# Verbio Software Reference

## User's Guide

**Verbio Technologies, S.L.**

# Verbio Software Reference: User's Guide

Verbio Technologies, S.L.

Published September 2009

# Table of Contents

# List of Tables

# List of Examples

# Chapter 1. Introduction

The most natural form for an individual to communicate has been, and will always be, the voice. Therefore, it is no wonder that during these recent years great efforts have been done to develop techniques that enable extending this communication beyond interpersonal field. At present, highly acceptable results have been achieved for a wide range of applications. That is why the market is starting to offer products that include speech technologies in order to facilitate, automate or make human-machine interaction more natural .

## 1. What is Verbio?

*Verbio* is a set of libraries and utilities intended for achieving a quick and simple incoroporation of speech tools (recognition and synthesis) in those applications in which it might be interesting to have a vocal interface.

Therefore, *Verbio* incorporates speech recognition and synthesis functionalities, the main characteristics of which are described in chapters Chapter 4, *Speech recognition* and Chapter 5, *Text-to-speech conversion* respectively.

Any environment including a device that enables obtaining and/or reproducing audio samples will be capable of incorporating the tools contained in `Verbio`. On one hand, the recognition system will require obtaining the audio samples pronounced by the speaker in order to process them and obtain the recognition result. On the other hand, the voice synthesis system will require reproducing the audio samples that have been generated from the introduced text.

## 2. Using this guide

The aim of this guide is to serve as link for all documentation included in `Verbio`. For that purpose, it deals with all necessary subjects that lead to a better understanding and optimum use of the resources provided by `Verbio`, pointing out specific documents when convenient. Therefore, it is advisable to read it before proceeding to application development as it contains important information that might be useful to avoid future problems or, more importantly, unefficient use of current voice recognition and synthesis systems.

Chapter 2, *Initializing, configuration and licenses* discusses the most important aspects when it comes to installing and configuring the server (or servers) that constitute *Verbio* engine. The recomended architecture is that based on `client-server` architecture, as it facilitates the system maintenance and redundancy. Likewise, this chapter also comprises information related to the system features and resource consumption, since both issues are of special interest when dimensioning the `hardware` to be used.

Chapter 3, *Platforms, tools and development kits* gives detail of the tools that the developer comprises in order to incorporate solutions for speech synthesis and recognition in its applications. It is important to find the most appropriate `SDK` in each case, as this will imply greater simplicity and speed in implementation. Moreover,`Verbio` includes a set of tools that are very useful for development, tests and later maintenance of developed applications.

In Chapter 4, *Speech recognition* you will find the aspects that need to be taken into account when using speech recognition tools. In particular, special stress is made on two issues that are crucial for an optimum use of the recognizer: creating recognition grammars and interpreting results returned by the recognizer correctly.

In Chapter 5, *Text-to-speech conversion* you will find the aspects to be considered when using speech synthesis tools. In particular, aspects such as sporadic changes in `TTS` behaviour and specification of abbreviations and acronyms are to be taken into account for a correct reproduction of the specified text.

## 3. Further information

Documentation on the various SDK:

- Verbio Software Reference: Reference to Advanced SDK functions
- Verbio Software Reference: Reference to Library SDK functions
- Verbio Software Reference: Reference to Dialogic SDK functions

- Verbio Software Reference: Reference to CT-ADE SDK functions
- Verbio Software Reference: Reference to Avaya IR SDK functions

Development tools:

- Verbio Software Reference: Grammar Manager User's Guide
- Verbio Software Reference: Read Aloud User's Guide

Contact information:

In the event of detecting errors in the documentation provided with any of `Verbio` versions or in case of needing additional support for application development, do not hesitate to contact our technical department on *support@verbio.com*.

In the event of requiring additional information related to prices, licenses or any other commercial issue, you can contact our commercial department on *info@verbio.com*.

# Chapter 2. Initializing, configuration and licenses

## 1. Prerequisites

At present, *Verbio* products are only available for `Windows 2000/XP` operative systems , although a version for Linux is now on trial period. Before `software` installation it is convenient to verify that the `hardware` being used meets the necessary basic requisites in order to cope with the computational and memory load used by *Verbio* speech recognition and synthesis systems. In Section 4, “Hardware requirements” in this section we give detail of the necessary aspects to dimension one or more systems according to particular work conditions: recoginition grammar size, speakers in use, etc.

Using speech recognition and speech synthesis systems and tools that compose the range of products comprised under the trade name *Verbio*, requires installing a set of modules distributed in independent installers. This way, it is only necessary to download and install, in the appropriate order, modules that are strictly necessary. The aim of this document is to serve as guide to installers so that they can determine the necessary modules and they can proceed to installation. Modules can be classified in accordance to various criteria, although the main subdivisions could be:

- Mandatory modules and optional modules
- Server modules and client modules
- Speech recognition modules and speech synthesis modules

## 2. Mandatory modules

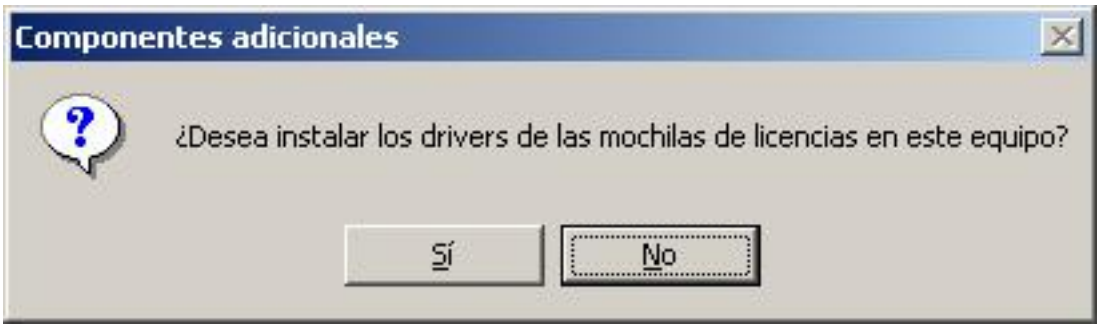
*Verbio* speech recognition and synthesis engines have been created to work in a client-server structure. In other words, it is necessary to have one or more services (`servers`) to deal with the requests of one or more clients. By client we refer to any application that uses speech recognition and/or synthesis resources. Particularly, those machines that contain audio device (CTI or sound card). This structure enables creating an easily scalable and redundant work environment. Client and server can function on the same machine in low computational load environments or in those that require so.

### 2.1. Server components (`Verbio Server Engine`)

It needs to be installed in those machines intended for hosting a `server` (only one`server`per machine is allowed), whether it is a speech recognition server, a speech synthesis one or both at the same time. To install `Verbio Server Engine`, the installer**VerbioEngines.exe** needs to be exectued. Once the installation directory has been selected, a selection window will appear with the components that need to be installed. If in the current machine only the server needs installing, solely the option `Components` needs to be selected from the `server.` In case that client and server are wanted in the same machine, both components need to be selected.


Selection of`Server Components`and/or `Client Components`.

Once *Verbio* specific components have been installed, the installer checks that in the system there is the license key`driver` (or `dongle` or `sentinel` or bitlock) required to put the service in operation. Keys are`hardware` devices that need to be connected to the parallel port or USB in the machine(s) where the server(s) is/are hosted. To obtain the keys you should contact your usual provider. In case that the `driver` is not installed, you have the chance to do so.




Installation option for license key `driver` if it has not been installed previously.

In case that you proceed to installation, default options need to be selected.

### Caution

In the event that there is some key inserted in the equipment, it is necessary to remove it before proceeding to `driver` installation. Otherwise, the information contained in the key might be lost.

## 2.2. Client Components (`Verbio Client Engine`)

It needs to be installed in every machine intended for hosting one or more`clients`, regardless of the sort of

transaction to be done.

To install `Verbio Client Engine`, the installer **VerbioEngines.exe** needs to be executed . Once the installation directory has been selected, a selection window will appear with the components that need to be installed. If in the current machine only the client needs installing, solely `Client components` will be selected. In case that client and server are wanted in the same machine, both components need to be selected.

# 3. Optional modules

## 3.1. Server components

Once the mandatory server modules have been installed as mentioned above, necessary optional components will need installing. *Verbio* distributes speech recognition modules (`configurations`) and speech synthesis modules (`speakers`). Therefore, depending on the languages to be supported by the recognition engine, it will be necessary to install one or more `configurations`. Likewise, for each speaker used, it will also be necessary to install its appropriate`speaker`.

### 3.1.1. Recognition configruations

Each `configuration`enables including one ore more languages in the speech recognition server in which it is installed. Therefore, it will only be possible to recognize sentences in the languages for which the appropriate `configuration` has been installed. Before using a vocabulary (preparation, activation, recognition, etc.) it is necessary to activate the `configuration` linked to the work language (by means of the appropriate programming `API` function), so it might be necessary to alternate it for those applications that are multilingual.

A particular case is that of multilingual `configurations` that enable using multilingual vocabularies and/or grammars; that is, vocabularies and/or grammars that include entries in various languages. If that is the case, it is not necessary to modify the `configuration` when changing from one language to another (provided that they all are included in the selected `configuration`), but it is enough to point to the system the default work language. Using these `configurations` is very useful when it comes to situations when the language that the users will use is unknown. It is specially useful for individual names, towns, etc in areas that are linguistically plural. Currently, only multilingual`configurations` of the oficial languages in Spain are available. For more information related to working with multilingual vocabularies and/or grammars, check Section 3.1.1.1, “Language”.

The following table shows the `configurations` that need to be installed in order to support the various languages available in *Verbio*, paying special attention to cases that need multilingual `configurations`.

**Table 2.1. Necessary configurations (ASRConfiguration_)**

| Spanish | Catalan | Basque | Galician | Portuguese | Brazilian | French | English | Mexican | Installer |
|---|---|---|---|---|---|---|---|---|---|
| y | n | n | n | - | - | - | - | - | es |
| o | y | n | n | - | - | - | - | - | es_ca |
| o | n | y | n | - | - | - | - | - | es_eu |
| o | n | n | y | - | - | - | - | - | es_ga |
| y/o | y/o | y/o | y/o | - | - | - | - | - | es_ca_eu_ga |
| - | - | - | - | y | - | - | - | - | pt |
| - | - | - | - | - | y | - | - | - | pt-br |
| - | - | - | - | - | - | y | - | - | fr |
| - | - | - | - | - | - | - | y | - | en-us |
| - | - | - | - | - | - | - | - | y | es-mx |

(y) required, (n) non-required, (-) indifferent, (o) optional

### 3.1.2. Synthesis speakers

To use a particular speaker in the synthesis server, it is essential to install its appropriate `speaker`. Depending on the languages that you want to support, one or more of the speakers available for each language need to be installed. By means of the appropriate programming`API`function you will need to select the appropriate language for every case (and in some occasions, the appropriate speaker), according to the application needs.

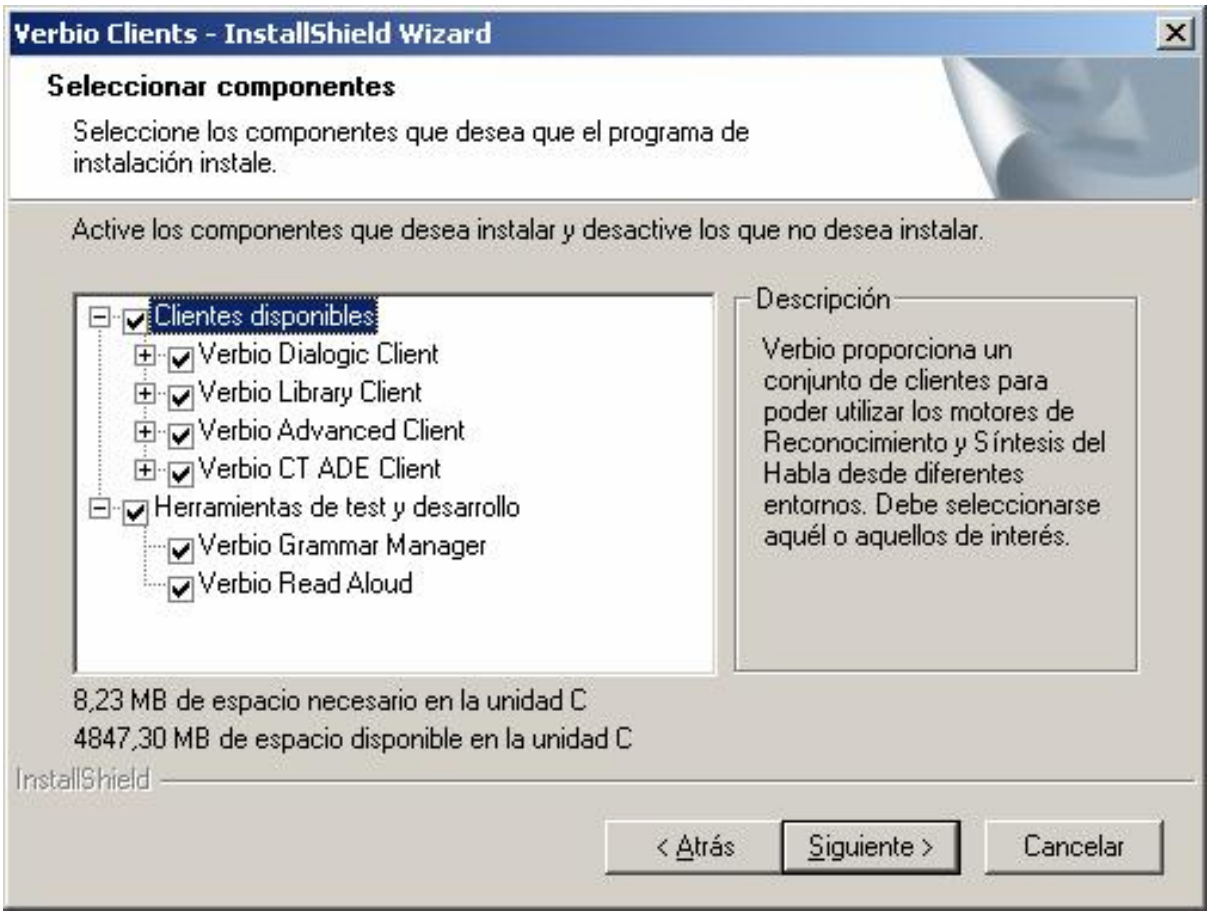
The following table shows the `speakers` available in *Verbio* for each of the supported languages.

**Table 2.2. Necessary speakers**

| Language | Sex | Age | Name | Installer |
|---|---|---|---|---|
| castilian Spanish | Male | Adult | Carlos | Spanish-Carlos |
| castilian Spanish | Female | Adult | Laura | Spanish-Laura |
| Catalan | Male | Adult | Pau | Catalan-Pau |
| Catalan | Female | Adult | Marta | Catalan-Marta |
| Basque | Male | Adult | Jon | Basque-Jon |

## 3.2. Client components

Within the additional modules for the client, included in **VerbioDeveloper.exe**, the following components are found:




Set of components (clients and tools) available for the client system.

### 3.2.1. Application development platforms (types of clients)

*Verbio* provides a set of development platforms so that multiple environment integrators can incorporate speech

technologies to their products. For each of the available platforms we provide:

- `Software Development Kit (SDK):` Kit of files intended for programming new applications that include speech recognition or synthesis (libraries, header, etc.).
- `Examples:` Example codes for SDK use.
- `Documentation:` Description of the functions contained in the SDK (`Function Reference`).

Deciding which `SDK` to use depends on various factors that are described below with the purpose that each integrator finds the most suitable `SDK` according to its programming environment and knowledges. The following table is a summary of the particularities of every`SDK`. More information can be found in Section 1, “Software and hardware platforms” .

**Table 2.3. Selecting the SDK**

| SDK | programming environment | Used hardware | Compatibility | Advanced tools |
|---|---|---|---|---|
| Advanced | Microsoft Visual Studio C/C++ | Any CTI card or I/O device | Incompatible with previous versions | Yes |
| Dialogic | Microsoft Visual Studio C/C++<br>Other C/C++ compilers | CTI Intel Dialogic cards | Compatible with Verbio Dialogic (vx_) | No |
| Library | Microsoft Visual Studio C/C++<br>Other C/C++ compilers | Any CTI card or I/O device | Compatible with Verbio Library (vox_) | No |
| CT ADE | Graphical VOS (Parity) - CT ADE | CTI Intel Dialogic cards | Compatible with previous IVR versions | No |

*Verbio* includes the `Advanced` development platform to facilitate the use of speech recognition and synthesis engines, specially recognition of the new built-in functionalities. The remaining platforms have been updated so they can enjoy the new *Verbio* features without losing compatibility with the previous versions, even though this means that they are slighly more complex than the new platform.

Advanced tools (only available in the `Advanced`development platform) affect synthesis as well as recognition, although the latter has been worked on more deeply, specially when it comes to processing the results returned by the recognition system.

### 3.2.2. Testing and development tools

- `Verbio Grammar Manager:`Tool intended for designing and testing recognition grammars used in the application. For further information consult `Verbio Software Reference: Grammar Manager User's Guide`.
- `Verbio Read Aloud:` Tool intended for testing the available speakers by specifying any text and by manipulating its basic parameters in operation. For further information consult `Verbio Software Reference: Read Aloud User's Guide`.

Installing testing and development tools does not require installing any of the type of clients that accompany

them in the installer, it only requires installing the mandatory client components. More information on any of these tools can be found in the appropriate documentation (provided with installation).

### 3.2.3. VoiceXML interpreter

Inside the additional modules for the client, *Verbio* also includes an interpreter based on the standard`VoiceXML`: `VerbioVoiceXML`, contained inside **VerbioVoiceXML.exe** installer.

This tool is suitable for all those developers and integrators that require a development environment which is simple (it does not require programming at a low level), flexible (it does not require knowing technical aspects on the phone hardware used) and fully integrated with *Verbio* speech technologies. For further information, consult product documentation or support information in http://www.verbio.com.

Installation only requires specifying the installation directory (it can be different from that in which *Verbio* mandatory components have been installed) and specifying whether you wish to install examples and product documentation.


*Verbio VoiceXML* installation process.

# 4. Hardware requirements

When determining the necessary equipment(s) for each installation, it is advisable to take into account the aspects that are of greater influence on `hardware` resource consumption. Among the most determinant aspects the most significant are the size of the vocabulary, the speakers being used and distribution of computational load (design strategy).

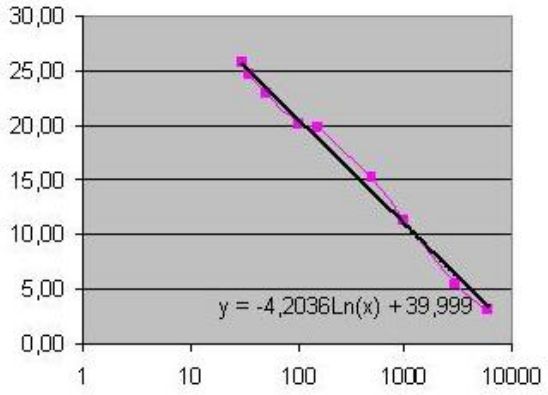
## 4.1. Size of the vocabulary

The size of the vocabularies or grammars being used is, within consumption deriving from the use of speech technologies, the aspect that has the greatest influence on the load supported by the processor. Therefore, the bigger the size of the active grammar is (or grammars if various of them are used in parallel), the greater is `CPU`

consumption. The following table can serve as guide for dimensioning the `hardware`, although the aspects described in the following sections should also be considered. The data shown has been obtained in a 2,4 GHz Intel Pentium IV server.

**Table 2.4. Simultaneous requests according to the size of the vocabulary**

| Size of the vocabulary | Simultaneous requests |
|---|---|
| 30 | 26 |
| 35 | 25 |
| 50 | 23 |
| 100 | 21 |
| 150 | 20 |
| 500 | 15 |
| 1000 | 11 |
| 3000 | 6 |
| 6000 | 3 |




Simultaneous requests according to the size of the vocabulary (in words) for a 2,4 GHz Pentium IV.

By`vocabulary word` we refer to any recognition unit that is part of the vocabulary or grammar file; that is, a unit that can be recognized in an isolated way. For instance, in a vocabulary including names+surnames, `vocabulary words` would be made up by the name plus the surname. More information on vocabularies and grammars can be found in Section 3, “Recognition grammars”.

These results are obtained by forcing an immediate answer time. That is, by not allowing any delay between the end of the phrase and the obtention of the recognition result. In non-critical application as for answer time, the number of simultaneous ports could be slighly increased without noticing any deterioration in the service quality.

In some cases, however, it is possible to break down big-sized vocabularies and/or grammars into subvocabularies and/or subgrammars that, despite having the same features, consume less resources.

## 4.2. Speakers being used

Using speech synthesis consumes a significant amount of RAM memory from the system, so memory will need to suit the type and amount of users being used. *Verbio* enables storing almost all synthesizer data in disks so that the amount of memory required is reduced, although this means that answer time will be slighly longer (specially in overloaded environments). Therefore, it is advisable to use this strategy only in those environments that require few simultaneous synthesis transactions (around 5) and have unavoidable memory limitations. Memory consumption due to speaker activation is high, but it remains constant during the whole usage period, regardless of requests being made. This way, memory required is conditioned by the amount of speakers used and not by the amount of simultaneous requests that the synthesizer should be able to process. As starting point, for a 2-speaker standard use, it is advisable to use 512MBytes of RAM memory at least (quick access type such as DDRAM is prefered). As for computational load, the following measurements might be considered:

**Table 2.5. Simultaneous requests according to vocabulary size**

| Machine features | Simultaneous requests |
|---|---|
| 1 GHz Intel Pentium III | 17 |
| 2,4 GHz Intel Pentium IV | 27 |

Like in speech recognition, these result have also been obtained by forcing an immediate answer time between request and the begining of the synthesis. In environments where answer time is not critical, the number of

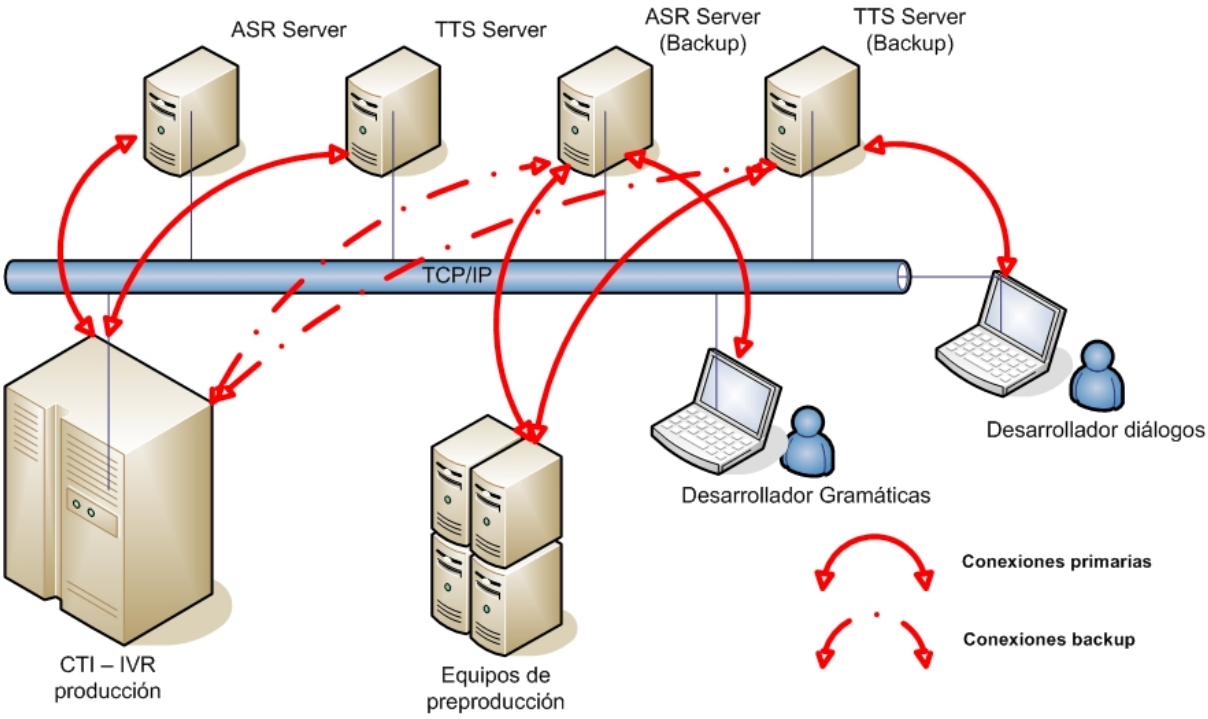
synthesis simultaneous requests might be slightly higher than those shown in the table, without causing any deterioration in the service quality.

## 4.3. Strategies for architecture design

Speech systems developed on *Verbio* follow a strategy based on client-server communication. This means that in the same work environment, various servers (all of them in different machines) and various clients (they can be in the same machine) can coexist. This panorama and the possibility that each client has (concretely, each of its separate lines) to connect to a different server, enables distribuiting computational load between all servers present in the system. Thanks to all this you obtain:

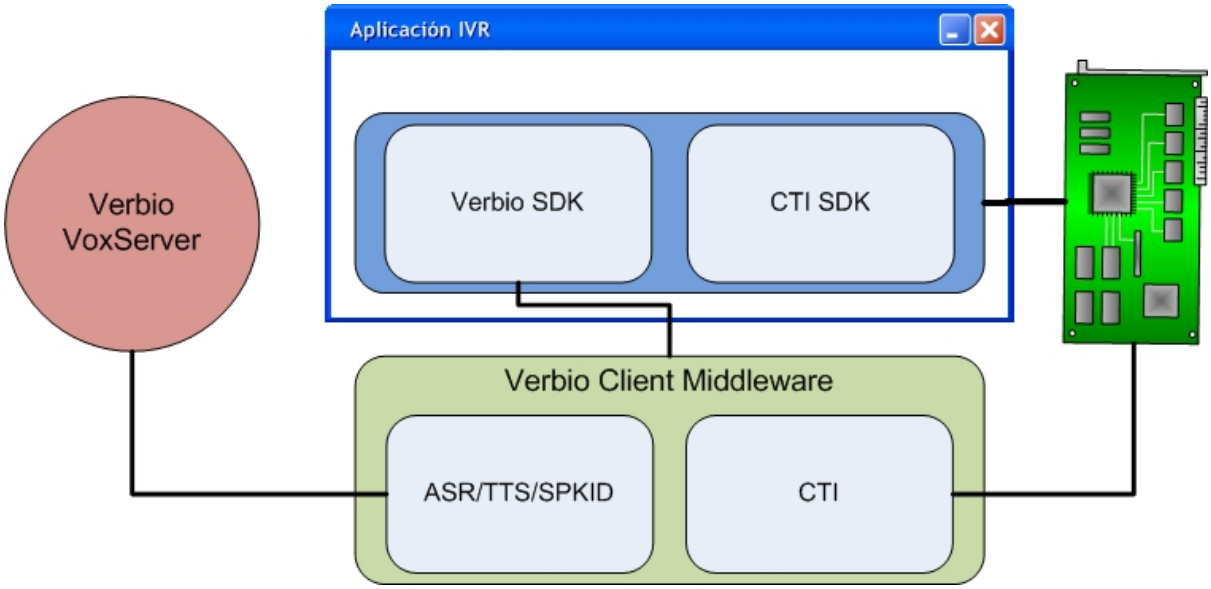
- Reuse and better use of the available resources: it is not necessary to get rid off old equipment if the size of the vocabularies increases or if the synthesizer is upgraded to another one that consumes more memory or if the number of lines dealt with increases. In any of these cases, it is enough to add another server to the environment so that by redistributing the load between all servers it is not necessary to replace them all for machines that offer better features.
- `Backup` strategies implementation: it is not only possible that each line (of each client) is connected to a different server, but it is also possible to indicate backup servers so that, in case any server drops out, the lines using it can connect automatically to some other server available. Backup servers can be equipments used exclusively for that function or they can be other system servers already in use (in this case, if any of them drops out it could imply, according to dimensioning, a certain overload on the remaining servers, which might cause small delays usually insignificant).

The following design shows an example of client-server architecture in which each client (a generic one developed by the integrator and another one based on `Verbio VoiceXML` interpret) are connected to their respective synthesis and recognition servers. In turn, these clients are configured in such way that, in case of losing connection with their server, they divert their requests to the other client's server. This strategy makes it possible that both clients remain operative in case one of the servers drops out. Obviously, in the event that the operative server was already very loaded, the overload derived from the additional client requests might lead to certain delays (during peak-time moments) in the systems, but this is still preferable to losing service in some of the clients.




Client-server architecture with redundancy at servers level.

In systems that due to simplicity or costs there is no need ofredundancy or more than one server, *Verbio* enables installing the client (or clients) and the server in the same machine, as shown in the following design. In addition, this design also shows the typical internal modules that are usually present in client applications.



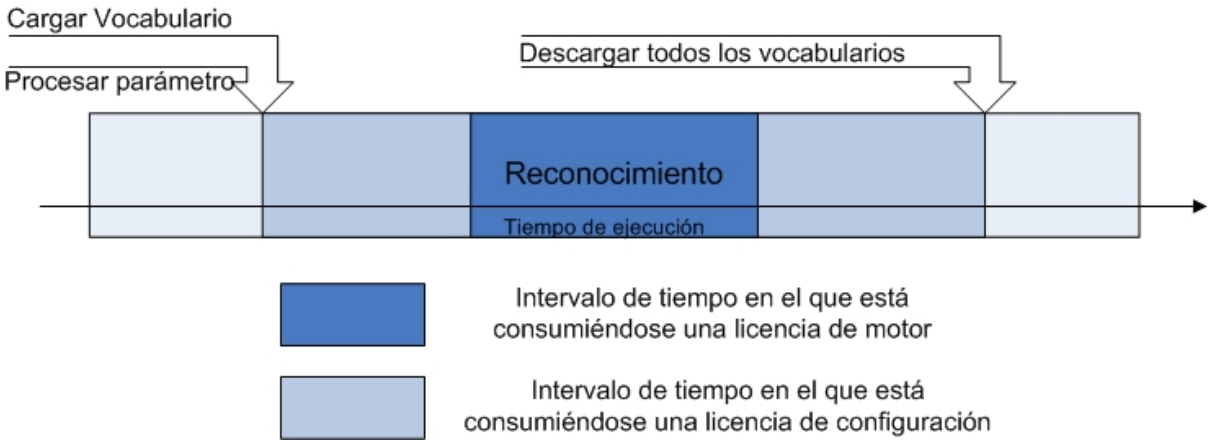

Client-server architecture in one-server environment.

# 5. license policy

*Verbio* has differentiated licenses for speech recognition and speech synthesis and, within these two groups, different licenses for each of the supported languages. That way, it is only necessary to purchase required licenses, making it possible to update or increase the number of licenses as system's needs increase.

## 5.1. Recognition licenses

Speech recognition licenses are divided in two main groups: `engine`licenses and `configuration`licenses. Engine licenses are consumed only during the recognition process; that is, while the server is processing voice samples to obtain the recognition result. Configuration licenses are associated to each of the configurations available, so that to be able to use a configuration it is essential to possess its appropriate license. Configuration licenses are consumed while there is some loaded vocabulary linked to such configuration or from the moment of consulting/establishing any of the parameters that rule recognition engine performance (consult technical documentations to know which parameters affect license consumption). Configuration licenses are released when all vocabularies are downloaded. During recognition, the system will be consuming one engine license and one configuration license at least (the one associated with active vocabulary(ies)).




Time frames for the different types of licences (engine and configuaration)

Because the time frame in which configuration licenses are consumed is longer than that in which engine licenses are consumed, it is likely that the number of configuration licenses is higher than the number of engine ones, if application configuration has not taken into account this aspect.

In the event of recognition in multiple languages (not contained in the same configuration), it will be necessary to have an engine license and a license for each of the configurations that take part in recognition.

## 5.2. Synthesis licenses

Speech synthesis licenses, alike recognition ones, are divided in two main groups: `engine`licenses and `language`licenses. Both licenses are consumed during the synthesis process; that is, while the speech synthesis process generates and sends client samples. To carry out the synthesis process, it is necessary to have one engine license and one license for each of the languages that appear in the text to be synthesized. In case that any of them is missing, none of the text fragments will be synthesized.

## 5.3. License supports

Licenses can be purchased in `software` (file) or `hardware` format: parallel port or USB sentinel. The advantage of the file format is delivery immediacy (they are sent by `e-mail`) and lower cost (you do not need to pay `hardware` supplements, duties, etc.). The main inconvenient is that they depend on a series number inherent to the machine. This makes the license unoperative in other equipments (it is not portable) or even in the same equipment after operative system reinstallations , `hardware` changes, etc. In these cases it is indispensable to purchase a new license, as there is no way to garantee non-fraudulent migration. For that reason, sentinel-based licenses give a solution to the problem because they are independent on the series number. Therefore, they can be used in any machine. High costs and lack of immediate availability can be solved by purchasing the sentinels in the target domain and by programming them through `Verbio Server Configuration Manager` with the aid of a programming file that can be sent by `e-mail` (http://www.rainbow.com/products/sentinel/superpro.asp) in its parallel port and USB version.

## 5.4. License upgrade

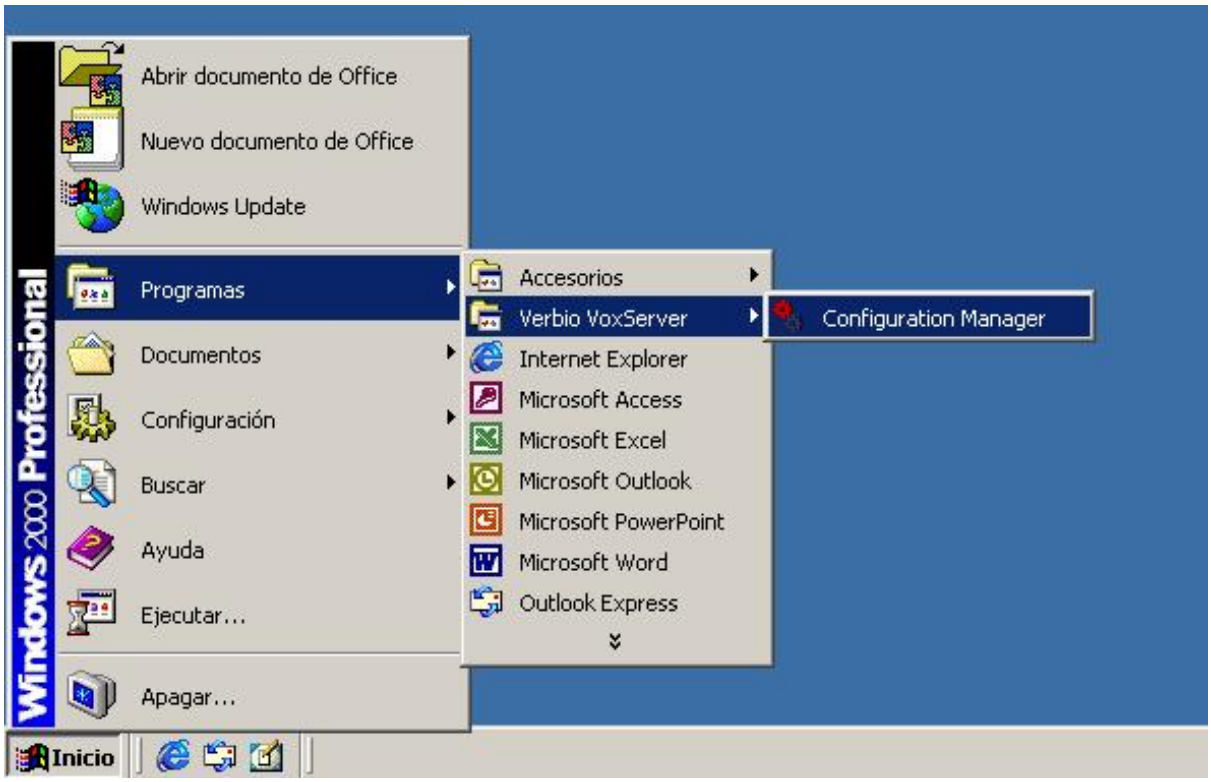
Unlike previous *Verbio 7* versions (`IberVox 6.41` and previous ones), from version 7 on there are two types of licenses, as it has been described in the sections above. This needs to be taken into account when requesting `upgrade`for previous versions, provided that to guarantee full compatibility you will need to purchase the necessary licenses of each type. Broadly speaking, these are the keys to upgrade the version properly:

- You need to purchase as many engine licenses (`ASR` and/or `TTS`) as licenses purchased for previous versions.
- You need to purchase as many `TTS` language licenses as licenses purchased for previous versions (they are consumed at the same time as engine licenses).
- The amount of `ASR` configuration licenses you need to purchase will depend on the programming strategy followed during application development. In case of loading the vocabularies just before launching the recognizer and downloading them all at the end of the recognition process, the number of configuration licenses would be equal to engine ones. The more time there is between loading and downloading (or parameter manipulation)more time will the configuration license be busy, so it will be more likely that the system finds all licenses busy. In the extreme case of loading the vocabularies at the begining of the aplication and downloading them at the end, you will need as many configuration licenses as open chanels.

# 6. Verbio server startup

## 6.1. Verbio Server Configuration Manager startup

Once server mandatory modules and necessary optional modules have been installed, you can proceed to *Verbio* speech recognition and/or synthesis server startup. To do so, you need to use `Verbio Server Configuration Manager`.




Starting up *Verbio* configurator, `Verbio Server Configuration Manager`

From `Verbio Server Configuration Manager` you can carry out the following operations:

- Starting up and shuting down *Verbio* recognition and/or synthesis server.
- Specifying the recognition `configurations` that will be used, by indicating the default language for each of them as well as the default `configuration`. The default `configuration` will be used for preparing the vocabularies and grammars unless a different `configuration` is indicated. See Section 6.2, “Specifying configurations of interest”.
- Specifying the *speakers* that will be used, as well as the default *speaker* for each language. The default *speaker* will be used in the synthesis process unless another speaker and/or language has been previously specified . See Section 6.3, “Specifying speakers of interest”.
- Specifying whether the *speakers* will be loaded in memory or whether they will have direct access to disk. In Section 4.3, “Strategies for architecture design” we give detail of the aspects concerning this choice. See Section 6.3, “Specifying speakers of interest”.
- Increasing the number of available licenses. See Section 6.4, “Upgrading licenses”.
- Determining whether *Verbio* recognition and/or synthesis service will boot automatically when starting up the equipment and also determine the work frequence of the synthesis engine. See Section 6.5, “Launching the synthesis and/or recogntion server”.

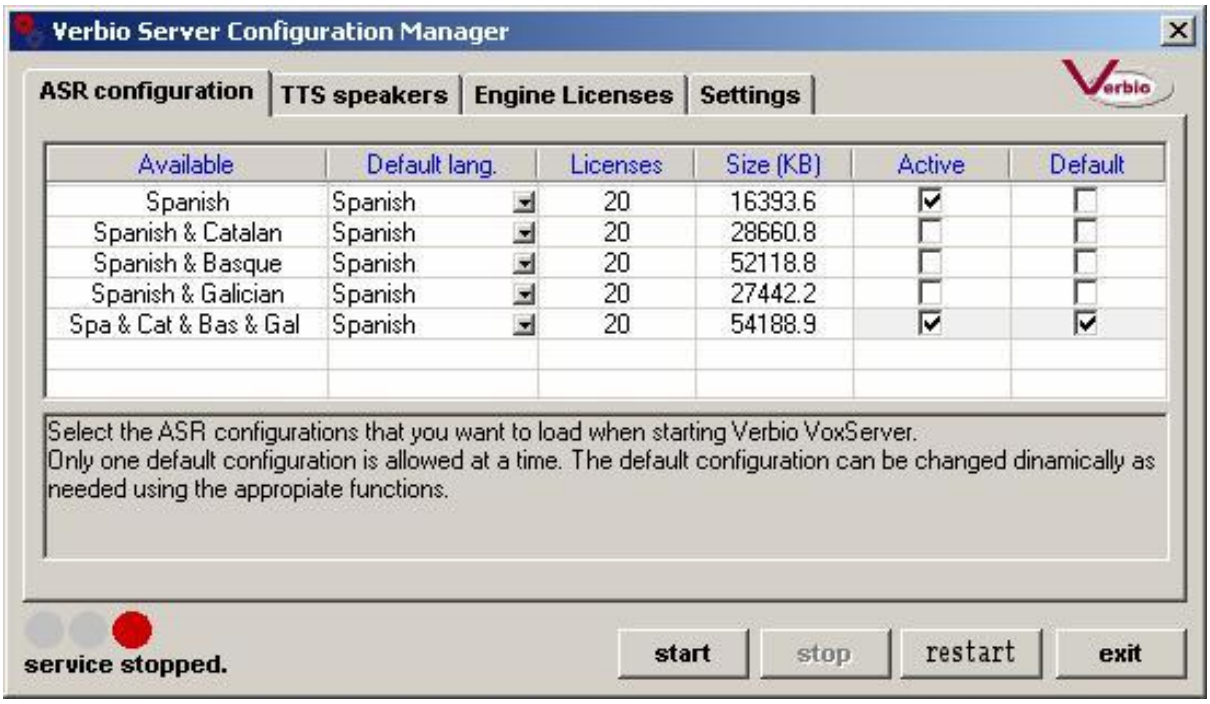
Likewise, `Verbio Server Configuration Manager` enables diplaying the following informations:

- `Configurations` installed in the equipment, as well as the number of licenses available for each of them and the size in disk space they take up.
- The available `speakers` in the system, as well as their characteristics, the available licenses for each of them and the space they take up when active.

- The available licenses for recognition, synthesis and SAPI use.
- The server series number, as well as the port by which it operates. It also shows the key series number (`sentinel` or bitlock) that contains the licenses when it is insterted in the slot. Otherwise, the value is *-1*.

Once `Verbio Server Configuration Manager` is booted, the first step is to verify that there are licenses for all those `configurations` (`ASR configuration` tab) and/or `speakers` (`TTS Speakers` tab) that you wish to use. If not, you need to consult the server or key series number (`Settings` tab) and contact your usual provider who will provide the appropriate `upgrade` file. If you do not have any licenses, you can test the system in assessment mode for 60 minutes periods. For more information on how the licensing system works consult Section 6.4, “Upgrading licenses” or contact your usual provider.

## 6.2. Specifying configurations of interest




Specifying and configuring recognition `configurations` of interest.

In case of wishing to use speech recognition, and after having verified that you have the necessary engine and `configuration`licenses (see previous chapter and Section 6.4, “Upgrading licenses”), you need to tell the server which `configurations` need to be booted, in what language it will be done ( in case of multilingual `configuration`) and which one needs to be the default `configuration`. The default `configuration` is the one that will be used for preparing the vocabularies and grammars unless a `different configuration` is indicated through any of the `SDK` functions.

You need to select the box in `Active` column for all `configurations` that need to be available in the server once it is operative. In Section 3.1.1, “Recognition configruations” you can obtain a list of the necessary `configurations` according to the requisits (clients) of the applications that communicate with the server.

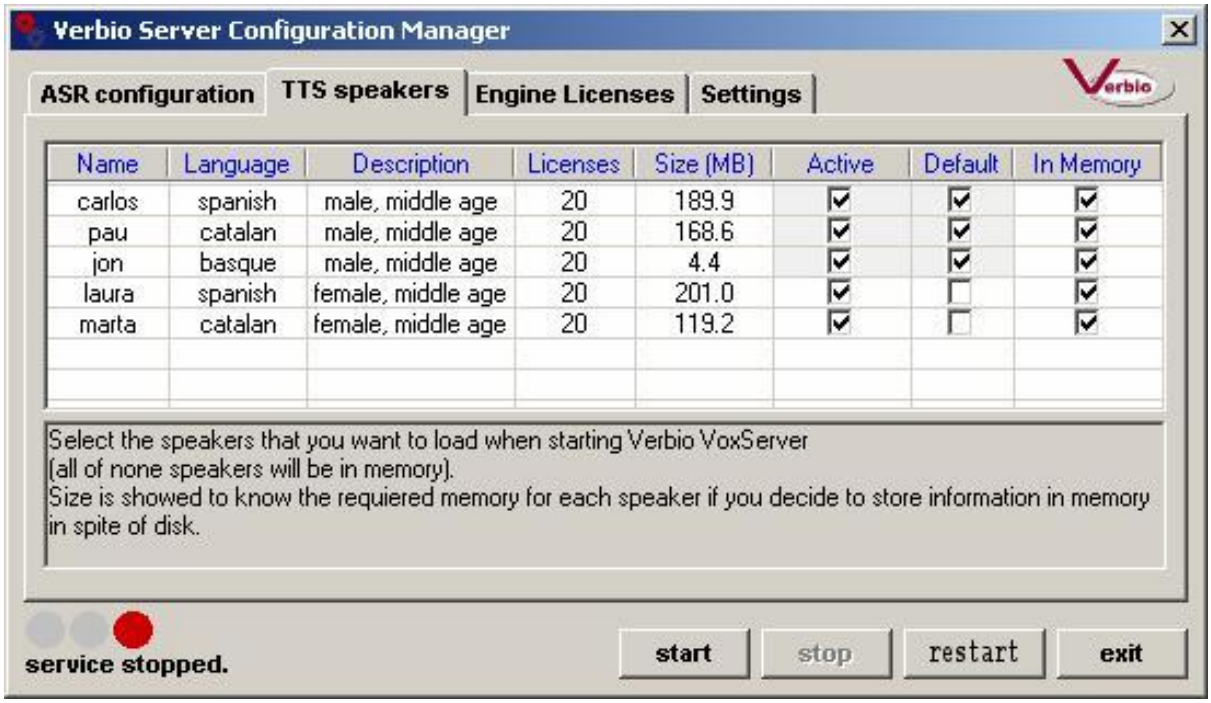
Only one default `configuration` needs to be selected of all active `configurations` (by default, the first activated `configuration` is selected) by selecting the appropriate box in`Default` column.

Last step consists in selecting the default language for all multilingual active `configurations`. The default language is the one that will be used for preparing the vocabularies and grammars when its associated `configuration` is used, unless it has been stated otherwise (by code).

### Caution

It is essential to stop and start the service up again to make configuration changes effective.

## 6.3. Specifying speakers of interest




Specifying and configuring synthesis `speakers` of interest.

In case of wishing to use speech synthesis and after having verified that you have the engine and `speaker` licenses desired (see previous section and Section 6.4, “Upgrading licenses”), it is necessary to tell the server which `speakers` need to be loaded, as well as the default `speakers` for each language.

You need to select the box in `Active` column of all speakers that need to be available in the server once it is operative.

For every work language you need to select a default `speaker` that will be used in requests in which no other `speaker` has been indicated for the affected language. By default, the `Default` box of the first activated`speaker` for each language is selected, although this selection can be modified afterwards.

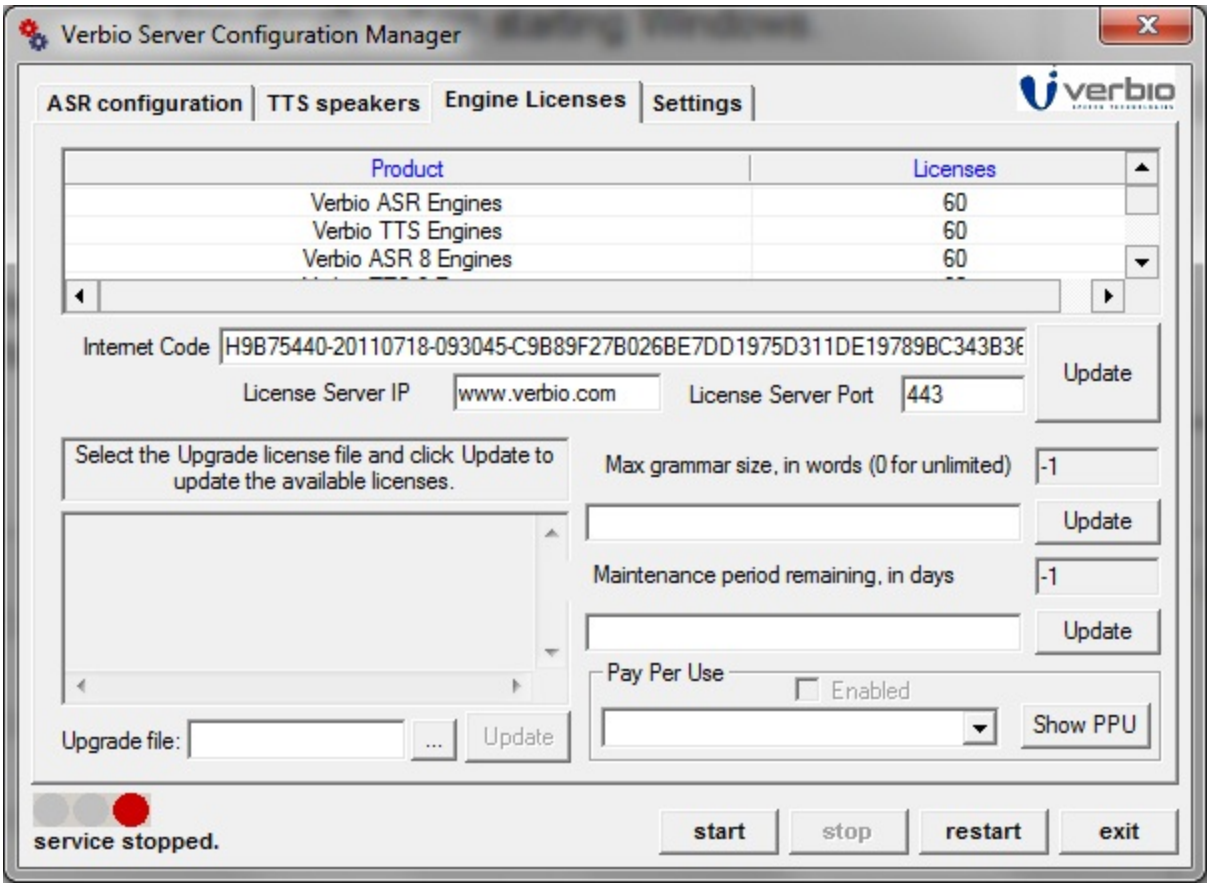
Finally, you have the choice to access disk or memory (this measure affects the whole set of selected`speakers`) during the synthesis processes. In Section 4.2, “Speakers being used” we give detail of all aspects to be taken into account when deciding between one option or the other. If you wish to access memory (higher speed but greater memory consumption) during the synthesis process you need to select `In Memory` box of any of the activated `speakers`. Otherwise, you need to unselect these boxes.

It is also possible to indicate the synthesis server work frequence. Most of *Verbio* speakers allow working at 8 KHz (for phone applications) and at 16 KHz (for `desktop`applications). In the event that all of them are able to work at both frequences, you will be able to select the frequence you wish through the drop-down box placed in `Settings` . It is important to take into account that it is not possible to select work frequence individually for each speaker.

### Caution

It is essential to stop and start the service up again to make configuartion changes effective.

## 6.4. Upgrading licenses




Consulting and upgrading available licenses.

Using *Verbio* speech synthesis and recognition systems requires, according to each case, acquiring various types of licenses, so they adapt better to different uses.

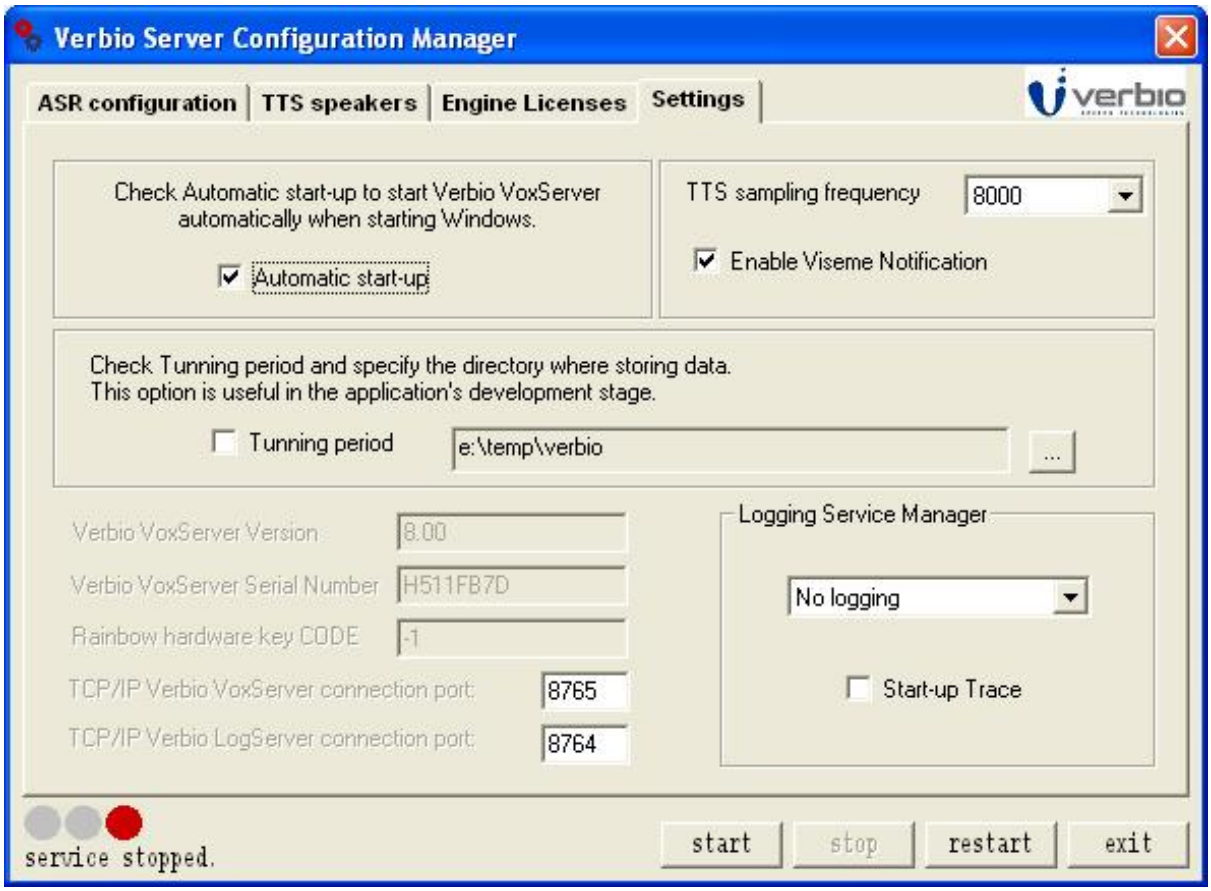
- *ASR engine licenses*: It is necessary to have as many recognition engine licenses as simultaneous recognition requests you wish the server to deal with, regardless of the `configurations` being used. The amount of recognition licenses available is shown in the`Engine Licenses` tab.
- *TTS engine licenses*: It is necessary to have as many synthesis engine licenses as simultaneous synthesis requests you wish the server to deal with, regardless of the languages being used. The amount of synthesis licenses available is shown in the `Engine Licenses` tab.
- *Access licenses to an external SAPI* : It is necessary to have as many external SAPI engine licenses as simultaneous requests you wish to carry out from *Verbio* to a compatible SAPI `speaker` from another speech technology provider. The amount of external SAPI licenses available is shown in the `Engine Licenses` tab.
- *Acces licenses via SAPI* : It is necesseray to have as many SAPI access engine license as simultaneous requests you wish to carry out from a compatible `SAPI` application to *Verbio* `speakers` using this protocol. The amount of SAPI acces licenses available is shown in the `Engine Licenses` tab.
- *Specific licenses for each configuration (recognition)*: According to the languages that you wish to use in recognition it is necessary to have licenses (as many as simultaneous requests to process) for the configuration(s) that comprise them. Obviously, the number of licenses for a configuration should not be higher than the number of ASR engine licenses available (they woud be underused). The amount of licenses available for each configuration is shown in the `ASR configuration` tab.
- *Specific licenses for each language (synthesis)*: According to the languages that you wish to use in synthesis it is necessary to have enough licenses (as many as simultaneous requests to process) of such languages (a language license enables using any of the speakers of that language). Obviously, the number of licenses for one language should not be higher than the number of TTS engine licenses available (they would be underused). The amount of licenses available for each speaker is shown in the `TTS Speakers` tab.

Licenses can be purchased/increased by applying for an `Upgrade` file to your usual provider. The following information should be provided:

- *Type of support desired*: *Verbio* products provide two types of supports to host licenses: text file and `hardware` key. The advantage of first type is immediacy in delivery (by email) and low cost as key needs not paying. The main inconvenient is that this license is associated to the product series number (or the phone card series number), so it stops being valid when the phone card is replaced or when the equipment is reformatted (product series number is obtained from the operative system information). The key has the advantage of being reusable in any environment or equipment in which the device is placed, regardless of the product series number or the phone card being used. In addition, in case of breakdown it is easily replaced for another key (after returnin the faulty one), which cannot be done with the file support. The main inconvenients are that delivery time depends on the transportation and cost is slightly higher because it includes key and port costs. In the event of choosing the keys, you will need to indicate whether you want a parallel port key or a USB port one.

- *Licenses desired*: You need to choose the type and amount of licenses (engine licenses, `configuration` ones, etc.). In this section there are the details on the various types of existing licenses. If you have any doubts on that, consult your usual provider.

- *Series number*: In case of file support, you need to give the product or CTI card series number. In case of key support, you need to give the key series number (if you want to reuse or reprogram an existing key). Both series numbers can be consulted in the `Engine Licenses` tab. If you cannot find a key in the equipment's port (USB or parallel), the box content is *-1*




Product and key series numbers required for license obtention.

In the event of using licenses in key support and wishing to increase the number of licenses, it is necessary to place the key in the appropriate slot and apply for an `Upgrade` file to your usual provider. This file needs to be selected in the box called `License file` in the `Engine Licenses` tab by using the attached file explorer. Once the file is selected, on the left window you can check its content (which needs to match the applied licenses). After verifying the file content, press the **Update**button and licenses will be automatically upgraded in

the key.

In the event of using licenses in file support and wishing to increase the number of licenses, it is enough to apply for a new file to your usual provider that will replace the initial one (the file provided will include the total of previous and new licenses).

### Caution

It is indispensable to stop and start the service up again for the server to detect the new licenses.

## 6.5. Launching the synthesis and/or recogntion server

Once `configurations` and/or `speakers` of interest have been selected, you need to start the recognition and/or synthesis server by pressing **Start**. If you have licenses for all selected `configurations` and/or `speakers`, *Verbio* will be installed as a `Windows` service, starting up jointly with the operative system if option `Automatic start-up` in `Settings` has been selected. Otherwise, it will have to be manually started through `Verbio Server Configuration Manager` or through `Windows service` control panel.

If some of the selected `configuration` and/or `speaker` licenses are missing, *Verbio* will start up in assessment mode and will finalize after the time frame shown on screen. Starting up light will remain red indicating that there is some license missing and that, therefore, service mode has not been started. However, it is possible to use that system once the sign`Started` appears on the command screen.

## 6.6. Stopping and removing the service

You can stop *Verbio* speech recognition and/or synthesis from `Verbio Server Configuration Manager` or from `Windows service` control panel. In both cases you need to press **Stop**

Service removal can only be done by using the appropriate remover from `Add and remove programs` in `Windows Control panel`. It is important to remember that removing the various modules must be done in reverse order of installation: in the first place, the ones associated to `configurations` and `speakers` and, afterwards, the one associated to engines.

### Caution

Before proceeding to upgrade, it is essential to stop the service and remove all previous version's components by following the recomendations in the paragraph above.

# Chapter 3. Platforms, tools and development kits

## 1. Software and hardware platforms

### 1.1. ComputerTelephony Integration (CTI)

Wide penetration of telephony within society (stronger since cellular telephony appeared) forces a great number of organizations to manage significant volumes of calls (costumer service, marketing campaigns, etc.). The possibility to automate most of this calls led to emergence of specific `hardware` : phone cards, which incorporate interfaces to interact with any telephone protocol, thus providing a wide range of functionalities intended for processing any sort of call. This new technology that exploits interaction between computer and telephony fields is known as *Computer Telephony Integration (CTI).*

Parallely, constant evolution of speech recognition (`ASR`) and speech synthesis (`TTS`) systems has enabled its incoroporation into automatic call reception and transmission systems. These systems, which due to their oral nature should have been the natural habitat for speech tecnhologies, resorted to strategies such as prerecorded messages reproduction and recognition through `DTMF`tone dialling in order to interact with users. Even though these strategies are still valid and even advisable in some cases, currently, features obtained by means of using `ASR` and/or `TTS`systems jointly or seprately enable providing automated systems with greater naturality and fluidity in the process of exchanging information with the user.

Emergence of a wide variety of phone cards that adapt to every need and development of current computers as for calculation power, have made `CTI` technology available also to small and medium-sized enterprises because it does not require big investments. Moreover, this technology is easily scalable; that means that its implementation can be done progressivley according to the needs of each moment.

`Verbio` development has taken into account that most part of the current applications in speech recognition and synthesis systems have been developed to work in telephone environments. For that reason, recognition systems in `Verbio` have been trained with audio signals from telephone environments, landline and cellular ones, with the purpose to obtain the best recognition rates possible in this type of environment.

`Verbio` has been used succesfully by integrators worldwide, workin on cards from the following manufacturers (in alphabetic order) :

#### 1.1.1. Dialogic (Intel)

You can find information on cards manufactured by `Intel Dialogic` in http://www.dialogic.com. It is possible to incorporate `Verbio` into `Dialogic` environments by using the specific *Dialogic SDK* or *CT-ADE SDK* programming environments, or by using generic *Library SDK* and *Advanced SDK* environments. In Section 2, “SDK - Software Development Kit” you will find more information on available programming environments so you can choose the best option for every situation.

#### 1.1.2. Eicon

You can find information on cards manufactured by `Eicon Networks` in http://www.eicon.com. It is possible to incorporate `Verbio` into `Eicon Diva Server` environments by using *Library SDK* or *Advanced SDK* programming environments. In Section 2, “SDK - Software Development Kit” you will find more information on available programming environments so you can select the most suitable option for every situation.

#### 1.1.3. NMS

You can find information on cards manufactured by `Natural Microsystems` in http://www.nmss.com. It is possible to incorporate `Verbio` into `NMS` environments by using *Library SDK* or *Advanced SDK* programming environments. In Section 2, “SDK - Software Development Kit” you can find more information on programming environments available so you can select the most suitable option for every situation.

#### 1.1.4. Avaya IR (Conversant)

You can find information on systems commercialized by Avaya in http://www.avaya.com. It is possible to incorporate Verbio into Avaya IR (Conversant) environments by using the following programming environment: *Avaya IR SDK*. In Section 2, “SDK - Software Development Kit” you will find more information on the various available programming environments .

### 1.1.5. Teima

You can find information on cards manufactured by Teima in http://www.teima.com. It is possible to incorporate Verbio into Teima Audiotextenvironments by using some of the following programming environments: *Library SDK* or *Advanced SDK*. In Section 2, “SDK - Software Development Kit” you will find more information on programming environments available so you can select the most suitable option for every situation.

### 1.1.6. Other platforms

Verbiocan be used together with CTI cards or other manufacturers through *Library SDK* or *Advanced SDK* programming environments. In both cases, the only requisite is that the developer knows enough API card programming as to be able to implement the part on acquisiton and sending of samples from/to the CTI card (sample exchange with voice recognition and synthesis system respectively).

In Section 2, “SDK - Software Development Kit” you can find more information on available programming environments so you can select the most suitable option for every case.

## 1.2. programming desktop applications

In the previous section it was highlighted that, at present, the main market for recognition and synthesis systems is telephone applications market. However, these systems can be incorporated in any environment that has audio sample acquisition and reproduction devices available. Typically, this device can be a personal computer with a sound card. Therefore, it is also possible to develop the desktop applications that interact with the user through a microphone and loudspeakers.

At present, Verbio does not offer comprehensive recognition models specifically adapted to non-telephone environments (desktop, domotics), although it does offer technology for generating specific recognition models for specific applications, as it has been done in many of the existing projects. This way, without changes in the applications, clients have models adapted to their needs at all times, thus obtaining high succes rates. As for the text-to-speech converter, it does have a version adapted to desktop environment characteristics, thus achieving better synthesis quality in comparison with telephone environments thanks to greater broadband availability.

Developing desktop applications needs to be done by means of *Library SDK* or *Advanced SDK* programming environments. In Section 2, “SDK - Software Development Kit” you can find more information on available programming environments so you can select the most suitable option for every case.

# 2. SDK - Software Development Kit

Verbioincludes a set of application development libraries (Software Development Kit - SDK) in order to facilitate including speech recognition and synthesis resources in multiple environments. In the section below you will find the description of each of the existing SDKwith the purpose that every developer finds the one that better suits the environments and hardware being used.

## 2.1. Library SDK - Generic platform

*Library SDK* provides a set of low level functions that enable using any audio device (CTI cards or sound cards), as long as the developer knows how to exchange audio samples with such device. That means that the developer will be responsible for obtaining audio samples and pass them to the recognition system (in speech recognition processes) or else reproducing the audio samples delivered by the synthesis system (in speech synthesis processes).

The functions in this SDK start with the prefix *vox_* and are available from C/C++ programming environments, although it is possible to invoke them directly to the DLL from different environments, such as Visual Basic. *Library SDK* is kept for its compatibility with old versions (although new functionalities have been incorporated to it). However, in case of developing an application from scratch, it is advisable to use the other *Advanced SDK* generic platform.

## 2.2. Advanced SDK - C++ generic platform

*Advanced SDK* provides a set of functions at a higher level than the functions contained in *Library SDK*, although this does not mean losing control level in functionalities of interest. This way, we intend to facilitate the use of speech recognition and synthesis, specially in those aspects that imply greater complexity, such as managing the results returned by the recognizer or managing client-server disconnections. There is larger duality between the functions of both SDK, so it would be advisable, when possible, to migrate from *Library SDK* to *Advanced SDK* to make the most of new (and future) functionalities.

*Advanced SDK* has been developed from *Library SDK*, incorporating elements from *Dialogic SDK* with the purpose to facilitate the use of Dialogic cards and also adding similar functions to facilitate the use of cards from other manufacturers, such as Natural Microsystems or Eicon.

*Advanced SDK* is made of a set of objects defined by C++ classes, which have been initially developed to be used from Microsoft Visual Studio.

## 2.3. Dialogic SDK - Dialogic

*Dialogic SDK* provides a set of low level functions specially designed to facilitate the use of Dialogic cards, since internally they already interact with the CTI card to obtain or send the audio samples in recognition and synthesis respectively.

The functions in this SDK start with the prefix *vx_* and are available from C (or C++) programming environments, although it is possible to invoke them directly to the DLL from different environments, such as Visual Basic. *Dialogic SDK* is kept for its compatibility with old versions (although new functionalities have been incorporated to it). However, in case of developing an application from scratch, it is advisable to use the other *Advanced SDK*generic platform, that also possesses features to interact with Dialogiccards.

## 2.4. CT-ADE SDK

*CT-ADE SDK* provides a set of functions to be used from CT ADEdevelpment environments (old Parity Software VOS).

## 2.5. Avaya IR SDK

*Avaya IR SDK* provides a set of functions to be used in Avaya IR (Conversant) development environments.

## 2.6. Microsoft SAPI 4/5 - TTS

Microsoft developed the standard *Speech API (SAPI)* with the purpose of establishing use patterns for recognition and synthesis engines developed by any manufacturer. Like so, any application can easily use any compatible SAPI 4/5 recognition and/or synthesis engine. All speakers included in Verbio meet SAPI 4/5 specifications, so they can be used by using Microsoft SDK. For further detail, you can consult http://www.microsoft.com/speech.

# 3. Developer's tools

## 3.1. Verbio Grammar Manager

Verbio Grammar Manager: Tool intended for designing and testing recognition grammars used in the application. For more information consult the associated documentation Verbio Software Reference: Grammar Manager User's Guide.

## 3.2. Verbio Read Aloud

Verbio Read Aloud: Tool intended for testing the available speakers by means of specifying any text and manipulating its basic parameters. For more information consult the associated documentation Verbio Software Reference: Read Aloud User's Guide.

# Chapter 4. Speech recognition

## 1. Introduction

Speech recognition allows an application to function as a transformer of a user voice input into written text. *Verbio* recognizer is independent from the speaker, which means that there is no need of training or adaptation of the system to the user for the recognizer to work. Verbio has been trained with a population of thousands of speakers of each language in a balanced proportion as for sex and age and from all areas of the regions of each language by recording their voices through landland and GSM networks.

This chapter explains how to work with *Verbio* speech recognizer and gives detail of the implied concepts and steps, which are summarised below.

Speech recognition is able transform the voice from a user into text if it knows all possible word combinations that can be said by the user at a particular moment. The description of the word vocabulary to be used by the recognizer as well as the set of its possible sequences in any valid sentence is what is known as recognition *grammar* (see Section 3, “Recognition grammars”).

The more the grammar is specialized, the better the results achieved. In contrast, it often decreases flexibility and naturality of the users speech style . For example, a grammar `A` for the recognition of dates is able consider dates in standard format “21st of May”. Another grammar `B` is also able consider relative dates such as “friday” or “next month, the fourth”. Grammar `B` will obviously be more flexible, but it could make more mistakes when trying to recognize dates in standard format. Even a third grammar `C`could consider times and destinations for flight booking as well as dates. In this case, flexibility will be much greater, but performance of the recognition could be diminshed in contrast with the more controlled situation offered by grammars `A` and `B`.The most commonly used grammars (dates, numbers, answers type yes/no,etc.), are already internaly defined in the system so that best results can be achieved at all times.

Once a grammar is defined, there are various time parameters to be considered: maximum duration of initial silence, silence permitted once the user has started to speak and the maximum total time (see Section 5, “Recognizing an utterance”).

Processing the result does not merely consist in obtaining text resulting from the input voice. It is important that the recognizer knows a reliability measure (which makes it possible to reject results or, at least, to ask for confirmation to the user), by assessing alternative hypothesis (`N-best`) or by processing the result through a semantic interpretation (see Section 6, “Obtaining the result”).

## 2. Initializing

Initializing the engine of speech recognition consists in enabling that one or more languages are used in the application as well as specifying the default language for the recognition vocabulary (see initializing functions in `Function Reference` for further detail or Section 6.2, “Specifying configurations of interest” to specify the default values).

## 3. Recognition grammars

A recognition grammar defines the wide range of possible answers that the system can recognize at a particular moment. This is the “language” subset accepted by the recognizer. Therefore, there has to be an appropiate grammar for every single moment in the dialogue between the person and the machine. A grammar specifies the word vocabulary that the system is able to recognize as well as the possible sequences of word combination.

### 3.1. Types of grammars

For example, given a word vocabulary“shut, open, the, window, door”, a grammar could be “one of the words of the vocabulary” (that is, recognition of isolated words allowing the user to say phrases like “open” or “window” for instance); another grammar could be “one or various words of the vocabulary” (that is, recognition of connected word allowing the user to say phrases like “door” or “open the door”, but also“the shut open”); finally, another grammar“based on patterns”like“ `[open | shut] [the ] [door | window]` ”setting natural language in a context-free grammar.

The different types of grammar define the language and terminology to express how to combine the words.

### 3.1.1. Word lists

The simplest grammar for speech recognition is the one mentioned above as “one of the words of the vocabulary”. *Verbio*allows the use of consistent text files in entry lists (each with one or more words), one per line. Every single entry will be denoted as *word of the vocabulary*. Example 4.1 shows an example of vocabulary including names of people composed of four entries or words of the vocabulary.

#### Example 4.1. Simple vocabulary of word lists

```
Luís Fernández
José Pérez
Pepe Pérez
María Sancho
```

Optionally, word lists can be arranged in columns divided by a tabular character(“\t”). Then, the second column of an entry is the word in its literal form, that is, the words to be pronounced (phonetic transcription). In this case, the first column is the result returned when the word is recognized. Example 4.2 is the same vocabulary including names of people but the result returned by the recognizer is the phone extension (note that the two entries can return the same word; “ `->` ” indicates tabulator (“\t”).

*Padding words* (entries preceded by #) are useful to indicate which accompanying words can be used in the grammar context but they are not really words with content (signifiers) (a padding word returned does not count as a returned word by the functions of result processing).

#### Example 4.2. Advanced vocabulary of word lists

```
1023       ->  Luís Fernández
1024       ->  José Pérez
1024       ->  Pepe Pérez
1032       ->  María Sancho
#ponme con
#por favor
```

“One of the words of the vocabulary” type of grammar is denoted in *Verbio* as *isolated words* (see `ISOLATED` flags in `Function Reference`). Similarly, “one or various words of the vocabulary” type of grammar, denoted in *Verbio* as *connected words* (see `CONNECTED`flags in `Function Reference`) use the same files but multiple entries for each user's utterance are allowed.

#### 3.1.1.1. Language

*Verbio* grammars include multilingual vocabularies: some words can be pronounced in one language and others in a different one. This is particularly interesting in name directories and, specially, in bilingual areas, where names ara usually pronounced according to their geographical origin.

Each language is specified with describers: “ `[$ESP]` ” (Spanish), “ `[$CAT]` ” (Catalan), “ `[$EUS]` ” (Basque), “ `[$GAL]` ” (Galician), “ `[$POR]` ” (Portuguese) and “ `[$DEF]` ” (default language). The default language is the one established when preparing the vocabulary.

In Example 4.3, the default language is Spanish. That means that any word pronounced in Catalan will be marked with the describer`[$CAT]`, stating the phonetic transcription rules to be applied. These describers indicate how to process the following words in the same vocabulary entry and not affect the processing of later entries (Note: blank spaces are left before and after language describers).

For further detail about its implementation, see information related to language in the initializing process and parameter establishment in `Function Reference.`

#### Example 4.3. Bilingual vocabulary of word lists (default languge: Spanish)

```
1023       ->  Luís Fernández              ; in Spanish
1024       ->  José Pérez                  ; in Spanish
1024       ->  Pepe Pérez                  ; in Spanish
1027       ->  [$CAT] Andreu Pujol         ; in Spanish
1027       ->  Andrés [$CAT] Pujol         ; "Andrés" in Spanish, "Pujol" in Catalan
1029       ->  [$CAT] Jordi [$DEF] Sevilla ; "Jordi" in Catalan, "Sevilla" in Spanish
1032       ->  María Sancho                ; in Spanish
```

```
#ponme con
#por favor
```

## 3.1.2. ABNF grammars

As it was mentioned before, recognition grammars can be seen as a structure based on patterns that describe the possible and valid sequences of words of the vocabulary. These grammars specify how *natural language* behaves in a particular task.

*Verbio* grammars follow the syntax described in the Speech Recognition Grammar Specification (SRGS) by W3C Speech Interface Framework (http://www.w3c.org/TR/speech-grammar) in its Augmented ABNF form (*ABNF*).

We recommend the use of `Verbio Grammar Manager` to edit, compile and test *Verbio* ABNF grammars.

### 3.1.2.1. ABNF grammars format

This section summarizes the SRGS (`Speech Recognition Grammar Specification`) in its ABNF form. Consult such specification for further detail.

A grammar defines the pattern of words to be recognized, a rule consistent of a regular expression that contains words, references to other rules or combinations of both.

#### 3.1.2.1.1. Tokens

A token is the part of the grammar that defines the word that can be pronounced, previously denoted as `word of the vocabulary`. In the syntax, a token is any combination of characters delimited by spaces or special symbols(`; = | () {} * + /* */ // <> ! "`). If a token contains spaces or special symbols it must be put in inverted commas (“”).

“ _ ” in between two words means that coarticulation between words should be carefully taken into account when transcribing. This option should not be used unless there is a particularly high coarticulation between words (for example, coarticulation of “y”in number 34, “treinta_y_cuatro”).

```
Hola                // token
ponme con Juan      // 3 different tokens
"buenos días"       // 1 token
treinta_y_dos       // 1 token
"[SIL]"             // 1 token: [SIL]
```

#### 3.1.2.1.2. Rules

Each rule is identified with a rule name. Reference to a local rule (defined in the grammar itself) is defined with its rule name, which always starts with the same character: “ `$` ”.

```
$day
$month
```

#### 3.1.2.1.3. Sequences

A sequence of tokens, rules or combinations of both are at the same time a rule. This determines the chronologic order in which the words can be pronounced.

```
ponme con Juan     // 3 token sequence
ponme con $name  // 2 token sequence and 1 reference to a rule
$padding $name   // sequence of references to rules
```

#### 3.1.2.1.4. Alternatives

A set of alternatives is also a rule. They are identified as a rule list separated by “ `|` ”.

```
Juan | José | "José Luis" | $names
```

Optionally, each alternative can have a weight (floating point value in between “ `/` ”).

```
/2/ Juan | /3/ José | /1/ "José Luis"
```

This weight is proportional to the instertion probability of such alternative in the recognition of an utterance. By default, the weight is 1.0.

A weight superior to 1 increases the instertion probability of a token in comparison with the rest of the words. A weight inferior to 1 implies a greater probability of such word being deleted.

If all words have a weight superior to 1, insertion probability of words is higher. Therefore, a list of alternatives with all weights at 1 is not equivalent to the same list with all weights at 10 for example. If that was the case, the insertion probability of words would be increased.

If the weight is assigned to a rule or expression, such weight will multiply the weights of all words in such rule. Therefore, the rule:

```
/2/ Juan | /3/ José | /3/ (/1/ Luis | /2/ Jorge)
```

is equivalent to:

```
/2/ Juan | /3/ José | /3/ Luis | /6/ Jorge
```

Therefore, it is advisable to be carefull with weights and it is preferable to use them exclusively in final words.

#### 3.1.2.1.5. Special rules

GARBAGE — It defines a rule that can map any speech segment until the following rule does so or otherwise at the end of the utterance.

SILENCE — It defines a rule that can map a silence. This is useful to indicate particularly long silences within an utterance. By default, silences at the beginning and at the end of the utterance are automatically inserted in the grammar processor. In between words, and also automatically, optional short silences are inserted.

```
// "ponme con Juan"
// "por favor podrías ponerme con José"
// "Luis"
$GARBAGE (Juan | José | Luis)
```

#### 3.1.2.1.6. Repetitions: optional, *, +

Optional words or rules are delimited by square brackets`[...]`. Symbols `*` and `+` indicate that the preceding word or rule can be repeated zero or more times or one or more times respectively.

```
// "1"
// "1 4 3 2"
$digit +

// "pizza"
// "pizza grande con queso"
// "pizza pequeña con queso y anchoas"
pizza [[muy] grande | pequeña] ([con | y] $complemento)*
```

#### 3.1.2.1.7. Tags

Each token can be associated to a text string that the recognizer returns in place of the pronounced word. A tag goes in between brace brackets `{...}`.

```
["ponme con" {}] ("Luís Fernández" {1023} | "José Pérez" {1024} | "Pepe Pérez" {1024})
```

When the tag is empty “ `{}` ”, the token is considered as a padding word and will never be returned by the recognizer. In the example above, the result given by the recognizer when dealing with the utterance `"ponme con José Pérez"` would be`"1024"`.

#### 3.1.2.1.8. Language

ABNF grammars include multilingual vocabularies, that is, words in the grammar that will be pronounced in a different language of in that set by default. There is always a default language used for the preparation of the vocabulary (for further detail about its implementation, see information related to language in the initializing process and parameter establishment in `Function Reference`).

The default language in a vocabulary or grammar can be set `inline` using the describer“ `language` ”followed by the language identifier ( “ `es` ” for Spanish, “ `ca` ” for Catalan, “ `eu` ” for Basque, “ `ga` ” for Galician, “ `pt` ” for Portuguese, “ `fr` ” for French and “ `en` ” for English) and a semicolon.

Token language can be set by adding an exclamation mark “ `!` ” after the token, followed by the language identifier.

```
language es;
$color =
         verde | azul | rojo |            // in Spanish
```

```
verd!ca | blau!ca | vermell!ca   // in Catalan
;
```

#### 3.1.2.1.9. Precedence

Operator precedence in rule definiton is:

1. Rule name denoted by symbol " `$` ".
2. Brackets " `()` "to group together, and square brakets " `[]` " to group optionals together.
3. Unary operators " `*` " and " `+` " and tags " `{}` " apply to an immediatly previous word or rule (brackets can group words and rules).
4. Sequences of words or/and rules.
5. Sequences of words or/and rules separated by " `|` ".

#### 3.1.2.1.10. Rule definition

A rule definition enables the association of a rule with a rule name. It consists of an optional field declaration followed by a rule name (which must start with " `$` "), an equal sign " `=` ", the rule defintion itself, and a semicolon " `;` " at the end.

```
$name = rule;
public $name = rule;
$color = rojo | verde | azul | "azul marino";
```

Field declaration can be "public" or "private", the latter being the default declaration. Rules defined as "public" are the initial rules for recognition.

#### 3.1.2.1.11. Main rule (root)

If one (and only one) of the rules is declared in the field "root", this will be the only rule used by the recognizer. If there is no rule declared as"root", the recognizer will use as the main rule the alternative of all rules declared as"public".

#### 3.1.2.1.12. Comments

Comments in `C/C++/Java` are accepted in grammar definition. Comments delimited by "/*" and "*/" or between "//" and the end of the line are ignored by the grammar processor.

Example 4.4 demonstrates syntax in ABNF grammars. Note that they all need to start with " `#ABNF 1.0 ISO8859-1;` ".

### Example 4.4. Examples of ABNF grammars

```
#ABNF 1.0 ISO8859-1;

$rgb  = rojo  {COLOR=R} |         // if "rojo" is recognized it returns COLOR=R
        verde {COLOR=G} |         // if "verde" is recognized it returns COLOR=G
        azul  {COLOR=B} |
        "azul marino" {COLOR=B};

// "el color azul"     -> COLOR=B
// "el rojo por favor" -> COLOR=R
// "azul marino"       -> COLOR=B
root $color = [color {} | "el color" {} | el {}] $rgb  "por favor" {}];


#ABNF 1.0 ISO8859-1;

public $day_week = [el {}] (lunes | martes | miércoles | jueves | viernes | sábado | domingo);
public $day_relative = ayer | hoy | mañana | "pasado mañana";

// implicit declaration of rule "root $day = $day_week | $day_rel;"
```

## 3.1.3. Built-in basic grammars

There are grammars specially designed for the most common tasks (and often difficult tasks) that have been

integrated in the recognizer as a built-in resource. Built-in basic grammars are not merely the definition of rules but also the internal processing of the results by considering reliability measures and optionally, multiple hipothesis. (`N-Best`).

| | |
|---|---|
| `boolean` | yes/no answer type, covering “yes”, “no” (with specific acoustic modeling), and the most common words and expressions for positive and negative answers (“exactly”, “right”, “no thanks”, etc.). |
| `digits` | length-limited or other digit strings, using specific acoustic models for connected digits (“one, three, four”, “seven”, etc.).. |
| `date` | dates in natural language, covering fixed and relative dates (“within three days”, “next monday”, etc.). |
| `number` | natural numbers within a limited rank (“two thousand three hundred and forty-seven”, “twelve”, etc.). |
| `phone` | phone numbers in natural language, covering the most frequent ways to group digits in natural numbers (“ninety- three, four zero nine, seventy-one, twenty”). |
| `currency` | currency amounts (“twenty-seven euros and five cents”) |
| `creditcard` | credit card numbers (spoken digit by digit) |
| `code` | numerical codes spoken as a sequence of natural numbers |
| `nif` | Spanish personal identification number (NIF, numero de identificación fiscal) |
| `spell` | spellings, alphabetical as well as numerical ones. |

Often, these grammars have parameters to set options such as the minimum and/or maximum length of a digit string, the range of number values, or date flexibility. These options make it possible to adjust the most suitable grammar based on the context or on the question being formulated so that best results are achieved.

Built-in basic grammars are denoted as follows:

```
builtin:grammar/name:language?param1=value1;param2=value2;...;paramN=valueN
```

For example:

```
builtin:grammar/number:es?max=2000
```

`Built-in` grammars may be used from the various SDK as if they were conventional ABNF grammars, the only difference being that*vocabulary does not need to be prepared*, given that it is already prepared. The grammar only needs to be set or activated so that it becomes operative. For example:

```
vx_loadvcb(chdev, "builtin:grammar/date:es?relative=true;year=false", GVX_ABNF); //Dialogic SDK
vox_setvcb(dev, "builtin:grammar/boolean:ca", GVX_ABNF); //Library SDK
EngineASRResource->LoadGrammar("builtin:grammar/nif:es", VERBIO_GRAMMAR_ABNF); //Advanced SDK
```

Likewise, `built-in`grammars may also be included in conventionalABNFgrammars. For example:

```
#ABNF 1.0 ISO8859-1;
root $item = $help | $spell;
$spell = $(builtin:grammar/spell:es?minlength=1;maxlength=1;alphanumeric=false);
$help = ayuda {HELP} | repetir {REPEAT};
```

The following sections give detail of the parameters for each of the built-in basic grammars as well as the semantic interpretation of the results.

#### 3.1.3.1. boolean

Yes/no answer type, covering “yes”, “no” (with specific acoustic modeling), and the most common words and expressions for positive and negative answers (“exactly”, “right”, “no thanks”, etc.).

This grammar does not have parameters.

As a result of the recognition, `boolean` grammar returns “true” (positive answer) or “false” (negative answer).

This grammar is available for Spanish, Catalan, Basque and Galician.

### 3.1.3.2. digits

Length-limited or other digit strings, using specific acoustic models for connected digits.

The parameters for this grammar are:

minlength
: `Values:` `int`

  `Default:` `1`

  `Definition:` The maximum length of digit strings.

maxlength
: `Values:` `int`

  `Default:` `10`

  `Definition:` The maximum length of digit strings.

length
: `Values:` `int`

  `Default:` `-`

  `Definition:` The exact length of digit strings (minlength = maxlength).

`Digits` grammar returns the recognized digit string (from “0” to “9”) without spaces. For example: “1972”.

This grammar is available for Spanish, Catalan, Basque and Galician.

### 3.1.3.3. date

Dates in natural language, covering fixed and/or relative dates (“within two days”, “next monday”) depending on the set parameters.

The parameters for this grammar are:

relative
: `Values:` `true`, `false`

  `Default:` `false`

  `Definition:` If the value is “true”, the grammar accepts relative dates like “tomorrow” and/or incomplete dates like“saturday”.

year
: `Values:` `true`, `false`

  `Default:` `false`

  `Definition:` The gramar accepts fixed dates that include the day and the month. If the value of this parameter is“true”, optionally the grammar also accepts the year.

voicexml
: `Values:` `true`, `false`

  `Default:` `false`

  `Definition:` If the parameter is “true”, the output format is `yyyymmdd`, using "?" in case some of the fields have not been filled in.

Setting “true” to both parameters increases flexibility on the way to say dates and does not worsen in any case recognition rates of utterances in day/month standard format. Thus differences are in terms of computational

costs. Grammars increase computational costs as their coverage broadens (and so their complexity), as detailed in Section 4.1, “Size of the vocabulary”.

`Date` grammar returns the date in “ `swddmmyyyy` format, ” filling solely those values that have been recognized. Unfilled characters will have "?" value.

s `Special characters`: "+", "-" for relative dates or "E" for special dates

w `Week day`: from "1" for monday to "7" for sunday. It is "0" in case of "next week" (+0????????) or in case of "last week" (-0????????)

dd `Month day`: from "01" to "31"

mm `Month`: from "01" to "12"

yyyy `Year`: since "1900"

Examples: "?12305????" ("monday 23d May"), "+???02????" ("in 2 months"), "E?2512????" ("Christmas day").

This grammar is available for Spanish and Catalan.

### 3.1.3.4. number

Natural numbers within a limited rank.

The parameters for this grammar are:

max `Values: int`

`Default: 99`

`Definition:` The maximum numerical value to be returned. The grammar cannot return values that are superior to 999999.

`Number` grammar returns the recognized number without spaces between the thousands and the hundreds. For example: “23710”.

This grammar is available for Spanish and Catalan.

### 3.1.3.5. phone

Telephone numbers in natural language, covering the most frequent ways to group digits in natural numbers (“ninety-three, four zero nine, seventy-one, twenty”).

The parameters for this grammar are:

minlength `Values: int`

`Default: 9`

`Definition:` The minimum number of digits of the telephone number.

maxlength `Values: int`

`Default: 9`

`Definition:` The maximum number of digits of the telephone number.

`Phone` grammar returns the recognized telephone number as a digit string (from “0” to “9”). For example:

“934097120”.

This grammar is available for Spanish and Catalan.

#### 3.1.3.6. currency

Currency amounts expressed in `Euros`. In case that cents are included, they will appear as decimals separated by a period from the `Euro`unit. For example: “EUR135.68”

This grammar does not have parameters.

This grammar is available for Spanish and Catalan.

#### 3.1.3.7. creditcard

It returns the recognized credit card number (spoken digit by digit) without any space in between them.

This grammar does not have parameters.

This grammar is available for Spanish, Catalan, Basque and Galician.

#### 3.1.3.8. code

Numerical codes spoken as a sequence of natural numbers (grouped together in one, two or three digits) For example: "twenty-three, five, ten" returns "23510".

The parameters for this grammar are:

length
`Values:` `int`
`Default:` `4`
`Definition:` The length of the code to be returned.

`Code` grammar returns the recognized numerical code without any spaces between the digits.

This grammar is available for Spanish and Catalan.

#### 3.1.3.9. nif

It returns the Spanish personal identification number, including the letter, without spaces in between.

This grammar does not have parameters.

This grammar is available for Spanish, Catalan, Basque and Galician.

#### 3.1.3.10. spell

Alphabetic or alphanumeric string without spaces in between.

The parameters for this grammar are:

minlength
`Values:` `int`
`Default:` `1`
`Definition:` The minimum lenght of the alpha/numeric string.

maxlength
`Values:` `int`
`Default:` `10`
`Definition:` The maximum lenght of the alpha/numeric string.

alphanumeric

`Values:` `true`, `false`

`Default:` `true`

`Definition:` Only alphabetic (false) or alphanumeric (true).

`Spell` grammar returns the recognized alpha/numeric string without spaces in between the digits.

This grammar is available for Spanish, Catalan, Basque and Galician.

### 3.1.4. Other speech recognition grammars

Speech recognition grammar specification (SRGS) in its XML form can be used if translated into its ABNF form using XSLT as specified in http://www.w3c.org/TR/speech-grammar. However, it is advisable to check the syntax of the resulting ABNF before using it in Verbio.

### 3.1.5. Specific word model

*Verbio* speech recognizer contains specific acoustic models for the most frequent and critical words in operation such as isolated and connected digits (from 0 to 9), yes/no words and letters for spelling.

Built-in basic grammars use these word models in order to increase the features for digit string and yes/no answers recognition. For example, a possible grammar for NIF numbers recognition could be:

```
#ABNF 1.0 ISO8859-1;

root $nif = $DIGITS $LETTERS;

// auxiliary grammars

$DIGITS = (( $S | $SD $DD* $DS ) "[SIL]" {} )+ ;

$LETTER =
                        "[A_es][_A_es]"  {A}  | // a
                        "[B_es][_E_es]"  {B}  | // be
                        "[C_es][_E_es]"  {C}  | // ce
                        "[D_es][_E_es]"  {D}  | // de
                        "[E_es][_E_es]"  {E}  | // e
                        "[E__es][F_es][_e_es]"  {F}  | // efe
                        "[G_es][_E_es]"  {G}  | // ge
                        "[H_es]"  {H}  | // hache
                        "[I_es]"  {I}  | // i
                        "[Il_es]" {I}  | // i latina
                        "[J_es]"  {J}  | // jota
                        "[K_es][_A_es]"  {K}  | // ca
                        "[E__es][L_es][_e_es]"  {L}  | // ele
                        "[E__es][M_es][_e_es]"  {M}  | // eme
                        "[E__es][N_es][_e_es]"  {N}  | // ene
                        "[O_es]"  {O}  | // o
                        "[P_es][_E_es]"  {P}  | // pe
                        "[Q_es][_U_es]"  {Q}  | // cu
                        "[E__es][R_es][_e_es]"  {R}  | // erre
                        "[E__es][r_es][_e_es]"  {R}  | // ere
                        "[E__es][S_es][_e_es]"  {S}  | // ese
                        "[T_es][_E_es]"  {T}  | // te
                        "[U_es][_U_es]"  {U}  | // u
                        "[V_es]"  {V}  | // uve
                        "[W_es]"  {W}  | // uve doble
                        "[dV_es]" {W}  | // doble uve
                        "[X_es]"  {X}  | // equis
                        "[Y_es]"  {Y}  | // y griega
                        "[Z_es]"  {Z};         // zeta

// auxialiary variables
$S =
                        "[S-cero_es][cero_es+S]"     {0} |
                        "[S-uno_es][uno_es+S]"       {1} |
                        "[S-dos_es][dos_es+S]"       {2} |
                        "[S-tres_es][tres_es+S]"     {3} |
                        "[S-cuatro_es][cuatro_es+S]" {4} |
                        "[S-cinco_es][cinco_es+S]"   {5} |
                        "[S-seis_es][seis_es+S]"     {6} |
                        "[S-siete_es][siete_es+S]"   {7} |
                        "[S-ocho_es][ocho_es+S]"     {8} |
                        "[S-nueve_es][nueve_es+S]"   {9} ;

$SD =
                        "[S-cero_es][cero_es+D]"     {0} |
                        "[S-uno_es][uno_es+D]"       {1} |
                        "[S-dos_es][dos_es+D]"       {2} |
                        "[S-tres_es][tres_es+D]"     {3} |
                        "[S-cuatro_es][cuatro_es+D]" {4} |
                        "[S-cinco_es][cinco_es+D]"   {5} |
                        "[S-seis_es][seis_es+D]"     {6} |
                        "[S-siete_es][siete_es+D]"   {7} |
```

```
                        "[S-ocho_es][ocho_es+D]"     {8} |
                        "[S-nueve_es][nueve_es+D]"   {9} ;

$DS =
                        "[D-cero_es][cero_es+S]"     {0} |
                        "[D-uno_es][uno_es+S]"       {1} |
                        "[D-dos_es][dos_es+S]"       {2} |
                        "[D-tres_es][tres_es+S]"     {3} |
                        "[D-cuatro_es][cuatro_es+S]" {4} |
                        "[D-cinco_es][cinco_es+S]"   {5} |
                        "[D-seis_es][seis_es+S]"     {6} |
                        "[D-siete_es][siete_es+S]"   {7} |
                        "[D-ocho_es][ocho_es+S]"     {8} |
                        "[D-nueve_es][nueve_es+S]"   {9} ;

$DD =                  "[D-cero_es][cero_es+D]"     {0} |
                        "[D-uno_es][uno_es+D]"       {1} |
                        "[D-dos_es][dos_es+D]"       {2} |
                        "[D-tres_es][tres_es+D]"     {3} |
                        "[D-cuatro_es][cuatro_es+D]" {4} |
                        "[D-cinco_es][cinco_es+D]"   {5} |
                        "[D-seis_es][seis_es+D]"     {6} |
                        "[D-siete_es][siete_es+D]"   {7} |
                        "[D-ocho_es][ocho_es+D]"     {8} |
                        "[D-nueve_es][nueve_es+D]"   {9} ;
```

**Table 4.1. Specific word models for castilian Spanish**

| *Words* | |
|---|---|
| [si_es] | It models the word "sí" |
| [no_es] | It models the word "no" |
| *Spellings* | |
| [A_es][_A_es] | It models letter A |
| [B_es][_E_es] | It models letter B |
| [C_es][_E_es] | It models letter C |
| [Ch_es][_E_es] | It models letter CH (che) |
| [CH_es] | It models letter CH (ce hache) |
| [D_es][_E_es] | It models letter D |
| [E_es][_E_es] | It models letter E |
| [E__es][F_es][_e_es] | It models letter F |
| [G_es][_E_es] | It models letter G |
| [H_es] | It models letter H |
| [I_es] | It models letter I |
| [Il_es] | It models the phrase "I latina" |
| [J_es] | It models letter J |
| [K_es][_A_es] | It models letter K |
| [E__es][L_es][_e_es] | It models letter L |
| [E__es][Ll_es][_e_es] | It models letter LL (elle) |
| [dL_es] | It models letter LL (doble ele) |
| [Ld_es] | It models letter LL (ele doble) |
| [LL_es] | It models letter LL (ele ele) |
| [E__es][M_es][_e_es] | It models letter M |
| [E__es][N_es][_e_es] | It models letter N |
| [E__es][NY_es][_e_es] | It models letter Ñ |
| [O_es] | It models letter O |
| [P_es][_E_es] | It models letter P |
| [Q_es][_U_es] | It models letter Q |
| [E__es][R_es][_e_es] | It models letter R (erre) |
| [E__es][r_es][_e_es] | It models letter R (ere) |

| | |
|---|---|
| [E__es][S_es][_e_es] | It models letter S |
| [T_es][_E_es] | It models letter T |
| [U_es][_U_es] | It models letter U |
| [V_es] | It models letter V |
| [W_es] | It models letter W (uve doble) |
| [dV_es] | It models letter W (doble uve) |
| [X_es] | It models letter X |
| [Y_es] | It models letter Y |
| [Z_es] | It models letter Z |
| *Digits in silence context (preceding and following silence)* | |
| [S-cero_es][cero_es+S] | Digit 0 |
| [S-uno_es][uno_es+S] | Digit 1 |
| [S-dos_es][dos_es+S] | Digit 2 |
| [S-tres_es][tres_es+S] | Digit to 3 |
| [S-cuatro_es][cuatro_es+S] | Digit 4 |
| [S-cinco_es][cinco_es+S] | Digit 5 |
| [S-seis_es][seis_es+S] | Digit 6 |
| [S-siete_es][siete_es+S] | Digit 7 |
| [S-ocho_es][ocho_es+S] | Digit 8 |
| [S-nueve_es][nueve_es+S] | Digit 9 |
| *Digits in context of preceding silence and following digit.* | |
| [S-cero_es][cero_es+D] | Digit 0 |
| [S-uno_es][uno_es+D] | Digit 1 |
| [S-dos_es][dos_es+D] | Digit 2 |
| [S-tres_es][tres_es+D] | Digit 3 |
| [S-cuatro_es][cuatro_es+D] | Digit 4 |
| [S-cinco_es][cinco_es+D] | Digit 5 |
| [S-seis_es][seis_es+D] | Digit 6 |
| [S-siete_es][siete_es+D] | Digit 7 |
| [S-ocho_es][ocho_es+D] | Digit 8 |
| [S-nueve_es][nueve_es+D] | Digit 9 |
| *Digits in context of preceding digit and following silence* | |
| [D-cero_es][cero_es+S] | Digit 0 |
| [D-uno_es][uno_es+S] | Digit 1 |
| [D-dos_es][dos_es+S] | Digit 2 |
| [D-tres_es][tres_es+S] | Digit 3 |
| [D-cuatro_es][cuatro_es+S] | Digit 4 |
| [D-cinco_es][cinco_es+S] | Digit 5 |
| [D-seis_es][seis_es+S] | Digit 6 |
| [D-siete_es][siete_es+S] | Digit 7 |
| [D-ocho_es][ocho_es+S] | Digit 8 |
| [D-nueve_es][nueve_es+S] | Digit 9 |
| *Digits in digit context (preceding and following digit)* | |
| [D-cero_es][cero_es+D] | Digit 0 |
| [D-uno_es][uno_es+D] | Digit 1 |
| [D-dos_es][dos_es+D] | Digit 2 |

| | |
|---|---|
| [D-tres_es][tres_es+D] | Digit 3 |
| [D-cuatro_es][cuatro_es+D] | Digit 4 |
| [D-cinco_es][cinco_es+D] | Digit 5 |
| [D-seis_es][seis_es+D] | Digit 6 |
| [D-siete_es][siete_es+D] | Digit 7 |
| [D-ocho_es][ocho_es+D] | Digit 8 |
| [D-nueve_es][nueve_es+D] | Digit 9 |

**Table 4.2. Specific word models for Catalan**

| *Words* | |
|---|---|
| [si_ca] | It models the word "sí" |
| [no_ca] | It models the word "no" |
| *Spellings* | |
| [A_ca] | It models letter A |
| [B_ca] | It models letter B |
| [B_ca][alta_ca] | It models letter B (be alta) |
| [C_ca] | It models letter C |
| [C_ca][trencada_ca] | It models letter Ç |
| [D_ca] | It models letter D |
| [E_ca] | It models letter E |
| [F_ca] | It models letter F |
| [G_ca] | It models letter G |
| [H_ca] | It models letter H |
| [I_ca] | It models letter I |
| [J_ca] | It models letter J |
| [K_ca] | It models letter K |
| [L_ca] | It models letter L |
| [LL_ca] | It models letter LL (ella) |
| [doble_ca][L_ca] | It models letter LL (doble ela) |
| [L_ca][doble_ca] | It models letter LL (ela doble) |
| [M_ca] | It models letter M |
| [N_ca] | It models letter N |
| [NY_ca] | It models letter NY |
| [O_ca] | It models letter O |
| [P_ca] | It models letter P |
| [Q_ca] | It models letter Q |
| [R_ca] | It models letter R (erre) |
| [S_ca] | It models letter S |
| [T_ca] | It models letter T |
| [U_ca] | It models letter U |
| [V_ca] | It models letter V |
| [B_ca][baixa_ca] | It models letter V (be baixa) |
| [uve_ca] | It models letter V (uve) |
| [B_ca][doble_ca] | It models letter W (be doble) |

| | |
|---|---|
| [doble_ca][B_ca] | It models letter W (doble be) |
| [X_ca] | It models letter X |
| [I_ca][grega_ca] | It models letter Y |
| [Z_ca] | It models letter Z |
| *Digits in silence context (preceding and following silence)* | |
| [S-zero_ca][zero_ca+S] | Digit 0 |
| [S-u_ca][u_ca+S] | Digit 1 |
| [S-dos_es][dos_es+S] | Digit 2 |
| [S-tres_ca][tres_ca+S] | Digit 3 |
| [S-quatre_ca][quatre_ca+S] | Digit 4 |
| [S-cinc_ca][cinc_ca+S] | Digit 5 |
| [S-sis_ca][sis_ca+S] | Digit 6 |
| [S-set_ca][set_ca+S] | Digit 7 |
| [S-vuit_es][vuit_es+S] | Digit 8 |
| [S-nou_ca][nou_ca+S] | Digit 9 |
| *Digits in context of preceding silence and following digit* | |
| [S-zero_ca][zero_ca+D] | Digit 0 |
| [S-u_ca][u_ca+D] | Digit 1 |
| [S-dos_es][dos_es+D] | Digit 2 |
| [S-tres_ca][tres_ca+D] | Digit 3 |
| [S-quatre_ca][quatre_ca+D] | Digit 4 |
| [S-cinc_ca][cinc_ca+D] | Digit 5 |
| [S-sis_ca][sis_ca+D] | Digit 6 |
| [S-set_ca][set_ca+D] | Digit 7 |
| [S-vuit_es][vuit_es+D] | Digit 8 |
| [S-nou_ca][nou_ca+D] | Digit 9 |
| *Digits in context of preceding digit and following silence* | |
| [D-zero_ca][zero_ca+S] | Digit 0 |
| [D-u_ca][u_ca+S] | Digit 1 |
| [D-dos_es][dos_es+S] | Digit 2 |
| [D-tres_ca][tres_ca+S] | Digit 3 |
| [D-quatre_ca][quatre_ca+S] | Digit 4 |
| [D-cinc_ca][cinc_ca+S] | Digit 5 |
| [D-sis_ca][sis_ca+S] | Digit 6 |
| [D-set_ca][set_ca+S] | Digit 7 |
| [D-vuit_es][vuit_es+S] | Digit 8 |
| [D-nou_ca][nou_ca+S] | Digit 9 |
| *Digits in digit context (preceding and following digit)* | |
| [D-zero_ca][zero_ca+D] | Digit 0 |
| [D-u_ca][u_ca+D] | Digit 1 |
| [D-dos_es][dos_es+D] | Digit 2 |
| [D-tres_ca][tres_ca+D] | Digit 3 |
| [D-quatre_ca][quatre_ca+D] | Digit 4 |
| [D-cinc_ca][cinc_ca+D] | Digit 5 |
| [D-sis_ca][sis_ca+D] | Digit 6 |
| [D-set_ca][set_ca+D] | Digit 7 |

| | |
|---|---|
| [D-vuit_es][vuit_es+D] | Digit 8 |
| [D-nou_ca][nou_ca+D] | Digit 9 |

**Table 4.3. Specific word models for Basque**

| *Words* | |
|---|---|
| [bai_eu] | It models the word "bai" |
| [ez_eu] | It models the word "ez" |
| *Spellings* | |
| [A_eu] | It models letter A |
| [B_eu] | It models letter B |
| [C1_eu] | It models letter C (X e) |
| [C2_eu] | It models letter C (T e) |
| [D_eu] | It models letter D |
| [E_eu] | It models letter E |
| [F_eu] | It models letter F |
| [G2_eu] | It models letter G |
| [H_eu] | It models letter H |
| [I_eu] | It models letter I |
| [J1_eu] | It models letter J (dj o t a) |
| [J2_eu] | It models letter J (x o t a) |
| [K_eu] | It models letter K |
| [L_eu] | It models letter L |
| [M_eu] | It models letter M |
| [N_eu] | It models letter N |
| [NY_eu] | It models letter Ñ |
| [O_eu] | It models letter O |
| [P_eu] | It models letter P |
| [Q_eu] | It models letter Q |
| [R_eu] | It models letter R (erre) |
| [S_eu] | It models letter S |
| [T_eu] | It models letter T |
| [U_eu] | It models letter U |
| [V_eu] | It models letterV |
| [W1_eu] | It models letter W (u B e B i k o i tX a) |
| [W2_eu] | It models letter W (u B e d o B l e) |
| [W3_eu] | It models letter W (d o B l e u B e) |
| [X1_eu] | It models letter X (i S a) |
| [X2_eu] | It models letter X (e k i s) |
| [X3_eu] | It models letter X (i k s a) |
| [Y1_eu] | It models letter Y (i G rr e k o a) |
| [Y2_eu] | It models letter Y (i G rr i e G a) |
| [Y3_eu] | It models letter Y (i G rr e k o t a rr a) |
| [Y4_eu] | It models letter Y (i G rr e X i a rr a) |
| [Y5_eu] | It models letter Y (i G rr e k e r a) |

| | |
|---|---|
| [Z1_eu] | It models letter Z (X e t a) |
| [Z2_eu] | It models letter Z (s e t a) |
| [Z3_eu] | It models letter Z (T e t a) |
| [Z4_eu] | It models letter Z (X e D a) |
| *Digits in silence context (preceding and following silence)* | |
| [S-zero_eu][zero_eu+S]<br>[S-huts_eu][huts_eu+S]<br>[S-hutsa_eu][hutsa_eu+S] | Digit 0 |
| [S-bat_eu][bat_eu+S] | Digit 1 |
| [S-bi_eu][bi_eu+S] | Digit 2 |
| [S-hiru_eu][hiru_eu+S] | Digit 3 |
| [S-lau_eu][lau_eu+S] | Digit 4 |
| [S-bost_eu][bost_eu+S] | Digit 5 |
| [S-sei_eu][sei_eu+S] | Digit 6 |
| [S-zazpi_eu][zazpi_eu+S] | Digit 7 |
| [S-zortzi_eu][zortzi_eu+S] | Digit 8 |
| [S-bederatzi_eu][bederatzi_eu+S] | Digit 9 |
| *Digits in context of preceding silence and following digit* | |
| [S-zero_eu][zero_eu+D] | Digit 0 |
| [S-bat_eu][bat_eu+D] | Digit 1 |
| [S-bi_eu][bi_eu+D] | Digit 2 |
| [S-hiru_eu][hiru_eu+D] | Digit 3 |
| [S-lau_eu][lau_eu+D] | Digit 4 |
| [S-bost_eu][bost_eu+D] | Digit 5 |
| [S-sei_eu][sei_eu+D] | Digit 6 |
| [S-zazpi_eu][zazpi_eu+D] | Digit 7 |
| [S-zortzi_eu][zortzi_eu+D] | Digit 8 |
| [S-bederatzi_eu][bederatzi_eu+D] | Digit 9 |
| *Digits in context of preceding digit and following silence* | |
| [D-zero_eu][zero_eu+S] | Digit 0 |
| [D-bat_eu][bat_eu+S] | Digit 1 |
| [D-bi_eu][bi_eu+S] | Digit 2 |
| [D-hiru_eu][hiru_eu+S] | Digit 3 |
| [D-lau_eu][lau_eu+S] | Digit 4 |
| [D-bost_eu][bost_eu+S] | Digit 5 |
| [D-sei_eu][sei_eu+S] | Digit 6 |
| [D-zazpi_eu][zazpi_eu+S] | Digit 7 |
| [D-zortzi_eu][zortzi_eu+S] | Digit 8 |
| [D-bederatzi_eu][bederatzi_eu+S] | Digit 9 |
| *Digits in digit context (preceding and following digit)* | |
| [D-zero_eu][zero_eu+D] | Digit 0 |
| [D-bat_eu][bat_eu+D] | Digit 1 |
| [D-bi_eu][bi_eu+D] | Digit 2 |
| [D-hiru_eu][hiru_eu+D] | Digit 3 |
| [D-lau_eu][lau_eu+D] | Digit 4 |

| | |
|---|---|
| [D-bost_eu][bost_eu+D] | Digit 5 |
| [D-sei_eu][sei_eu+D] | Digit 6 |
| [D-zazpi_eu][zazpi_eu+D] | Digit 7 |
| [D-zortzi_eu][zortzi_eu+D] | Digit 8 |
| [D-bederatzi_eu][bederatzi_eu+D] | Digit 9 |

**Table 4.4. Specific word models for Galician**

| *Words* | |
|---|---|
| [si_ga] | It models the word "sí" |
| [non_ga] | It models the word "non" |
| *Spellings* | |
| [A_ga] | It models letter A |
| [B_ga] | It models letter B |
| [C_ga] | It models letter C |
| [CH_ga] | It models letter CH |
| [D_ga] | It models letter D |
| [E_ga] | It models letter E |
| [F_ga] | It models letter F |
| [G_ga] | It models letter G |
| [H_ga] | It models letter H |
| [I_ga] | It models letter I |
| [J_ga] | It models letter J |
| [K_ga] | It models letter K |
| [L_ga] | It models letter L |
| [LL_ga] | It models letter LL |
| [dobre_ga][L_ga] | It models letter LL |
| [L_ga][dobre_ga] | It models letter LL |
| [M_ga] | It models letter M |
| [N_ga] | It models letter N |
| [NY_ga] | It models letter NY |
| [O_ga] | It models letter O |
| [P_ga] | It models letter P |
| [Q_ga] | It models letter Q |
| [R_ga] | It models letter R (erre) |
| [S_ga] | It models letter S |
| [T_ga] | It models letter T |
| [U_ga] | It models letter U |
| [V_ga] | It models letter V |
| [V_ga][dobre_ga] | It models letter W |
| [dobre_ga][V_ga] | It models letter W |
| [X_ga] | It models letter X |
| [Y_ga] | It models letter Y |
| [Z_ga] | It models letter Z |
| *Digits in silence context (preceding and following silence)* | |

| | |
|---|---|
| [S-cero_ga][cero_ga+S] | Digit 0 |
| [S-un_ga][un_ga+S] | Digit 1 |
| [S-dous_ga][dous_ga+S] | Digit 2 |
| [S-tres_ga][tres_ga+S] | Digit 3 |
| [S-catro_ga][catro_ga+S] | Digit 4 |
| [S-cinco_ga][cinco_ga+S] | Digit 5 |
| [S-seis_ga][seis_ga+S] | Digit 6 |
| [S-sete_ga][sete_ga+S] | Digit 7 |
| [S-oito_ga][oito_ga+S] | Digit 8 |
| [S-nove_ga][nove_ga+S] | Digit 9 |
| *Digits in context of preceding silence and following digit* | |
| [S-cero_ga][cero_ga+D] | Digit 0 |
| [S-un_ga][un_ga+D] | Digit 1 |
| [S-dous_ga][dous_ga+D] | Digit 2 |
| [S-tres_ga][tres_ga+D] | Digit 3 |
| [S-catro_ga][catro_ga+D] | Digit 4 |
| [S-cinco_ga][cinco_ga+D] | Digit 5 |
| [S-seis_ga][seis_ga+D] | Digit 6 |
| [S-sete_ga][sete_ga+D] | Digit 7 |
| [S-oito_ga][oito_ga+D] | Digit 8 |
| [S-nove_ga][nove_ga+D] | Digit 9 |
| *Digits in context of preceding digit and following silence* | |
| [D-cero_ga][cero_ga+S] | Digit 0 |
| [D-un_ga][un_ga+S] | Digit 1 |
| [D-dous_ga][dous_ga+S] | Digit 2 |
| [D-tres_ga][tres_ga+S] | Digit 3 |
| [D-catro_ga][catro_ga+S] | Digit 4 |
| [D-cinco_ga][cinco_ga+S] | Digit 5 |
| [D-seis_ga][seis_ga+S] | Digit 6 |
| [D-sete_ga][sete_ga+S] | Digit 7 |
| [D-oito_ga][oito_ga+S] | Digit 8 |
| [D-nove_ga][nove_ga+S] | Digit 9 |
| *Digits in digit context (preceding and following context)* | |
| [D-cero_ga][cero_ga+D] | Digit 0 |
| [D-un_ga][un_ga+D] | Digit 1 |
| [D-dous_ga][dous_ga+D] | Digit 2 |
| [D-tres_ga][tres_ga+D] | Digit 3 |
| [D-catro_ga][catro_ga+D] | Digit 4 |
| [D-cinco_ga][cinco_ga+D] | Digit 5 |
| [D-seis_ga][seis_ga+D] | Digit 6 |
| [D-sete_ga][sete_ga+D] | Digit 7 |
| [D-oito_ga][oito_ga+D] | Digit 8 |
| [D-nove_ga][nove_ga+D] | Digit 9 |

**Table 4.5. Specific word models for Portuguese**

| *Words* | |
|---|---|
| [sim_pt] | It models the word "sim" |
| [nao_pt] | It models the word "não" |
| *Digits in silence context (preceding and following silence)* | |
| [S-zero_pt][zero_pt+S] | Digit 0 |
| [S-um_pt][um_pt+S] | Digit 1 |
| [S-dois_pt][dois_pt+S] | Digit 2 |
| [S-tres_pt][tres_pt+S] | Digit 3 |
| [S-quatro_pt][quatro_pt+S] | Digit 4 |
| [S-cinco_pt][cinco_pt+S] | Digit 5 |
| [S-seis_pt][seis_pt+S] | Digit 6 |
| [S-sete_pt][sete_pt+S] | Digit 7 |
| [S-oito_pt][oito_pt+S] | Digit 8 |
| [S-nove_pt][nove_pt+S] | Digit 9 |
| *Digits in context of preceding silence and following digit* | |
| [S-zero_pt][zero_pt+D] | Digit 0 |
| [S-um_pt][um_pt+D] | Digit 1 |
| [S-dois_pt][dois_pt+D] | Digit 2 |
| [S-tres_pt][tres_pt+D] | Digit 3 |
| [S-quatro_pt][quatro_pt+D] | Digit 4 |
| [S-cinco_pt][cinco_pt+D] | Digit 5 |
| [S-seis_pt][seis_pt+D] | Digit 6 |
| [S-sete_pt][sete_pt+D] | Digit 7 |
| [S-oito_pt][oito_pt+D] | Digit 8 |
| [S-nove_pt][nove_pt+D] | Digit 9 |
| *Digits in context of preceding digit and following silence* | |
| [D-zero_pt][zero_pt+S] | Digit 0 |
| [D-um_pt][um_pt+S] | Digit 1 |
| [D-dois_pt][dois_pt+S] | Digit 2 |
| [D-tres_pt][tres_pt+S] | Digit 3 |
| [D-quatro_pt][quatro_pt+S] | Digit 4 |
| [D-cinco_pt][cinco_pt+S] | Digit 5 |
| [D-seis_pt][seis_pt+S] | Digit 6 |
| [D-sete_pt][sete_pt+S] | Digit 7 |
| [D-oito_pt][oito_pt+S] | Digit 8 |
| [D-nove_pt][nove_pt+S] | Digit 9 |
| *Digits in digit context (preceding and following digit)* | |
| [D-zero_pt][zero_pt+D] | Digit 0 |
| [D-um_pt][um_pt+D] | Digit 1 |
| [D-dois_pt][dois_pt+D] | Digit 2 |
| [D-tres_pt][tres_pt+D] | Digit 3 |
| [D-quatro_pt][quatro_pt+D] | Digit 4 |
| [D-cinco_pt][cinco_pt+D] | Digit 5 |
| [D-seis_pt][seis_pt+D] | Digit 6 |

| | |
|---|---|
| [D-sete_pt][sete_pt+D] | Digit 7 |
| [D-oito_pt][oito_pt+D] | Digit 8 |
| [D-nove_pt][nove_pt+D] | Digit 9 |

### Caution

`ABNF` grammar models should be put in quotation marks (see the example above). However, conventional vocabularies are written without quotations marks.

In very specific applications or in those that operate in noisy conditions, etc. specific models of the phrases to be recognized can be trained (even for every operator), by integrating them automatically inside *Verbio*. By doing this, it is possible to achieve a significant increase on the recognition rates in specific work environments or under the use of the system by trained users or both of them.

# 4. Using the grammars

As described in the previous chapter, the recognizer needs a recognition grammar at its disposal that describes the permitted language at a particular moment in the dialogue.

## 4.1. Creating a grammar

Recognition grammars are text files that can be edited by using any editor. For ABNF grammars there is the tool`Verbio Grammar Manager`, which enables editing but also compiling, testing with text strings or voice (microphone), generating sentences permitted by the grammar, etc.
For more information on `Verbio Grammar Manager` refer to `Verbio Software Reference: Grammar Manager User's Guide`.

Section 3, “Recognition grammars” specifies the correct format of the grammars to use in Verbio.

## 4.2. Preparing the vocabulary

Once a grammar has been created, preparation is needed for using it in*Verbio*'s recognition system. Preparing (or compiling) a grammar means processing the grammar file in order to obtain the language structure (isolated or connected words if specified or the complex node structure in case of ABNF grammars) and to extract a word vocabulary that needs to be transcribed into phonemes recognized by the system.

Therefore, two types of errors can ocurr: a) structure type, for example a rule uses a reference which is not specified in the grammar, or b) phonetic type, when the word does not have a standard transcription in the specified language. In the first case, the grammar edition tool `Verbio Grammar Manager` makes it possible to identify these errors more easily. In the second case, the system points out the word causing transcription problems. The solution implies writing correctly the word or an equivalent form in the same language in case of foreign words or acronyms.

### Important

It is necessary to prepare a vocabulary every time that *Verbio*'s version is updated, as the phonetic modeling might have changed. Likewise, it is only necessary to prepare the vocabulary once for all the resources. That is why it is advisable to carry out the process *off-line*, if possible, or at the time of starting up an application (it consumes a lot of`hardware`resources).

## 4.3. Activating a grammar for the recognition

Once a recognition grammar is prepared for its use in*Verbio*, it needs to be activated so that utterances can be recognized on its basis. The activation process consists in loading (`load`) in memory and activating (`activate`). The other way around, there are the deactivating (`deactivate`) and unloading (`unload`) processes.

Loading in memory reduces computational load when activating/deactivating some grammars dynamically.

### Important

It is essential that at least one grammar is found active before trying to recognize an utterance.

### 4.3.1. Activating multi-grammars

The recognition system needs at least one active grammar to be able to recognize, but this does not mean that various grammars cannot coexist simultaneously. That is, *Verbio* enables that two or more grammars are activated simultaneously, this being useful when waiting for an answer from the user (for example a date) but the user has the choice to answer with an alternative option (for example a menu-type answer “repeat”, “home”, etc). The resulting active pseudo-grammar will be the optional grammar that enables the recognition of any of the expressions considered independently by active grammars.

*Verbio* makes it possible to know to which of the active grammars belongs the text string recognized by the system.

# 5. Recognizing an utterance

As mentioned in Section 4, “Using the grammars”, it is essential that there are one or various active grammars before starting the recognition process of an utterance.

## 5.1. Activating speech recognition

When activating the recognition of an utterance, termination conditions of this process must be stated. There are time conditions and/or conditions of events external to voice, for instance if the user presses a telephone key (DTMF tones detection).

Time conditions used in*Verbio*for recognition are mainly the following three:

| | |
|---|---|
| `Timeout (initsil)` | It is the maximum silence allowed since the recognition process is activated until the user starts speaking. Therefore, it is the initial silence or the time permitted by the system for the user to speak before it finishes and decides that the user does not respond. This parameter is indicated with the variable `initsil` in the various SDK. |
| `Complete timeout (maxsil)` | It is the maximum silence allowed once voice from the user has been detected. Therefore, it is the silence between the moment that the user stops speaking and the moment the system decides to finish the recognition. Consequently, it is also the maximum silence allowed between words. It is important to configure appropriately this parameter according to the type of grammar. This parameter is indicated with the variable `maxsil` in the various SDK. |
| `Maxspeech timeout (maxtime)` | It is the maximum time of an utterance, and it includes voice as well as silence. That is, it is the maximum time of recognition regardless of voice and silence use. This parameter is indicated with the variable`maxtime` in the various SDK. |

Special termination conditions that are platform-dependant need to be added to the previous termination conditions. In case of telephone applications, telephone keys need to be considered, as in many occasions the user is offered the possibility of using either the voice or the DTMF keyboard to answer certain questions. In this case, using the keyboard to answer implies not using the voice.

Activating the recognition process might involve, depending on the SDK and the platform, activating the optional beep that indicates the beginning of the recognition.

## 5.2. Voice samples format

Activating the recognition involves starting to send voice samples from the audio acquisition platform to the recognizer. The recognizer needs to know the form in which the samples will be sent. That is, sample frequence (by default and in telephone applications: 8 KHz) and sample format (linear with 16 bytes, 8 bytes in A-law and U-law)

## 5.3. Barge-in

In certain platforms (such as Intel Dialogic cards including CSP echo cancelation) recognition can work simultaneously with audio reproduction from files or from the text-to-speech converter. *Verbio* provides functions to activate audio reproduction together with speech recognition without barging-in the reproduction.

This means that the user can speak before the end of the audio reproduction (the question) by the system.

# 6. Obtaining the result

## 6.1. Reliability measures

Recognition verification (i.e. assigning a reliability measure to the recognition result) is a task as complex or more so than recognition itself. Verification is based on comparing two independent systems' probabilities with the input utterance itself. As a result a *score* is obtained. This is an unlimited value that indicates whether the obtained recognition is “reliable” and, therefore, it allows rejecting some recognition results, or else it suggests confirmation according to the application error tolerance.

In general,`score` values range between 0 and 40 for non-reliable results and are higher than 50 for reliable results or result with a high reliability rate. Moreover, these values are dependent on the vocabulary being used and therefore the acceptance and rejection threshold need to be adjusted according to the grammar and its vocabulary in order to make the recognition process more precise. Therefore, it is advisable, during the development and testing stage, to carry out a study on the common values and establish, for each case, an acceptance threshold and a rejection one. By acceptance threshold we refer to the threshold above which results returned by the recognizer are accepted. By rejection threshold we refer to the threshold below which results are not accepted and therefore, the question needs to be repeated. In case that the `score` is in between both thresholds, it is advisable to make and explicit confirmation (yes/no) of the result.

## 6.2. Semantic interpretation

The result of the recognition can be consulted in regard to the recognized words, thus obtaining the number of recognized words together with its reliability measure and the time frames where the word has been detected.

When ABNF grammars are used, there exist result processing functions that enable semantic interpretation of the results based on the format used when writing the grammar. For instance, it can be used to consult the recognized words belonging to a specific grammar rule, to know if the user has filled any particular field, etc. Moreover, it is possible to obtain a reliability measure for the specified rule and the time in which this part of the utterance has been detected.

This result is obtained in the various SDK in the text string to be processed by the programmer, otherwise it can be consulted by means of specific functions like `VerbioResult` in `Advanced SDK` (see appropriate information).

## 6.3. Multiple hypothesis

In some applications it is interesting to have at one's disposal not only the most likely word sequence but also other hypothesis about the utterance. A later processing of the set of most likely sequences (according to their length, structure, checksum, whether they are found in any database,etc) can enable selecting with great confidence the recognized sequence.

For instance, if at the time of recognizing a digit sequence of a phone number the most likely hypothesis are 994506789, 934506789 and 9345067899, we can reject the first and last ones because they do not match with any valid telephone number. Likewise, this method can also be used in the recognition of control codes including digits or letters such as credit cards or Spanish personal identification numbers (NIF).

The number of hypothesis to be returned by the recognizer has to be made specific before activating the recognition with *N-Best* parameter.

# Chapter 5. Text-to-speech conversion

Converting text to speech makes it possible for an application to transform a written text into speech. That is, it enables that the machine can “read aloud” any text.

In order to do so, it uses an intelligent expansion of numbers, dates, abbreviations, etc., that is able to become personal thanks to its own dictionaries. By means of a standard markup language we can control the converter's behaviour, in order to change the language or the speaker, alter utterance speed, enter pauses, apply for notifications, etc.

This chapter discusses how to work with *Verbio* text-to-speech converter and describes the most important concepts and steps implied in the process.

## 1. Initialization

Initializing the text-to-speech conversion engine entails enabling one ore more languages and speakers so they can be used in the application as well as specifying the default language and speaker (see initialization functions in Function Reference for further detail or the Section 6.3, “Specifying speakers of interest”).

## 2. Basic concepts

Text-to-speech conversion is completely automatic and that is why it is exclusively based on the text to be converted. For that reason, it is extremely important that the text is well written, specially regarding punctuation, accents and typographic and/or ortographic mistakes. Note that although a person is able to read without any problem a written text such as “9th October”, it won't probably be so with the voice resulting from the text synthesis.

In general, it is best to facilitate as far as possible the task of converting text to speech. For instance: introducing appropriate punctuation marks when the text to be synthesized is known (it is not like so for example when it comes to email reading), desambiguating abbreviations and exceptions (by expanding them or by entering dictionaries personalized to the use context planned), standardizing dates, currency amounts or any sort of non-normalized text under control.

*Verbio* provides `Verbio Read Aloud`, a tool which allows testing all speakers incorporated in *Verbio*, as well as their behaviour when configuration changes occur, dictionaries, etc. More information on this tool can be found in the associated documentation `Verbio Software Reference: Guía del Usuario de Read Aloud`.

## 3. Speech synthesis markup

### 3.1. Speech Synthesis Markup Languages

Markup languages for text-to-speech conversion systems appear as tools to add additional information to a text that will by synthesized. They enable specifying paralinguistic (such as utterance speed or sex of the speaker) or text interpretation concepts in ambiguous segments, such as delimiting that a text segment matches with a date.

#### 3.1.1. SSML

`W3C` (http://www.w3.org/TR/speech-synthesis/) develops a markup standard based on XML that is called`Speech Synthesis Markup Language` (SSML). Next you will find the `labels`supported by *Verbio*. Further information can be found in `W3C` webpage.

| Etiqueta | Atributos | Valores permitidos |
|---|---|---|
| p, paragraph | xml:lang | código del idioma |
| say-as | interpret-as | "date, "time", "literal", "telephone", "currency", "cardinal", "ordinal", "digits" |
| | format | "dmy", "mdy", "ymd", "dm", "md", "my", "ym", "d", "m", "y", "hms", "hm", "ms", "h", "m", "s" |
| | detail | *atributo no soportado* |
| sub | alias | texto a utilizar para la sustitución |
| voice | xml:lang | código del idioma |
| | gender | "male", "female" |
| | age | entero representando la edad del locutor |
| | variant | *atributo no soportado* |
| | name | nombre de la voz a utilizar |
| emphasis | level | "strong", "moderate", "none", "reduced" |
| prosody | pitch | "x-high", "high", "medium", "low", "x-low", "default"<br>o bien especificar un valor absoluto (p. ej.: "100Hz") o un cambio relativo (p. ej.: "-20%", "+10Hz", "-.5st") |
| | contour | *atributo no soportado* |
| | range | "x-high", "high", "medium", "low", "x-low", "default"<br>o bien especificar un valor absoluto (p. ej.: "40Hz") o un cambio relativo (p. ej.: "-20%", "+10Hz", "-.5st") |
| | rate | "x-fast", "fast", "medium", "slow", "x-slow", "default"<br>o bien especificar un valor absoluto (p. ej.: "180") o un cambio relativo, por ejemplo: (p. ej.: "+10%", "+20") |
| | duration | *atributo no soportado* |
| | volume | "silent", "x-soft", "soft", "medium", "loud", "x-loud", "default"<br>o bien especificar un valor absoluto (entre 0 y 100) o un cambio relativo, (p. ej.: "-20%", "+20") |
| break | strength | none, "x-small", "small", "medium", "large", "x-large" |
| | time | duración de la pausa, por ejemplo: "250ms", "3s" |
| audio | src | URI del documento |
| desc | - | - |
| mark | name | identificador de la marca |

List of SSML `labels` supported by *Verbio*.

`Verbio Read Aloud` enables modifying *Verbio* speech synthesizer behaviour by using the SSML labels shown below.

### 3.1.2. SABLE

SABLE is a markup language for speech synthesis systems that has appeared thanks to combining 3 previously existing languages: SSML, STML and JSML. However, currently, it is narrowly used as SSML has been further standardized through that of W3C. Further information can be found in http://www.research.att.com/~rws/SABPAP/sabpap.htm and http://www.cstr.ed.ac.uk/projects/sable/. SABLE labels supported by *Verbio* are:

- BREAK
- RATE
- VOLUME
- AUDIO
- ENGINE
- MARKER
- LANGUAGE
- SABLE
- SPEAKER
- DIV

# 4. Exception and abbreviation dictionary

*Verbio* uses exception and abbreviation dictionaries in order to expand correctly non-normalized words to a form that has a direct correspondence with its pronunciation. Besides the dictionaries that are already incoroporated for each language, *Verbio* enables establishing dictionaries for each port and changing dictionaries dinamically according to the application.

## 4.1. User exception dictionary

Exception dictionaries contain those words that have a non-standard pronunciation. That is to say that its phonetic transcription does not comply with its language standard rules. Neologisms and foreign words in general, acronyms, etc. are considered exceptions.

Each entry in these dictionaries includes the word exception (in a single word without spaces) and the replacing word (or words) written according to its standard pronunciation, separated by the equal sign “=”. For instance, in Spanish:

```
hardware = “jarguar”
```

First, the words are searched in the dictionary as they appear in the text and then in small letters. If for instance the word “Jordi”is found in the text, a search is made for “Jordi” and if it is not found, a search will be made for “jordi”. Therefore, if

```
Jordi = “llordi”
```

is written in the dictionary, it will read the word “Jordi” in pseudo-Catalan but not “JORDI” neither “jordi”. Otherwise, if

```
jordi = “llordi”
```

is written in the dictionary, it will read the word “Jordi”, “JORDI”, “jordi”, etc. in pseudo-Catalan.

## 4.2. User abbreviation dictionary

Abbreviation dictionaries contain words that are abbreviations. They have the same format as the exception dictionaries. In this case, the first field is the abbreviation to be replaced.

When a full stop is included in the first field of the abbreviation dictionary, the text-to-voice conversor knows that this abbreviation is not the end of a sentence (for example, “Dr.”). Therefore, if there is a full stop following the abbreviation, the text-to-voice conversor always interprets it as part of the abbreviation. If the abbreviation does not include a full stop, the coversor knows that, opposite to the previous case, this abbreviation might be the end of a sentence (for example, “pta”). In this case, if there is a full stop following the abbreviation, the text-to-voice conversor interprets it as part of the abbreviation if the following letter is written in small letters, or as the end of the sentence if the following letter is written in capital letters.

# Index