**se	"r\a_Iz
a_U	r**ou**se	"r\a_Uz
e_@	st**ai**rs	"ste_@z
e_I	r**ai**se	"r\e_Iz

A.4.2.4 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

A.4.3 English (en-US)

A.4.3.1 Consonants

A.4.3.1.1 Plosives

b	**b**in	"bIn
d	**d**in	"dIn
g	**g**ive	"gIv
k	**k**in	"kIn
p	**p**in	"pIn
t	**t**in	"tIn

A.4.3.1.2 Nasals

N	thi**ng**	"TIN
m	**m**ock	"mA:k
n	**kn**ock	"nA:k

A.4.3.1.3 Fricatives

D	**th**is	"DIs
S	**sh**in	"SIn
T	**th**in	"TIn
Z	mea**s**ure	"mE.Z@`
f	**f**it	"fIt
h	**h**it	"hIt
s	**s**in	"sIn
v	**v**im	"vIm
W	**wh**ale	"We_Il
z	**z**ing	"zIN

A.4.3.1.4 Approximants

j	**y**acht	"jA:t
l	**l**ong	"lA:N
l=	litt**le**	"lI.tl=
r\	**r**ight	"r\a_It
w	**w**asp	"wA:sp

A.4.3.1.5 Affricates

d_Z	**g**in	"d_ZIn
t_S	**ch**in	"t_SIn

A.4.3.2 Monophthongs

A.4.3.2.1 Short

@	**a**llow	@."la_U
@`	act**or**	"{k.t@`
E	p**e**t	"pEt
I	p**i**t	"pIt
U	p**u**t	"pUt
V	c**u**t	"kVt
{	p**a**t	"p{t

A.4.3.2.2 Long

3`:	f**ur**s	"f3`:z
A:	st**a**rs	"stA:rz
i:	**ea**se	"i:z
O:	f**ou**r	"fO:r\
u:	l**o**se	"lu:z

A.4.3.3 Diphthongs

o_U	n**o**se	"no_Uz
O_I	n**oi**se	"nO_Iz
a_I	r**i**se	"r\a_Iz
a_U	r**ou**se	"r\a_Uz
e_I	r**ai**se	"r\e_Iz

A.4.3.4 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

A.4.4 Spanish (es-ES)

A.4.4.1 Consonants

A.4.4.1.1 Plosives

b	**v**ino	"bi.no
d	**d**onde	"don.de
g	**g**ato	"ga.to
k	**c**asa	"ka.sa
p	**p**uente	"pu_^en.te
t	**t**iempo	"tjem.po

A.4.4.1.2 Fricatives

f	**f**ácil	"fa.Til
x	hi**j**o	"i.xo
s	**s**opa	"so.pa
B	ca**b**eza	ka."Be.Ta
D	de**d**o	"de.Do
G	lue**g**o	"lu_^e.Go
T	**z**ona	"To.na
j\	cón**y**uge	"kon.j\u.xe

A.4.4.1.3 Affricates

t_S	mu**ch**o	"mu.t_So

A.4.4.1.4 Approximants

l	**l**ejos	"le.xos
L	caba**ll**o	ca."Ba.Lo
j	**r**ey	"rej

A.4.4.1.5 Nasals

J	a**ñ**o	"a.Jo
N	nu**n**ca	"nuN.ka
m	**m**onte	"mon.te
n	**n**ota	"no.ta

A.4.4.1.6 Tap, trill

4	hora	"o.4a
r	torre	"to.re

A.4.4.2 Vowels

a	v**a**lle	"ba.Le
e	p**e**ro	"pe.4o
i	p**i**co	"pi.ko
o	t**o**t**o**	"to.4o
u	d**u**ro	"du.4o
u_^	b**ue**no	"bu_^e.no

A.4.4.3 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

A.4.5 French (fr-FR)

A.4.5.1 Consonants

A.4.5.1.1 Plosives

b	**b**on	"bO~
d	**d**ans	"dA~
g	**g**ant	"gA~
k	**qu**and	"kA~
p	**p**ont	"pO~
t	**t**emps	"tA~

A.4.5.1.2 Nasals

J	oi**gn**on	O."JO~
N	campi**ng**	kA~."piN
m	**m**ont	"mO~
n	**n**om	"nO~

A.4.5.1.3 Fricatives

R	**r**ond	"RO~
S	**ch**amp	"SA~
Z	**g**ens	"ZA~
f	**f**emme	"fam
s	**s**ans	"sA~
v	**v**ent	"vA~
z	**z**one	"zon

A.4.5.1.4 Approximants

H	j**ui**n	"ZHE~
j	p**i**erre	"pjER
l	**l**ong	"lO~
w	c**oi**n	"kwE~

A.4.5.2 Vowels

2	d**eux**	"d2
9	n**eu**f	"n9f
@	just**e**ment	"Zys.t@.mA~
E	s**ei**ze	"sEz
O	c**o**mme	"kOm
a	p**a**tte	"pat
e	s**es**	"se
i	s**i**	"si
o	gr**os**	"gRo
u	d**oux**	"du
y	d**u**	"dy
9~	br**un**	"bR9~

E~	v**in**	"vE~
O~	b**on**	"bO~
A~	v**ent**	"vA~

A.4.5.3 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

A.4.6 Italian (it-IT)

A.4.6.1 Consonants

A.4.6.1.1 Plosives

b	**b**anco	"baN.ko
d	**d**anno	"da.n:o
g	**g**amba	"gam.ba
k	**c**ane	"ka:.ne
p	**p**ane	"pa:.ne
t	**t**anto	"tan.to
b:	go**bb**o	"go.b:o
d:	ca**dd**e	"ka.d:e
g:	fu**gg**a	"fu.g:a
k:	no**cc**a	"no.k:a
p:	co**pp**a	"ko.p:a
t:	zucche**tt**o	t_su."k:e.t:o

A.4.6.1.2 Fricatives

S	**sc**endo	"Sen.do
f	**f**ame	"fa:.me
s	**s**ano	"sa:.no
v	**v**ano	"va:.no
z	**s**baglio	"zba.L:o
S:	a**sci**a	"a.S:a
f:	be**ff**a	"be.f:a
s:	ca**ss**a	"ka.s:a
v:	be**vv**i	"be.v:i

A.4.6.1.3 Affricates

d_z	**z**ona	"d_zO:.na
d_Z	**g**ita	"d_Zi:.ta
t_s	**z**itto	"t_si.t:o
t_S	**c**ena	"t_Se:.na
d_z:	me**zz**o	"mE.d_z:o
d_Z:	o**gg**i	"O.d_Z:i

t_s:	bo**zz**a	"bo.t_s:a
t_S:	bra**cci**o	"b4a.t_S:o

A.4.6.1.4 Approximants

l	**l**ama	"la:.ma
L	**gl**i	"Li
l:	co**ll**a	"ko.l:a
L:	fo**gli**o	"fo.L:o

A.4.6.1.5 Nasals

J	**gn**occhi	"JO.k:i
N	ma**n**ca	"maN.ka
m	**m**odo	"mO:.do
n	**n**ato	"na:.to
J:	guada**gn**a	gu_^a."da.J:a
m:	parti**mm**o	pa4."ti.m:o
n:	mado**nn**a	ma."do.n:a

A.4.6.1.6 Tap, trill

4	pe**r**a	"pE:.4a
r	a**rr**abiata	a."ra.b:i_^a:.ta

A.4.6.2 Vowels

a	rat**a**	"4a:.ta
e	ret**e**	"4e:.te
E	compi**e**ndo	kom."pi_^En.do
i	modern**i**	mo."de4.ni
o	sabat**o**	"sa:.ba.to
u	occ**u**lta	o."k_kul.ta
a:	abit**a**re	a.bi."ta:.4e
e:	ab**e**te	a."be:.te
E:	alfi**e**ri	al."fi_^E:.4i
i:	alp**i**na	al."pi:.na
o:	ard**o**re	a4."do:.4e
O:	mem**o**ria	me."mO:.4i_^a
u:	sal**u**bre	sa."lu:.b4e
i_^	fili**a**le	fi."li_^a:.le
u_^	freq**u**enta	f4e."ku_^En.ta

A.4.6.3 Other Symbols

#	word separator
%	secondary stress
.	syllable break
"	primary stress

B SVOX Pico Text Preprocessing

The following section describes in detail the text preprocessing capabilities built into the lingware of the languages supported by SVOX Pico.

B.1 German (de-DE)

B.1.1 Numbers

B.1.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with up to 2 leading zeros are pronounced as full numbers; more than 2 leading zeros make the number to be pronounced digit by digit.

Examples:

```
123456        hundert drei und zwanzig tausend vier hundert sechs und
fünfzig
1234567       eins zwei drei vier fünf sechs sieben
0012          null null zwölf
00012         null null null eins zwei
```

If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300   hundert elf millionen zwei hundert tausend drei hundert
1’000         tausend
1.456.000     eine million vier hundert sechs und fünfzig tausend
```

B.1.1.2 Ordinal Numbers

Numbers up to 3 digits followed by a period are interpreted as ordinal numbers and pronounced accordingly. Numbers of more than 3 digits are read according to the rules for cardinal numbers, the period being interpreted as a sentence terminator.

Examples:

```
999.          neun hundert neun und neunzigste
1000. Mal     tausend [PAUSE] mal
```

B.1.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23           plus drei und zwanzig
-120          minus hundert zwanzig
```


B.1.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. The part after the decimal point is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

```
1000.1234     tausend punkt eins zwei drei vier
10’400,99     zehn tausend vier hundert komma neun neun
```

B.1.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

```
2/3           zwei schrägstrich drei
123/30        hundert drei und zwanzig schrägstrich dreissig
```

B.1.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

```
der XXVI.     der sechs und zwanzigste
XVIII         achtzehn
```

B.1.1.7 Ranges and Relations

If numbers are separated by a hyphen they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

```
100 - 1000    hundert bis tausend
23 > 12       drei und zwanzig grösser zwölf
100=100.00    hundert gleich hundert punkt null null
```

B.1.2 Numbers with Units

B.1.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

The German TTS system differentiates between feminine and masculine measurement units and pronounces them in singular or plural depending on the number factor applied to the units.

Examples:

```
10km          zehn kilometer
```



```
1 s           eine sekunde
100s          hundert sekunden
1MB           ein megabyte
80 km/h       achtzig kilometer pro stunde
```

B.1.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

```
$20,45        zwanzig dollar fünf und vierzig cent
101,90 CHF    hundert eins schweizer franken neunzig rappen
EUR 1.000.000 eine million euro
```

B.1.3 Dates and Time

B.1.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[6]

d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
MM	-		DD	-		y	12-10-98
d	-		mn	-		y	29-Okt-2000
MMM	.	[W]	d	,	W	y	Nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. Apr

[6] Meaning of the symbols used for describing dates:

```
y   = YY YYYY          year number: 03 or 2003
d   = D DD             day number: 1 or 02
m   = M MM             month number: 1 or 02
mn  = MMM MMMM         abbreviated or full month: Jan or Januar
md  = MMM. MMMM        abbreviated with period or full month: Jan. or Januar
W   = whitespace
[X] = X is optional
```


d	.	[W]	md [[W]	y]	8.Jan.2008 or 04. April 03 or 8.Jan.
d	.	[W]	md [W] , W	y	9.Feb., 1970

Notice that the same input could be read according to different formats depending on its ambiguity. For instance, 12-10-03 could be read in two different ways, taking either 03 or 12 as the year.

Examples:

1970-11-25	fünf und zwanzigste elfte neunzehn hundert siebzig
04/08/98	vierte achte acht und neunzug
8. Jan. 2008	achte januar zwei tausend acht
9.Feb., 1970	neunte februar neunzehn hundert siebzig
1999 18. Apr	achtzehte april neunzehn hundert neun und neunzig

B.1.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[7]

HH	[W]	UNIT1	[W]	MM	23Uhr10 or 10h 25
HH	SEP	MM	[W]	UNIT1	15:59h or 7.35 Uhr
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

10h 25	zehn uhr fünf und zwanzig
15:59 P.M.	fünfzehn uhr neun und fünfzig p m
12:24:03	zwölf uhr vier und zwanzig und drei sekunden
15:59 h	fünfzehn uhr neun und fünfzig

[7] Meaning of the symbols used for describing time indications:

```
UNIT1      = h Uhr
UNIT2      = p P PM p.m. P.M.
UNIT3      =   A AM a.m. A.M.
SEP        = : .
W          = whitespace
[X]        = X is optional
HH, MM, SS = hours, minutes, seconds as number
```


B.1.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but are synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[8]

Common SMS abbreviations and acronyms are also supported.

Examples:

```
hans.huber@svox.com       hans punkt huber at svox punkt com
http://www.svox.com       w w w punkt svox punkt com
http://www.t-online.net/handy w w w punkt t online punkt net schrägstrich
handy
CUL8R                     see you later
b4                        before
```

B.1.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. Brackets, possibly surrounding the area code, and the symbol “+”, which introduces a country code, are spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

```
089 / 44451989            null acht neun [PAUSE]
                          vier vier vier fünf eins neun acht neun

0143-675676               null eins vier drei [PAUSE]
                          sechs sieben fünf sechs sieben sechs

+41 (04) 220-381              plus vier eins [PAUSE] klammer auf [PAUSE]
                          null vier [PAUSE] klammer zu [PAUSE]
                          zwei zwei null [PAUSE] drei acht eins
```

B.1.6 Acronyms and Abbreviations

B.1.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

[8] See later under „Acronyms“ for more details.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

```
ISBN          I s b n
PCMCIA        p c m c I a
DIN           din
UNO           uno
MPEG          m peg
```

B.1.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

```
o. Prof.      ordentlicher professor
mfg           mit freundlichen grüssen
jmdm.         jemandem
i.A.          im auftrag
Hbf           hauptbahnhof
```

B.2 British English (en-GB)

B.2.1 Numbers

B.2.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with the leading zeros are pronounced digit by digit. Numbers in the range 1701-1999 are pronounced as years.

Examples:

```
123456        one hundred and twenty three thousand four hundred and fifty
six
1234567       one two three four five six seven
0012          zero zero one two
1678          one thousand six hundred and seventy eight
1961          nineteen sixty one
```


If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300    one hundred and eleven million two hundred thousand three
hundred
1’000          one thousand
1.456.000      one million four hundred and fifty six thousand
```

B.2.1.2 Ordinal Numbers

Numbers up to 4 digits followed by an ordinal suffix (‘st’,’nd’,’rd’ or ‘th’) are interpreted as ordinal numbers and pronounced accordingly. Numbers of more than 4 digits are read according to the rules for cardinal numbers.

Examples:

```
999th          nine hundred and ninety ninth
5671st         five thousand six hundred and seventy first
```

B.2.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23            plus twenty three
-120           minus one hundred and twenty
```

B.2.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. The part after the decimal point is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

```
1000.1234      one thousand point one two three four
10’400,99      ten thousand four hundred comma nine nine
```

B.2.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

```
2/3            two slash three
12/13          twelve slash thirteen
```

B.2.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

```
XXVI           twenty six
```



```
XVIII          eighteen
```

B.2.1.7 Ranges and Relations

If numbers are separated by a hyphen they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

```
10 - 25        ten to twenty five
23 > 12        twenty three is greater than twelve
100=100.00     one hundred equals one hundred point zero zero
```

B.2.2 Numbers with Units

B.2.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

Examples:

```
10km           ten kilometers
1 s            one second
100s           one hundred seconds
1MB            one megabyte
80 km/h        eighty kilometers per hour
```

B.2.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

```
$20,45                    twenty dollars forty five cents
101,90 CHF                one hundred and one Swiss Francs and ninety
centimes
EUR 1.000.000             one million Euros
```

B.2.3 Dates and Time

B.2.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[9]

[9] Meaning of the symbols used for describing dates:

```
y   = YY YYYY          year number: 03 or 2003
d   = D DD       day number: 1 or 02
```


d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
d	-		mn	-		y	29-Oct-2000
MMM	.	[W]	d	,	W	y	Nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. Apr
d	.	[W]	md	[[W]		y]	8.Jan.2008 or 04. April 03 or 8.Jan.
d	.	[W]	md	[W]	, W	y	9.Feb., 1970

Examples:

1970-11-25	twenty fifth of the eleventh nineteen seventy
04/08/98	fourth of the eighth nineteen ninety eight
8. Jan. 2008	eighth of January two thousand and eight
9.Feb., 1970	ninth of February nineteen seventy
1999 18. Apr	eighteenth of April nineteen ninety nine

B.2.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[10]

```
m   = M MM        month number: 1 or 02

mn  = MMM MMMM    abbreviated or full month: Jan or January

md  = MMM. MMMM   abbreviated with period or full month: Jan. or January

W   = whitespace

[X] = X is optional
```

[10] Meaning of the symbols used for describing time indications:

```
UNIT1      = h

UNIT2      = p P PM p.m. P.M.

UNIT3      =   A AM a.m. A.M.

SEP        = : .

W          = whitespace
[X]        = X is optional

HH, MM, SS = hours, minutes, seconds as number
```


HH	[W]	UNIT1	[W]	MM	23h10 or 10h 25
HH	SEP	MM	[W]	UNIT1	15:59h or 7.35 h
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

```
10h 25                  ten twenty five
15:59 P.M.              fifteen fifty nine p m
12:24:03                twelve twenty four and three seconds
15:59 h                 fifteen fifty nine
```

B.2.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[11]

Common SMS abbreviations and acronyms are also supported.

Examples:

```
jonathan.swift@svox.com   jonathan dot swift at svox dot com
http://www.svox.com       w w w dot svox dot com
http://www.t-online.net/handy w w w dot t online dot net slash handy
CUL8R                     see you later
b4                        before
```

B.2.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. Brackets, possibly surrounding the area code, and the symbol "+", which introduces a country code, are spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

```
089 / 44451989          zero eight nine [PAUSE]
                        four four four five one nine eight nine
```

[11] See later under „Acronyms“ for more details.

0143-675676	zero one four three [PAUSE] six seven five six seven six
+41 (04) 220-381	plus four one [PAUSE] left parenthesis zero four [PAUSE] right parenthesis [PAUSE] two two zero [PAUSE] three eight one

B.2.6 Acronyms and Abbreviations

B.2.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

ISBN	I s b n
PCMCIA	p c m c I a
DIN	din
UNO	uno
MPEG	m peg

B.2.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

Mr.	Mister
Ltd.	limited
RN	Royal Navy

B.3 American English (en-US)

B.3.1 Numbers

B.3.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with the leading zeros are pronounced digit by digit. Numbers in the range 1701-1999 are pronounced as years.

Examples:

```
123456        one hundred twenty three thousand four hundred fifty six
1234567       one two three four five six seven
0012          zero zero one two
1678          one thousand six hundred seventy eight
1961          nineteen sixty one
```

If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300   one hundred eleven million two hundred thousand three
hundred
1’000         one thousand
1.456.000     one million four hundred fifty six thousand
```

B.3.1.2 Ordinal Numbers

Numbers up to 4 digits followed by an ordinal suffix (‘st’,’nd’,’rd’ or ‘th’) are interpreted as ordinal numbers and pronounced accordingly. Numbers of more than 4 digits are read according to the rules for cardinal numbers.

Examples:

```
999th         nine hundred ninety ninth
5671st        five thousand six hundred seventy first
```

B.3.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23           plus twenty three
-120          minus one hundred twenty
```

B.3.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. The part after the decimal point is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

```
1000.1234     one thousand point one two three four
10’400,99     ten thousand four hundred comma nine nine
```


B.3.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

```
2/3           two slash three
12/13         twelve slash thirteen
```

B.3.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

```
XXVI          twenty six
XVIII         eighteen
```

B.3.1.7 Ranges and Relations

If numbers are separated by a hyphen they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

```
10 - 25       ten to twenty five
23 > 12       twenty three is greater than twelve
100=100.00    one hundred equals one hundred point zero zero
```

B.3.2 Numbers with Units

B.3.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

Examples:

```
10km          ten kilometers
1 s           one second
100s          one hundred seconds
1MB           one megabyte
80 km/h       eighty kilometers per hour
```

B.3.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

$20,45	twenty dollars forty five cents
101,90 CHF	one hundred and one Swiss Francs and ninety centimes
EUR 1.000.000	one million Euros

B.3.3 Dates and Time

B.3.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[12]

d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
d	-		mn	-		y	29-Oct-2000
MMM	.	[W]	d	,	W	y	Nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. Apr
d	.	[W]	md	[[W]		y]	8.Jan.2008 or 04. April 03 or 8.Jan.
d	.	[W]	md	[W] ,	W	y	9.Feb., 1970

Examples:

1970-11-25	twenty fifth of the eleventh nineteen seventy
04/08/98	fourth of the eighth nineteen ninety eight
8. Jan. 2008	eighth of January two thousand eight
9.Feb., 1970	ninth of February nineteen seventy
1999 18. Apr	eighteenth of April nineteen ninety nine

[12] Meaning of the symbols used for describing dates:

```
y   = YY YYYY          year number: 03 or 2003
d   = D DD       day number: 1 or 02
m   = M MM       month number: 1 or 02
mn  = MMM MMMM   abbreviated or full month: Jan or January
md  = MMM. MMMM  abbreviated with period or full month: Jan. or January
W   = whitespace
[X] = X is optional
```


B.3.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[13]

HH	[W]	UNIT1	[W]	MM	23h10 or 10h 25
HH	SEP	MM	[W]	UNIT1	15:59h or 7.35 h
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

```
10h 25                  ten twenty five
15:59 P.M.              fifteen fifty nine p m
12:24:03                twelve twenty four and three seconds
15:59 h                 fifteen fifty nine
```

B.3.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[14]

Common SMS abbreviations and acronyms are also supported.

Examples:

```
jonathan.swift@svox.com   jonathan dot swift at svox dot com
http://www.svox.com       w w w dot svox dot com
http://www.t-online.net/handy w w w dot t online dot net slash handy
```

[13] Meaning of the symbols used for describing time indications:

```
UNIT1      = h

UNIT2      = p P PM p.m. P.M.

UNIT3      =   A AM a.m. A.M.

SEP        = : .

W          = whitespace
[X]        = X is optional

HH, MM, SS = hours, minutes, seconds as number
```

[14] See later under „Acronyms“ for more details.

CUL8R	see you later
b4	before

B.3.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. Brackets, possibly surrounding the area code, and the symbol "+", which introduces a country code, are spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

089 / 44451989	zero eight nine [PAUSE] four four four five one nine eight nine
0143-675676	zero one four three [PAUSE] six seven five six seven six
+41 (04) 220-381	plus four one [PAUSE] left parenthesis [PAUSE] zero four [PAUSE] right parenthesis [PAUSE] two two zero [PAUSE] three eight one

B.3.6 Acronyms and Abbreviations

B.3.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

ISBN	I s b n
PCMCIA	p c m c I a
DIN	din
UNO	uno
MPEG	m peg

B.3.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

```
Mr.          Mister
Ltd.         limited
RN           Royal Navy
```

B.4 Spanish (es-ES)

B.4.1 Numbers

B.4.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with the leading zeros are pronounced digit by digit.

Examples:

```
123456       ciento veintitrés mil cuatrocientos cincuenta y seis
1234567      uno dos tres cuatro cinco seis siete
0012         cero cero uno dos
```

If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300  ciento once millones doscientos mil trescientos
1’000        mil
1.456.000    un millón cuatrocientos cincuenta y seis mil
```

B.4.1.2 Ordinal Numbers

Numbers which are no bigger than 31 and followed by an ordinal suffix (‘o’,’a’,’os’,’as’) are interpreted as ordinal numbers and pronounced accordingly. Numbers bigger than 31 are read according to the rules for cardinal numbers.

Examples:

```
31o          trigésimo primero
5a           quinta
```

B.4.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23          signo de más veintitrés
-120         menos ciento veinte
```


B.4.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. If the part after the decimal point is between 1 and 99, then it is pronounced as a full number, otherwise it is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

1000.1234	mil punto uno dos tres cuatro
10’400,99	diez mil cuatrocientos coma noventa y nueve

B.4.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

2/3	dos barra tres
12/13	doce barra trece

B.4.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

XXVI	veintiséis
XVIII	dieciocho

B.4.1.7 Ranges and Relations

If numbers are separated by a hyphen, they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

10 - 25	diez guión veinticinco
23 > 12	veintitrés es mayor que doce
100=100.00	cien es igual a cien punto cero cero

B.4.2 Numbers with Units

B.4.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

Examples:

10km	diez kilómetros
1 g	un gramo
100g	cien gramos

1MB	un megabyte
80 km/h	ochenta kilómetros por hora

B.4.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

$20,45	veinte dólares y cuarenta y cinco céntimos
101,90 CHF	ciento un francos suizos y noventa céntimos
EUR 1.000.000	un millón de euros

B.4.3 Dates and Time

B.4.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[15]

d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
d	-		mn	-		y	29-oct-2000
MMM	.	[W]	d	,	W	y	nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. abr
d	.	[W]	md	[[W]		y]	8.ene.2008 or 04. abril 03 or 8.ene.
d	.	[W]	md	[W] ,	W	y	9. feb., 1970

[15] Meaning of the symbols used for describing dates:

y	= YY YYYY	year number: 03 or 2003
d	= D DD	day number: 1 or 02
m	= M MM	month number: 1 or 02
mn	= MMM MMMM	abbreviated or full month: en, ene or enero
md	= MMM. MMMM	abbreviated with period or full month: en., ene. or enero
W	= whitespace	
[X]	= X is optional	

Examples:

1970-11-25	veinticinco de noviembre de mil novecientos setenta
04/08/98	cuatro de agosto de mil novecientos noventa y ocho
8. en. 2008	ocho de enero de dos mil ocho
9. feb., 1970	nueve de febrero de mil novecientos setenta
1999 18. abr	dieciocho de abril de mil novecientos noventa y nueve

B.4.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[16]

HH	[W]	UNIT1	[W]	MM	10h 25 or 23h10
HH	SEP	MM	[W]	UNIT1	7.35 h or 15:59h
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

10h 25	diez y veinticinco
15:59 P.M.	quince cincuenta y nueve pm
12:24:03	doce horas veinticuatro minutos y tres segundos
12:30 h	doce y media

B.4.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct

[16] Meaning of the symbols used for describing time indications:

```
UNIT1      = h
UNIT2      = p P PM p.m. P.M.
UNIT3      =   A AM a.m. A.M.
SEP        = : .
W          = whitespace
[X]        = X is optional
HH, MM, SS = hours, minutes, seconds as number
```


pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[17]

Common SMS abbreviations and acronyms are also supported.

Examples:

jonathan.swift@svox.com	jonathan punto swift arroba svox punto com
http://www.svox.com	w w w punto svox punto com
http://www.t-online.net/handy	w w w punto t guión online punto net barra handy
CUL8R	see you later
b4	before

B.4.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. The symbol "+", which introduces a country code, is spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

089 / 44451989	cero ocho nueve [PAUSE] cuatro cuatro cuatro cinco uno nueve ocho nueve
0143-675676	cero uno cuatro tres [PAUSE] seis siete cinco seis siete seis
+41 (04) 220-381	signo de más cuatro uno [PAUSE] abrir paréntesis [PAUSE] cero cuatro [PAUSE] cerrar paréntesis [PAUSE] dos dos cero [PAUSE] tres ocho uno

B.4.6 Acronyms and Abbreviations

B.4.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

ISBN	I s b n

[17] See later under „Acronyms“ for more details.


```
PCMCIA          p c m c I a
DIN             din
UNO             uno
MPEG            m peg
```

B.4.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

```
Sra.            señora
izda.           izquierda
p. ej.          por ejemplo
```

B.5 French (fr-FR)

B.5.1 Numbers

B.5.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with the leading zeros are pronounced digit by digit.

Examples:

```
123456          cent vingt-trois mille quatre cent cinquante-six
1234567         un deux trios quatre cinq six sept
0012            zero zero un deux
```

If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300     cent onze millions deux cent mille trois cents
1’000           un mille
1.456.000       un million quatre cent cinquante-six mille
```

B.5.1.2 Ordinal Numbers

Numbers which are no bigger than 31 and followed by an ordinal suffix (‘e’) are interpreted as ordinal numbers and pronounced accordingly. Numbers bigger than 31 are read according to the rules for cardinal numbers.

Examples:


```
31e             trente et unième
5e              cinquième
```

B.5.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23             plus vingt-trois
-120            moins cent vingt
```

B.5.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. If the part after the decimal point is between 1 and 99, then it is pronounced as a full number, otherwise it is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

```
1000.1234       mille point un deux trois quatre
10’400,99       dix mille quatre cents virgule quatre-vingt-dix-neuf
```

B.5.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

```
2/3             deux barre oblique trois
12/13           douze barre oblique treize
```

B.5.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

```
XXVI            vingt-six
XVIII           dix-huit
```

B.5.1.7 Ranges and Relations

If numbers are separated by a hyphen they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

```
10 - 25         dix traitdunion vingt-cinq
23 > 12         vingt-trois est supérieur à douze
100=100.00      cent est égal cent point zero zéro
```


B.5.2 Numbers with Units

B.5.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

Examples:

10km	dix kilomètres
1 s	un seconde
100s	cent secondes
1MB	un mégabyte
80 km/h	quatre-vingts kilomètres par heure

B.5.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

$20,45	vingt dollars quarante-cinq cents
101,90 CHF	cent un francs suisses and quatre-vingt-dix centimes
EUR 1.000.000	un million Euros

B.5.3 Dates and Time

B.5.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[18]

[18] Meaning of the symbols used for describing dates:

```
y   = YY YYYY     year number: 03 or 2003
d   = D DD        day number: 1 or 02
m   = M MM        month number: 1 or 02
mn  = MMM MMMM    abbreviated or full month: Jan or January
md  = MMM. MMMM   abbreviated with period or full month: Jan. or January
W   = whitespace
[X] = X is optional
```


d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
d	-		mn	-		y	29-Oct-2000
MMM	.	[W]	d	,	W	y	Nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. Avr
d	.	[W]	md	[[W]		y]	8.Jan.2008 or 04. Avril 03 or 8.Jan.
d	.	[W]	md	[W]	, W	y	9. Fév., 1970

Examples:

1970-11-25	vingt-cinq novembre mille neuf cent soixante-dix
04/08/98	quatre août mille neuf cent quatre-vingt-dix-huit
8. Jan. 2008	huit janvier deux mille huit
9. Fév., 1970	neuf février mille neuf cent soixante-dix
1999 18. Avr	dix-huit avril mille neuf cent quatre-vingt-dix-neuf

B.5.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[19]

[19] Meaning of the symbols used for describing time indications:

```
UNIT1      = h

UNIT2      = p P PM p.m. P.M.

UNIT3      =   A AM a.m. A.M.

SEP        = : .

W          = whitespace
[X]        = X is optional

HH, MM, SS = hours, minutes, seconds as number
```


HH	[W]	UNIT1	[W]	MM	23h10 or 10h 25
HH	SEP	MM	[W]	UNIT1	15:59h or 7.35 h
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

10h 25	dix heures vingt-cinq
15:59 P.M.	quinze heures cinquante-neuf pm
12:24:03	douze heures vingt-quatre minutes et trois seconds
15:59 h	quinze heures cinquante-neuf

B.5.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[20]

Common SMS abbreviations and acronyms are also supported.

Examples:

jonathan.swift@svox.com	jonathan point swift arobase svox point com
http://www.svox.com	w w w point svox point com
http://www.t-online.net/handy	w w w point t online point net oblique handy
CUL8R	see you later
b4	before

B.5.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. The symbol "+", which introduces a country code, is spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

089 / 44451989	zéro huit neuf [PAUSE] quatre quatre quatre cinq un neuf huit neuf
0143-675676	zéro un quatre trois [PAUSE] six sept cinq six sept six

[20] See later under „Acronyms“ for more details.


```
+41 (04) 220-381    plus quatre un [PAUSE]
                    zéro quatre [PAUSE]
                    deux deux zéro [PAUSE] trois huit un
```

B.5.6 Acronyms and Abbreviations

B.5.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

```
ISBN          I s b n
PCMCIA        p c m c I a
DIN           din
UNO           uno
MPEG          m peg
```

B.5.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

```
Mlle.         Mademoiselle
Hist.         histoire
S.V.P.        s'il vous plaît
```


B.6 Italian (it-IT)

B.6.1 Numbers

B.6.1.1 Cardinal Numbers

Cardinal numbers up to 6 digits are read as full numbers. Cardinal numbers of more than 6 digits as spelled out, that is, the number is pronounced digit by digit. Numbers with the leading zeros are pronounced digit by digit.

Examples:

```
123456        cento ventitre mila quattrocento cinquantasei
1234567       uno due tre quattro cinque sei sette
0012          zero zero uno due
```

If separators (space, apostrophe, comma or period) are used, numbers up to 9 digits are pronounced as full numbers.

Examples:

```
111,200,300   cento undici milioni duecento mila trecento
1’000         mille
1.456.000     un milione quattrocento cinquantasei mila
```

B.6.1.2 Ordinal Numbers

Numbers which are no bigger than 31 and followed by an ordinal suffix (‘o’,’a’,’i’,’e’) are interpreted as ordinal numbers and pronounced accordingly. Numbers bigger than 31 are read according to the rules for cardinal numbers.

Examples:

```
31o           trentunesimo
5a            quinta
```

B.6.1.3 Signed Numbers

Cardinal numbers may be prefixed by the “+” or “-“ symbols.

Examples:

```
+23           più ventitré
-120          meno cento venti
```

B.6.1.4 Floats

Numbers with a decimal part are supported according to the rules for cardinal numbers for the part left to the decimal point. If the part after the decimal point is between 1 and 99, then it is pronounced as a full number, otherwise it is pronounced digit by digit. The decimal point may be written as a period or as a comma.

Examples:

```
1000.1234     mille punto uno due tre quattro
10’400,99     dieci mila quattrocento virgola novantanove
```


B.6.1.5 Fractions

The reading of numbers separated by a slash is supported. However, they are pronounced as if they were read separately.

Examples:

2/3	due barra tre
12/13	dodici barra tredici

B.6.1.6 Roman Numerals

Most roman numerals up to 31 are read correctly. If followed by a period, they are pronounced as ordinals. Several exceptions apply in case of ambiguity or of well known or important acronyms (CD, D, etc.).

Examples:

XXVI	ventisei
XVIII	diciotto

B.6.1.7 Ranges and Relations

If numbers are separated by a hyphen they are read as expressing a range. Cardinals and floats are supported. Relations like “greater than” or “equal to” are also pronounced correctly. Cardinals, floats (eventually signed) are supported.

Examples:

10 - 25	dieci trattino venticinque
23 > 12	ventitré è maggiore di dodici
100=100.00	cento è uguale a cento punto zero zero

B.6.2 Numbers with Units

B.6.2.1 Measurements

Most common measurement units are supported. In addition to these, many other less common units are also pronounced correctly. Some exceptions apply in case of very short (typically one letter long) units, whose reading might be ambiguous or in case of units that conflict with common or important acronyms and abbreviations.

Cardinals, floats (eventually signed and with separators) and fractions are supported.

Examples:

10km	dieci kilometri
1 g	un grammo
100g	cento grammi
1MB	un megabyte
80 km/h	ottanta kilometri all'ora

B.6.2.2 Currencies

Most common currency units are pronounced properly. Besides the standard currency codes, currency symbols are also read correctly if available for a certain currency. Cardinals and floats (eventually with separators) are supported. The currency unit may be written before or after the number factor.

Examples:

$20,45	venti dollari e quarantacinque centesimi
101,90 CHF	centoun franchi svizzeri e novanta centesimi
EUR 1.000.000	un milione di euro

B.6.3 Dates and Time

B.6.3.1 Dates

Several different formats for reading dates are supported. The following is a list of all the formats with samples:[21]

d	.	W	m	.	W	YYYY	23. 03. 2005
DD	.		MM	.		[y]	04.08.[98]
DD	/		MM	/		y	04/08/98
D	.		M	.		[y]	8.7.[85]
D	/		M	/		y	8/7/85
m	/		d	/		y	03/05/03 or 3/5/03
y	-		MM	-		DD	1970-11-25
d	-		mn	-		y	29-oct-2000
MMM	.	[W]	d	,	W	y	nov. 2, 1980
YYYY		W	d	.	[W]	mn	1999 18. apr
d	.	[W]	md	[[W]		y]	8.gen.2008 or 04. aprile 03 or 8.gen.
d	.	[W]	md	[W] ,	W	y	9. feb., 1970

Examples:

1970-11-25	venticinque novembre mille novecento settanta
04/08/98	quattro agosto mille novecento novantotto
8. gen. 2008	otto gennaio due mila otto
9. feb., 1970	nove febbraio mille novecento settanta
1999 18. apr	diciotto aprile mille novecento novantanove

[21] Meaning of the symbols used for describing dates:

```
y   = YY YYYY     year number: 03 or 2003
d   = D DD        day number: 1 or 02
m   = M MM        month number: 1 or 02
mn  = MMM MMMM    abbreviated or full month: gen, genn or gennaio
md  = MMM. MMMM   abbreviated with period or full month: gen., genn. or gennaio
W   = whitespace
[X] = X is optional
```


B.6.3.2 Time

Several different formats for reading time indications are supported. The following is a list of all the formats with samples:[22]

HH	[W]	UNIT1	[W]	MM	10h 25 or 23h10
HH	SEP	MM	[W]	UNIT1	7.35 h or 15:59h
HH	SEP	MM	[W]	UNIT2	11.12PM or 6:50 P.M.
HH	SEP	MM	[W]	UNIT3	11.12 a.m.
HH	:	MM	[:	SS]	12:34 or 12:34:03
HH	.	MM		am	8.48am

Examples:

10h 25	ore dieci e venticinque
15:59 P.M.	ore quindici e cinquantanove pm
12:24:03	ore dodici e ventiquattro e tre secondi
12:30 h	ore dodici e trenta

B.6.4 E-mail Addresses, URLs and SMS Abbreviations

The system is able to recognize the format of E-mail addresses and URLs and to read them correctly. One limitation should be highlighted: currently no graphotactic analysis of the input has been implemented and therefore it is possible that impossible sequences for G2P are not spelled as they should but synthesized. However, a system of rules and exceptions has been built around this temporary limitation, which should allow for the correct pronunciation of at least most common words (typically domain names) embedded in E-mail addresses and URLs.[23]

Common SMS abbreviations and acronyms are also supported.

Examples:

jonathan.swift@svox.com	jonathan punto swift chiocciola svox punto com
http://www.svox.com	vuvuvu punto svox punto com

[22] Meaning of the symbols used for describing time indications:

```
UNIT1      = h

UNIT2      = p P PM p.m. P.M.

UNIT3      =   A AM a.m. A.M.

SEP        = : .

W          = whitespace
[X]        = X is optional

HH, MM, SS = hours, minutes, seconds as number
```

[23] See later under „Acronyms“ for more details.


```
http://www.t-online.net/handy vuvuvu punto t trattino online punto net
                              barra handy
CUL8R                     see you later
b4                        before
```

B.6.5 Phone Numbers

Telephone numbers are recognized and pronounced according to a few simple rules. The general rule is that a phone number is read digit by digit. This applies also to country and area codes. The symbol "+", which introduces a country code, is spelled. Symbols used for separating groups of digits (the slash, the hyphen or whitespace) are not pronounced; instead a short pause is generated. The longest sequence of digits without any separator that is allowed for an input sequence to be recognized as a telephone number is nine.

Examples:

```
089 / 44451989      zero otto nove [PAUSE]
                    quattro quattro quattro cinque uno nove otto nove
0143-675676         zero uno quattro tre [PAUSE] sei sette cinque sei sette
                    sei
+41 (04) 220-381    più quattro uno [PAUSE]
                    zero quattro [PAUSE]
                    due due zero [PAUSE] tre otto uno
```

B.6.6 Acronyms and Abbreviations

B.6.6.1 Acronyms

As a general rule, acronyms of two or three letters written all in uppercase are spelled. If they are longer than three letters, they are processed by general G2P mechanisms in the system.

However, several exceptions apply: depending on the readability and conventions in the respective language, some two and three letters acronyms are pronounced as normal words. Also, some longer acronyms, which would be otherwise read as normal words, are spelled, and some other sequences that would not fall into the recognized class of acronyms, because they contain for instance lowercase letters, are spelled.

Currently, no graphotactic analysis is done that would allow for a systematic readability test. Therefore, it is possible for unpronounceable words to be eventually processed by the general G2P functionality.

Examples:

```
ISBN          I s b n
PCMCIA        p c m c I a
DIN           din
UNO           uno
MPEG          m peg
```

B.6.6.2 Abbreviations

A big number of common abbreviations is recognized and pronounced correctly by the system. Some of the abbreviations are potentially ambiguous and they are pronounced depending on the context in which they are embedded.

Punctuation marks are allowed inside abbreviations and do not generate any additional prosody if the abbreviation was recognized successfully. The recognition is flexible, allowing in some cases optional whitespace or even punctuation marks.

Abbreviations unknown to the system are first checked against the rules for acronyms described above and eventually spelled; otherwise they are processed by the general G2P mechanisms built into the system.

Examples:

```
Sig.          signor
ecc.          eccetera
p. es.        per esempio
```


C SVOX Pico SDK Installation

C.1 Introduction

The SVOX Pico SDK consists of several installation packages grouped as follows:

- **Base package** (required): The base package contains the SVOX Pico TTS engine, the SVOX Pico API, the SVOX Pico test program binary, and C source code for example applications. The file naming scheme for base packages is
  `pico_<version-subrevision>_base_<platform>-<date>.zip`
  All SVOX Pico product types share the same platform-dependent base package on a specific hardware/OS platform.
- **Lingware packages** (one or more required): Each SVOX Pico lingware package contains resource files for one voice of a specific language. The file naming scheme for a standard lingware packages is
  `pico_<bver>_lw_<lang>_<ptid>_<sf>_<speakerid>_<bsub>_<ver>-<sub>.zip`
  where `<lang>` is the language identification (according to RFC 3066, ISO 639), `<ptid>` is the product type identifier, and `<sf>` is the sampling frequency in kHz. `<bver>` stands for the reqired base version, `bsub` for the minimally required base subrevision. For a specific product type and voice, the same platform-independent lingware package can be used on all hardware/OS platforms.
  Note: For customized lingware packages an extended naming structure will be applied.
- **Lingware extension packages** (optional): The optional lingware extension packages extend the functionality of SVOX Pico lingware packages. The file naming scheme for lingware extension packages is
  `pico_<version>_lwx_<xtype>_<xid>-<date>.zip`
  where `<xtype>` is the extension type and `<xid>` is an extension identifier that further describes the content of the package. The same platform-independent lingware extension package can be used on all hardware/OS platforms.

The base package and at least one of the lingware packages are needed to install the SVOX Pico SDK . Optionally, one or more extension packages and additional lingware packages can be installed.

Several pre-packaged voices and languages are available for the SVOX Pico TTS engine. For productive versions of SVOX Pico, additional voices and languages are available from SVOX upon request.

C.2 Installation on Windows and Unix

C.2.1 Installation

To install the SVOX Pico SDK on Windows 2000/XP/Vista or Unix, carry out the following steps:

- Check the availability of all package components, as described in C.2.2.
- Unzip all files of the base package and of one or more lingware packages (using e.g. "WinZip" or "unzip") into an empty directory of your choice, e.g. `d:\pico\`. (or `/usr/local/pico/` on Unix). All files are extracted into a subdirectory named "`pico_xxx`" where "`xxx`" is the version number, for instance "`pico_100`".
- Optionally, extract the files of needed lingware extension packages into the same directory.
- Test the installation by running the `picosh` binary test application. Refer to chapter 2 for a detailed application description.

C.2.2 SDK Contents

Run-time environment files and tools

- `picodyn.dll` : dynamic library containing the SVOX Pico engine (Win32 only)
- `picosh.exe` : the SVOX Pico test program binary executable (named `picosh` on Unix)
- `*_ta_*.bin` , `*_sg_*.bin` : the lingware packages for the current language

Files needed to create, compile, and link your application

- `picoapi.h` : header file of the SVOX Pico API definition
- `picodefs.h` : header file of the SVOX Pico API constants
- `picodyn.lib` : DLL description, to properly link C/C++ application programs (Win32 only)
- `libpico.a` : link library containing the SVOX Pico engine software (Unix only)

Example source code file

- `testpico.c` : example application using the SVOX Pico API

C.2.3 Compiling and linking C/C++ applications of SVOX Pico

On Windows

The files `picoapi.h, picodefs.h` and the DLL definition `picodyn.lib` should be included in the application project in order to properly compile and link the application.

On Unix

In C/C++ applications, the files `picoapi.h and picodefs.h` must be included in the application program. In order to link the application with the SVOX software, the library `libpico.a` must be included in the list of searched libraries. The following example shows how the application `testpico.c` can be compiled and linked using `gcc`, under the assumption that the SVOX Pico system is located in the directory `/usr/local/pico/pico_100`:

```
gcc –o testpico testpico.c  –L /usr/local/pico/pico_100  -lpico_100 –lm
```


C.2.3.1 Build and test the application

In the file `testpico.c`, adapt the values for the `#define` values `RESOURCE_NAME_DE_SI`, `RESOURCE_NAME_DE_SD`, to the respective names of the lingware shipped to you.

Using the project file and the selected compiler, build the executable version of the testpico application and then launch it, either in the debug or release version type of build. The `testpico` application does the following:

- Allocates the needed memory
- Initializes the runtime System object.
- Creates a voice definition “Susanne” and loads the needed resources
- Creates a new engine with the voice “Susanne”.
- Starts the loop of putting text/getting data until
    - A)The text is complete
    - B)The engine returns an idle state
- Disposes the engine
- Unloads the resources
- Frees the allocated memory

The text sent to the engine in the example is as follows

```
<genfile file=\"test.wav\">Hallo world.</genfile>
```

This includes a text markup command “genfile” (cfr 8.2) inside the text string. The real text to be synthesized is “Hallo world”. This text markup instructs the engine to store the synthesis output on a wav file.

Upon exit, on the current directory, a new wav file has to be found with name “test.wav”. Sampling frequency is 16kHz, number of bits is 16. The audio content is a female voice saying the sentence “Hallo world”. If this happens, then Pico has been installed and programmatically tested successfully on Your platform.

C.3 Installation for Symbian Development on Windows

C.3.1 Installation

Before installing the SVOX Pico SDK , ensure that a Symbian Series 60 development environment is installed on your Windows development host (except for the GUI-based test application the SVOX Pico SDK can also be used with Series 80 development environments).

To install the SVOX Pico SDK for Symbian on your Windows 2000/XP/Vista host, carry out the following steps:

- Remove any previously installed versions of the SVOX Pico SDK from your Symbian target device and the development environment (simply by removing the SVOX Pico packages and files on your Symbian device and development host).
- Unzip all files of the base package and of one or more lingware packages (using e.g. "WinZip" or "unzip") into an empty directory of your choice, e.g. `S:\pico-symbian\`. All files are extracted into a subdirectory "`pico_xxx`" where "`xxx`" is the package version number, for instance "`pico_100`".
- Note: in order to compile the SVOX Pico test application, the installation directory has to be located on the same drive as the Symbian development environment. Furthermore, the full path name of the installation directory should not contain any spaces.
- Setup of Symbian development environment. WARNING: path names shown on the following example may be different, depending on the operating system and compiler versions You use. Also compiling for the runtime or for the emulator may originate different path names and library extensions.
- An example of environment setup is given in the following:
    - Create a subdirectory `pico_100\` in the `epoc32\wins\c\system\apps\` in your Symbian development environment and copy the files
      `pico_100\*.bin`
      into this newly created directory.
    - Copy the files
      `pico_100\wins\picodyn.lib`
      `pico_100\wins\picodyn.dll`
      into the `epoc32\release\wins\udeb\` directory located in your Symbian development environment.
    - Copy the file
      `pico_100\thumb\picodyn.lib`
      into the `epoc32\release\thumb\urel\` directory located in your Symbian development environment.

C.3.2 SDK Contents

Run-time environment files for the Epoc32 emulator

`wins\picodyn.dll`	dynamic link library containing the SVOX TTS engine software
`*.bin`	data needed in the actual synthesis process (lingware file)

Run-time environment files for the target phone

`thumb\picodyn.dll`	dynamic link library containing the SVOX TTS engine software

OR

`thumb\picodyn.dso`	dynamic link library containing the SVOX TTS engine software when gcc compiler.
`*.bin`	data needed in the actual synthesis process (lingware file)

Files needed to compile, and link your application

`wins\picodyn.lib`	DLL description, to properly link C/C++ application programs for the Epoc32 emulator
`thumb\picodyn.lib`	DLL description, to properly link C/C++ applications for the target phone
`picoapi.h`	header file of the SVOX Pico TTS API definition
`picodefs.h`	header file of the SVOX Pico defines and constants

Test application source code files

`testpico\readme.txt`	build instructions for the test application
`testpico\*.*`	test application using the SVOX Pico TTS API

C.3.3 Compiling and Linking C/C++ Applications of SVOX

The files `picoapi.h, picodefs.h` and the DLL definition `picodyn.lib` should be included in the application project in order to properly compile and link the application.

C.4 Installation for Windows CE 5.0 Development on Windows

C.4.1 Installation

Before installing the SVOX Pico SDK , ensure that a development environment for Windows CE 5.0 or higher is installed on your Windows development host.

To install the SVOX Pico SDK for Windows CE on your Windows 2000/XP/Vista host, carry out the following steps:

- Remove any previously installed versions of the SVOX Pico SDK from your Windows CE target device and the development environment (simply by removing the SVOX Pico packages and files on your Windows CE device and development host).
- Unzip all files of the base package and of one or more lingware packages (using e.g. "WinZip" or "unzip") into an empty directory of your choice, e.g. `S:\pico-wince\`. All files are extracted into a subdirectory named "`pico_xxx`" where "`xxx`" is the version number, for instance "`pico_100`".

C.4.2 SDK Contents

Run-time environment files for x86 and ARM

`x86\picodyn.dll`	dynamic link library containing the SVOX Pico TTS engine software (x86)
`ARM\picodyn.dll`	dynamic link library containing the SVOX Pico TTS engine software (ARM)
`*.bin`	data needed in the actual synthesis process (lingware file)

Files needed to create, compile, and link your application

`x86\picodyn.lib`	DLL description, to properly link C/C++ applications for x86
`ARM\picodyn.lib`	DLL description, to properly link C/C++ applications for ARM
`picoapi.h`	header file of the SVOX Pico TTS API definition
`picodefs.h`	header file of the SVOX Pico TTS defines and constants

Example source code files

`testpico\*.*`	example application using the SVOX TTS API

C.4.3 Compiling and Linking C/C++ Applications of SVOX

The files `picoapi.h, picodefs.h` and the import library `picodyn.lib` should be included in the application project in order to properly compile and link the application.

C.4.4 Build and test the application

In the file `testpico.c`, adapt the values for the `#define` values `RESOURCE_NAME_DE_SI`, `RESOURCE_NAME_DE_SD`, to the respective names of the lingware shipped to you.

Using the project file and the selected compiler, build the executable version of the testpico application, either in the debug or release version type of build

The test program can be made run on PocketPC devices. To install the program and the required run-time files, proceed as follows:

1. Create a folder of your choice on your PocketPC device (e.g. `\pico`).
2. Copy `tespico.exe`, `*.bin` (and any additional lingware files needed) to the folder you created in step 1.
3. Copy `picodyn.dll` to the `\Windows` folder on your PocketPC device.
4. Create a shortcut to `tespico.exe` in the `\Windows\Start Menu` folder.

Now you can run the test application by selecting it from the start menu. The `testpico` application does then following:

- Allocates the needed memory
- Initializes the runtime System object.
- Creates a voice definition “Susanne” and loads the needed resources
- Creates a new engine with the voice “Susanne”.
- Starts the loop of putting text/getting data until
  - A)The text is complete
  - B)The engine returns an idle state
- Disposes the engine
- Unloads the resources
- Frees the allocated memory

The text sent to the engine in the example is as follows

```
<genfile file=\"test.wav\">Hallo world.</genfile>
```

This includes a text markup command “genfile” (cfr 8.2) inside the text string. The real text to be synthesized is “Hallo world”. This text markup instructs the engine to store the synthesis output on a wav file.

Upon exit, on the `\pico` directory, a new wav file has to be found with name “test.wav”. Sampling frequency is 16kHz, number of bits is 16. The audio content is a female voice saying the sentence “Hallo world”. If this happens, then Pico has been installed and programmatically tested successfully on Your platform.

