Vox Studio 2.05 / 3.0 Command Line Reference for Programmers

Thre are two ways you can use the Vox Studio file conversion capabilities from your own application : you can use this command-line capability or you can use the newer file conversion DLL capability. This is the reference documentation for the command-line. The conversion DLL documentation is available in a separate document.

Vox Studio command line parameters:

Example:

```
voxtudio /Sc:\in\test.wav /Dc:\out\test.vox /I0 /O41 /Q8000
```

Flags can appear in any order. The maximum length of a command line under Windows is 128 characters. If you need more characters, use the /F command first to store defaults.

You HAVE to give all the parameters to Vox Studio in order to use the command line feature. If you forget to enter some parameters errors can occur. If the calling application is not in the same directory as Vox Studio you should precede the command with the full path to the Vox Studio program.The command-line defaults are different from the normal "manual operation" Vox Studio defaults.You can give these parameters all at once on the conversion command line itself or you can use one or more /F default setting command(s) to set the default command line conversion parameters. Each /F command sets the parameter defaults it has on its own command line and leaves the other parameter defaults untouched. The /V flag allows verbose operation and writes a log file.

Here is a complete description of the flags known to Vox Studio. Flags are shown in upper case, but Vox Studio also accepts lower case flags and parameters, or a mixture of both.

Available flags:

/F:
Syntax: /F
or /Fs
where s is a single character.
Purpose: to indicate that the other parameters on the command line will be stored as default values for subsequent conversions. If this flag is absent, then Vox Studio is in conversion mode, using the other parameters for (and only for)

the current conversion; if it is present, Vox Studio goes in configuration mode and only stores the other parameters for use as default values for the following conversions. The character s can take on two values: Y (for Yes) or N (for No). It indicates if the changes in configuration parameters must to be reported in the log file (see flag /V below). The absence of this character is equivalent to Yes (/F is identical to /FY).

/S:
Syntax: /Sc:\voxtudio\input
or /Sd:\files\vox\test.vox
or /Stest.vox
Purpose: to give the path and the name of the input sound file to be converted. This string following this flag must give a filename (with or without a path) when converting, but must only be a valid path (without a filename) in configuration mode (flag /F is present). If no input path was ever given to Vox Studio, then the program uses it's own path as a default.

/D:
Syntax: /Dc:\voxtudio\output
or /Dd:\files\wav\test.wav
or /Dtest.wav
Purpose: to give the path and filename of the output converted sound file.
This flag must give a filename (with or without a path) when converting, but must only be a valid path (without a filename) in configuration mode (flag /F is present). If no output path was ever given to Vox Studio, then the program uses it's own path as a default.

/I:
Syntax: /Id where d is an integer number.
Purpose: to indicate to Vox Studio the type of sound format of the input file.See below for a list of the sound format codes. If no input format code was ever given to Vox Studio, it defaults to Windows wave.

/O:
Syntax: /Od where d is an integer number.
Purpose: to indicate to Vox Studio the type of sound format of the output file. See below for a list of the sound format codes. If no input format code was ever given to Vox Studio, it defaults to Windows wave, 8 bit linear mono.

/R:
Syntax: /Rl where l is a long integer number.
Purpose: to give to Vox Studio the sampling frequency of the input file. In some cases, this information is not necessary. This happens if the input sound file has a header in which this info is given. For example, Microsoft .wav files have such a header. For such files, if this info is given anyway, it will be overridden by the value of the header. A table at the end of this document shows all allowed sampling frequencies for the various sound formats recognized by Vox Studio. If

this flag is not present when converting a sound file, the previously configured value is used. If no input sampling frequency was ever indicated to Vox Studio, it defaults to 8000 Hz.

/Q:
Syntax: /Ql where l is a long integer number.
Purpose: to give to Vox Studio the sampling frequency of the output file. See the /R flag above for additional infos.

/M:
Syntax: /M1 or /M2.
Purpose: indicates if the input file is a mono (1) or a stereo (2) sound file. If this flag is not present when converting a sound file, the previously configured value is used. If this parameter was never indicated to Vox Studio, it defaults to mono. If a sound file with a header gives this information, the value on the command line will be overridden by the one in the header.

/L:
Syntax: /L
or /Ls
or /Lsd;l1;l2
where s is a character, d, l1 and l2 are integer numbers.
Purpose: to tell Vox Studio if it must perform a Leader/Trailer silence adjustment when converting. The character s can take the two values Y (for Yes) or N (for No). The first integer number d is the threshold in percent full scale of the sound amplitude under which a signal is considered silence. The second number l1 is the length in milliseconds to which the silence at the beginning (the leader) of the output file must be adjusted. l2 is the length of the adjusted ending silence (the trailer), also in milliseconds. The numbers must be separated by semi-colons. The character and the numbers are optional, but if one number is given then they must appear all three, and the leading s character must be given too. If no number is given, the previously configured values are used. The syntax /L is equivalent to /LY. If no values were ever given to Vox Studio for Leader/Trailer adjustment, it defaults to no adjustment, threshold = 5%, lengths = 500 msec.

/N:
Syntax: /N
or /Ns
or /Nsd
where s is a character and d an integer number.
Purpose: to instruct Vox Studio to perform a volume normalization of the sound during the conversion process. The character s can take the two values Y (for Yes) or N (for No). The integer number d is the threshold in percent full scale of the sound amplitude to which the sound will be normalized. The character and the number are optional, but if the number is given then the leading s character must be given too. If no number is given, the previously configured value is used.

The syntax /N is equivalent to /NY. If no values were ever configured in Vox Studio (/F flag) for normalization, it defaults to no normalization, and 75% of full scale energy.

/C:
Syntax: /C
or /Cs
where s is a single character.
Purpose: to indicate if centering of the sound around the mean value must take place during conversion. The character s can take the two values Y (for Yes) or N (for No). The syntax /C is equivalent to /CY. If this flag is not given on the command line during a conversion, the previously configured value is used. If this flag was never configured, the default is No.

/G:
Syntax: /G
or /Gs1
or /Gs1s2s3
where s1, s2 and s3 are three single characters.
Purpose: to ask Vox Studio to enhance the quality of the output sound file by using the clarity filter and/or by boosting the signal with the boost option. The character s1 can take the two values Y (for Yes) or N (for No). The character s2 can be W (for Weak), S (for Strong) or N (for No) and relate to the clarity filter. The character s3 can also take the three values S, W, N (same meaning as above) and govern the strength of the boosting process. The characters are optional, but if s2 or s3 are given, then they must all appear, including s1. The syntax /G is equivalent to /GY. If this operation was never configured in Vox Studio, the defaults are no enhancement, filtering and boosting both set to Weak.

/V:
Syntax: /V
or /Vs1
or /Vs1s2
or /Vs1s2c:\voxtudio\voxcmdln.log
Purpose: to instruct Vox Studio to write to a log file the succession of operations and their success or failure, together with the date and time. If the flag alone is present on the command line, all operations will be reported in the current log file. The character s1 can take the two values Y (for Yes) or N (for No). If Yes, the operations will be reported to the log file; if No, no report will be generated. The character s2 can also take the two values Y or N. If it’s value is Y, then the previously generated log file will be erased before writing the current report; if N, then the report is appended to the current log file. If s2 is present, then s1 must also be given. The remaining string for this flag indicates to Vox Studio, if present, the name and path of the log file the report is to be written to. If it is present, then s1 and s2 must also be present. If no pathname is given, the

previously configured log file pathname will be used. In case Vox Studio has a problem writing to or generating the log file, it will simply beep to indicate the problem, but will nevertheless continue the current operation, without any report. If the log file parameters were never given to Vox Studio, then the program defaults to it's own path and to the filename voxcmdln.log, with verbose being on. The syntax /V is equivalent to /VY

/E:

Syntax: /E
or /Es
or /Esc:\voxtudio\voxcmdln.end

Purpose: to instruct Vox Studio to write a special file at the end of the conversion. This file is overwritten at the end of each conversion performed by Vox Studio. It is an ASCII file comprising 2 lines (separated by a carriage return and a line feed). The first line gives the complete pathname for the output file just created by the current conversion. The second line is either OK if the conversion succeeded, or BAD if the conversion failed. This file indicates to the command line calling program if Vox Studio has finished the conversion process, and if it was successful or not. Once the file is written to disk, Vox Studio stops running and is ready for the next command line operation. The calling program can easily check for the existence of this file to know if the conversion process has come to an end, check if the conversion was OK and then erase the file and send the next command line to Vox Studio, which will re-create it at the end of it's task, and so on… The character s can take the two values Y (for Yes) or N (for No). If Yes, the file will be created; if No, it will not. The remaining string for this flag indicates to Vox Studio, if present, the name and path of the file to be created. If it is present, then s must be too. If no pathname is given, the previously configured file pathname will be used. If the file pathname was never given to Vox Studio, then the program defaults to it's own path and to the filename voxcmdln.end. The syntax /E is equivalent to /EY

/H

Syntax: /Hxxxx
where xxxx is the ASCII translation of the HWND handle of the calling program.
Purpose: To enable Vox Studio to post a message to the calling program at the completion of a conversion task. This special message is VS_RETURNCODE and has a value of 40007. The wParam of the message sent to the calling program will contain the return code of Vox Studio. This procedure offers a more straightforward way of communicating the result (success or failure, and why) of a command-line-initiated Vox Studio conversion. It is intended for programmers, and has been introduced specifically to overcome the difficulty of retrieving program closure return codes in a 16 bits environment (Win 3.1). The error value returned is the same as the return code defined elsewhere in this documentation.

Input codes for the sound formats known to Vox Studio:

Some sound format files have a header with all the necessary information in it: compression algorithm type, sample rate, stereo or mono, … Some of these even have a specific signature which makes them uniquely recognizable. And some of them have no header at all. The following description and codes for all the formats known to Vox Studio give this information. Of course, for files with a header, Vox Studio uses the info contained in this header for the decompression (this will override any info given on the command line for the input file format, sample rate, …).
Following are the codes to be used to read files of a particular sound format.

Microsoft Wave: This is a format with a header containing all necessary information for decompression. In this case, the input code is the same for all formats known to Vox Studio, and the true code will be generated from the header.

- 8 bit linear: 0
- 16 bit linear: 0
- A-law: 0
- Mu-law: 0
- Dialogic 32K ADPCM: 0
- OKI 32K ADPCM: 0
- MS 4bit ADPCM: 0
- IMA 3 bit ADPCM: 0
- IMA 4 bit ADPCM: 0
- NMS G726 32K: 0

Raw PCM: This is a headerless format.

- 8 bit linear: 20
- 16 bit linear: 21
- A-law: 22
- Mu-law: 23

Dialogic: This is a headerless format.

- ADPCM: 40
- A-law: 41
- Mu-law: 42

Dialogic Wave: This is a format with a header containing all necessary information for decompression.

- 32K ADPCM: 43
- A-law: 43
- Mu-law: 43

ITU-CCITT: This is a headerless format.

- G711 A-law: 60
- G711 Mu-law: 61

G721 32K: 62
G726 40K: 63
G726 32K: 64
G726 24K: 65
G726 16K: 66

Natural MicroSystems: Two different species coexist: the VCE one which is a single prompt headerless format; the VOX one which is an indexed multi-prompt format with a header, but without a specific signature to discrimate it from headerless formats. Vox Studio can only decompress indexed VOX files with one single prompt in it, or only the first prompt of a multi-prompt file.

VCE A-law: 80
VCE Mu-law: 81
VCE 32K: 82
VCE 24K: 83
VCE 16K: 84
VOX single prompt (Europe): 85
VOX single prompt (USA): 86
G726 32K Wave: 90

Voicetek: This is a format with a header containing all necessary information for decompression, but without a specific signature.

A-law: 100
Mu-law: 100
32K ADPCM: 100
24K ADPCM: 100

Micrologˑ: This is a format with a header containing all necessary information for decompression, but without a specific signature.

A-law: 120
Mu-law: 120
32K ADPCM: 120
24K ADPCM: 120

Philips: This is a format with a header containing all necessary information for decompression, but without a specific signature.

A-law: 140
Mu-law: 140
32K ADPCM: 140
24K ADPCM: 140

Elan Informatique: This is a headerless format.

32K ADPCM: 160

SCII: This is a headerless format.

32K ADPCM: 180

Sound Designer 2: This is a headerless format.
16 bit linear: 200

Rockwell: This is a headerless format. Vox Studio can only decompress files wich do not contain additional silence compression code sequences.
4 bits: 220
3 bits: 221
2 bits: 222

PhoneBlaster: This is a format with a header containing all necessary information for decompression. Vox Studio can only decompress files which do not contain additional silence compression code sequences.
4 bits: 240
3 bits: 240
2 bits: 240

Group 2000: This is a format with a header containing all necessary information for decompression, but without a specific signature.
A-law: 260
Mu-law: 260
32K ADPCM: 260
24K ADPCM: 260

Nortel: This is a format with a header containing all necessary information for decompression, but without a specific signature.
A-law: 280
Mu-law: 280
32K ADPCM: 280
24K ADPCM: 280

IBM: Only the DirecTalk format is known to Vox Studio. This is a headerless format.
DirecTalk: 300

Intervoice: This is a format with a header containing all necessary information for decompression.
Mu-law 64K PCM: 320
Mu-law 32K APCM: 320
Mu-law 24K APCM: 320
Mu-law 16K APCM: 320
A-law 64K PCM: 320
A-law 32K APCM: 320
A-law 24K APCM: 320
A-law 16K APCM: 320

Bicom: This is a headerless format.
    32K ADPCM: 340

NewVoice: This is a headerless format.
    CVSD: 360

OKI: This is a headerless format.
    32K ADPCM: 380

OKI Wave: This is a format with a header containing all necessary information for decompression.
    32K ADPCM: 381

Next/Sun: This is a format with a header containing all necessary information for decompression.
    8 bit linear: 400
    16 bit linear: 400
    A-law: 400
    Mu-law: 400

Centigram: This is a format with a header containing all necessary information for decompression.
    32K ADPCM: 420
    24K ADPCM: 420
    16K ADPCM: 420

<u>Output codes for the sound formats known to Vox Studio:</u>

These are the codes to be used to generate files of a particular sound format. For additional info concerning sound formats, see the input codes paragraph above.
Microsoft Wave:
    8 bit linear: 0
    16 bit linear: 1
    A-law: 2
    Mu-law: 3
    Dialogic 32K ADPCM: 4
    OKI 32K ADPCM: 5
    MS 4bit ADPCM: 6
    IMA 3 bit ADPCM: 7
    IMA 4 bit ADPCM: 8
    NMS G726 32K: 9

Raw PCM:
    8 bit linear: 20
    16 bit linear: 21

- A-law: 22
- Mu-law: 23

Dialogic:

- ADPCM: 40
- A-law: 41
- Mu-law: 42

Dialogic Wave:

- 32K ADPCM: 43
- A-law: 44
- Mu-law: 45

ITU-CCITT:

- G711 A-law: 60
- G711 Mu-law: 61
- G721 32K: 62
- G726 40K: 63
- G726 32K: 64
- G726 24K: 65
- G726 16K: 66

Natural MicroSystems:

- VCE A-law: 80
- VCE Mu-law: 81
- VCE 32K: 82
- VCE 24K: 83
- VCE 16K: 84
- VOX single prompt A-law (Europe): 85
- VOX single prompt Mu-law (USA): 86
- VOX single prompt 32K ADPCM (USA or Europe): 87
- VOX single prompt 24K ADPCM (USA or Europe): 88
- VOX single prompt 16K ADPCM (USA or Europe): 89
- G726 32K Wave: 90

Voicetek:

- A-law: 100
- Mu-law: 101
- 32K ADPCM: 102
- 24K ADPCM: 103

Micrölog:

- A-law: 120
- Mu-law: 121
- 32K ADPCM: 122
- 24K ADPCM: 123

Philips:
- A-law: 140
- Mu-law: 141
- 32K ADPCM: 142
- 24K ADPCM: 143

Elan Informatique:
- 32K ADPCM: 160

SCII:
- 32K ADPCM: 180

Sound Designer 2:
- 16 bit linear: 200

Rockwell:
- 4 bits: 220
- 3 bits: 221
- 2 bits: 222

Phone Blaster:
- 4 bits: 240
- 3 bits: 241
- 2 bits: 242

Group 2000:
- A-law: 260
- Mu-law: 261
- 32K ADPCM: 262
- 24K ADPCM: 263

Nortel:
- A-law: 280
- Mu-law: 281
- 32K ADPCM: 282
- 24K ADPCM: 283

IBM:
- DirecTalk: 300

Intervoice:
- Mu-law 64K PCM: 320
- Mu-law 32K APCM: 321
- Mu-law 24K APCM: 322
- Mu-law 16K APCM: 323

A-law 64K PCM: 324
A-law 32K APCM: 325
A-law 24K APCM: 326
A-law 16K APCM: 327

Bicom:
32K ADPCM: 340

NewVoice:
CVSD: 360

OKI:
32K ADPCM: 380

OKI Wave:
32K ADPCM: 381

Next/Sun:
8 bit linear: 400
16 bit linear: 401
A-law: 402
Mu-law: 403

Centigram:
32K ADPCM: 420
24K ADPCM: 421
16K ADPCM: 422

Sample rates commonly used for the sound formats described above:

Microsoft Wave:
Except for the Dialogic and OKI formats , any sampling frequency is allowed (but it is limited by the capabilities of your sound card when playing or recording). However, some preferred frequencies have been defined: 11025, 22050 and 44100 Hz.
Dialogic: 6000, 6053, 8000, 8117 Hz, and more recently 11025 Hz.
OKI: 8000 and 16000Hz

Raw PCM:
Any frequency is allowed (but it is limited by the capabilities of your sound card when playing or recording).

Dialogic:
6000, 6053, 8000 and 8117 Hz. More recently 11025 Hz has also been defined.

Dialogic Wave:

Same as Dialogic.

ITU-CCITT:
Only one frequency is allowed: 8000 Hz.

Natural MicroSystems:
Only one frequency is allowed: 8000 Hz.

Voicetek:
A-law, Mu-law and 32K ADPCM: 8000 Hz.
24K ADPCM: 6000 Hz.

Microlog:
A-law, Mu-law and 32K ADPCM: 8000 Hz.
24K ADPCM: 6000 Hz.

Philips:
A-law, Mu-law and 32K ADPCM: 8000 Hz.
24K ADPCM: 6000 Hz.

Elan Informatique:
Only one frequency is allowed: 8000 Hz.

SCII:
Only one frequency is allowed: 8000 Hz.

Sound Designer 2:
44100 and 48000 Hz

Rockwell:
Only one frequency is allowed: 7200 Hz.

Phone Blaster:
Only one frequency is allowed: 7200 Hz.

Group 2000:
A-law, Mu-law and 32K ADPCM: 8000 Hz.
24K ADPCM: 6000 Hz.

Nortel:
A-law, Mu-law and 32K ADPCM: 8000 Hz.
24K ADPCM: 6000 Hz.

IBM:
Only one frequency is allowed: 8000 Hz.

Intervoice:
Only one frequency is allowed: 8000 Hz.

Bicom:
Only one frequency is allowed: 8000 Hz.

NewVoice:
24000, 32000 and 64000 Hz

OKI:
8000 and 16000 Hz

OKI Wave:
8000 and 16000 Hz

Next/Sun:
8000, 22050 and 44100 Hz

Centigram:
Only one frequency is allowed: 8000 Hz.

Detection of conversion completion

A program launched by another program does not send back real completion information. There are, however, ways for the calling program to know if Vox Studio has completed it's conversion.

In C language, under Windows, the FindWindow function indicates if a given program is running, or at least if it is loaded in memory. The complete syntax for this command is as follows:

hwnd = FindWindow(lpszClassName,lpszProgramTitle);
where:
lpszCassName is a far pointer to a string giving the name of the program's registered class.
lpszProgramTitle is a far pointer to a string giving the name of the program as it appears in the title bar of it's window.
Any of these far pointers can take the NULL value, in which case the corresponding parameter is not relevant in the search.
The return value hwnd is the handle of the program if it has been found, or zero if the program is not running, or if there was an error in the search. So, by checking if the return value is zero or not the calling program can check if Vox Studio is running or not, provided it knows at least one of the class name or the title name of Vox Studio. The class name is "VoxStudioComLine" and the title name is "Vox Studio by Xentec" (without the quotes).

This procedure can detect quite easily if Vox Studio is running, and when to launch the next command line operation. If Vox Studio is running, it is busy, and one has to wait to send the next command. Vox Studio only runs one instance of itself (it will not convert several sound prompts at the same time). Although it detects the completion of a conversion, the above procedure can't tell the calling program if the conversion was successfull. This is what the /E flag on the command line is for.

The /E flag forces Vox Studio to write a tiny ASCII file, at the end of each conversion process, which holds the status of the conversion (see above for a full description of this flag). If the calling program asks for this file to be generated, it can then check for it's presence on the disk. It's existence means that the conversion process has ended. The calling program can then open the file, check if the conversion was successfull, then erase it and send the next conversion command, at the end of which Vox Studio will re-create it, and so on…

Finally, Vox Studio also returns the following error codes when called from an external program:

Vox Studio command line return codes:

Note: the error return codes from the following 3 groups can be OR-ed bitwise to error codes from another group. For instance error code 1036 is a combination of error codes 1024, 8 and 4.

0: Conversion succeeded
1: Vox Studio is already running
2: Uninstalled or corrupt version of vox Studio
3: Cannot open Vox Studio
4: Cannot open the log file. Continuing without logging
5: Cannot write to the log file. Closing and continuing

8: No input file name
16: Incorrect input file name
24: No output file name
32: Incorrect output file name
40: Number of channels must be 1 or 2 (mono or stereo)
48: Silence threshold must be between 1 and 99 inclusive
56: Normalize level must be between 0 and 100 inclusive
64: Input and output sound files would be identical. Stopping conversion
72: OPN: Cannot create output file
80: OPN: Unknown input sound file format
88: OPN: Cannot open input file
96: OPN: Cannot read from input file
104: OPN: Cannot write to output file

112: OPN: Not enough memory left
120: FLT: Cannot read from input file
128: FLT: Cannot write to output file
136: FLT: Cannot create output file
144: FLT: Cannot open input file
152: FLT: Not enough memory
160: FLT: Input file too short (less than 200 msec)
176: VOL: Cannot read from input file
184: VOL: Cannot write to output file
192: VOL: Cannot create output file
200: VOL: Cannot open input file
208: VOL: Not enough memory
216: CLA: Cannot read from input file
224: CLA: Cannot write to output file
232: CLA: Cannot create output file
240: CLA: Cannot open input file
248: CLA: Not enough memory
256: CTR: Cannot read from input file
264: CTR: Cannot write to output file
272: CTR: Cannot create output file
280: CTR: Cannot open input file
288: CTR: Not enough memory
296: SIL: No sound in file. Aborting
304: SIL: Cannot read from input file
312: SIL: Cannot write to output file
320: SIL: Cannot create output file
328: SIL: Cannot open input file
336: SIL: Not enough memory
344: RSP: Cannot read from input file
352: RSP: Cannot write to output file
360: RSP: Cannot create output file
368: RSP: Cannot open input file
376: RSP: Not enough memory
384: WRI: Cannot create output file
392: WRI: Cannot open input file
400: WRI: Cannot read from input file
408: WRI: Cannot write to output file
416: WRI: Not enough memory

512: Cannot create the end of conversion notification file
1024: Cannot write to the notification file
2048: Cannot close Vox Studio. Subsequent calls to Vox Studio will most probably fail

OPN: Opening and decompression of the input sound file procedure
FTL: Filtering of the sound procedure

VOL: Normalization of the sound procedure
CLA: Intelligibility filter procedure
CTR: Sound centering procedure
SIL: Leader/Trailer length adjustment procedure
RSP: Sampling frequency adjustment procedure
WRI: Compression and closing of the output sound file procedure

<u>Getting Vox Studio return codes</u>

The description below is for confirmed programmers only. Please familiarize yourself with the compiled versions VSTest16.exe or VSTest32.exe and their provided source before attempting to roll your own code.

In addition to writing a result file at the completion of a task, Vox Studio can also post a special message to the calling program to communicate the result of a conversion initiated by a command line set of parameters. In order to do this, a separate command line flag has been defined:

/H
Syntax: /Hxxxx
where xxxx is the ASCII translation of the HWND handle of the calling program.
Purpose: To enable Vox Studio to post a message to the calling program at the completion of a conversion task. This special message is VS_RETURNCODE and has a value of 40007. The wParam of the message sent to the calling program will contain the return code of Vox Studio. This special procedure has been introduced in order to get a more straightforward way of communicating the result (success or failure, and why) of a command-line-initiated Vox Studio conversion. It is intended for programmers, and has been introduced specifically to overcome the difficulty of retrieving program closure return codes in a 16 bits environment (Win 3.1). The error value returned is the same as the return code defined elsewhere in this documentation.

To illustrate the implementation of this special flag, we have developed for you a small program which retrieves the return code from Vox Studio, using the above flag. This program is “VSTest16.exe”, and it comes with source code in C language in order to assist you in implementing something similar in your own programs. This is a 16-bit program, compiled with MSVC ver. 1.52c. Here is how it works:

The program’s window contains one edit control to type the actual command line, one button labeled Go to send this command line to Vox Studio, and one static control to show the return code at conversion completion. The first thing the user has to do is to type a valid command line in the edit control. This command line must contain the full path and filename of Vox Studio, followed by the various required flags.
The user then presses the Go button, and VSTest16 retrieves the command line from the edit control. VSTest16 then appends the /H flag with the value of it’s

HWND handle to this command and sends it as a parameter of the WinExec Windows function. WinExec then launches Vox Studio, which interprets the various flags and takes appropriate action. When Vox Studio has finished it’s task, it uses the HWND handle value of the /H flag to post the message VS_RETURNCODE to the calling program, using the PostMessage Windows function. The wParam parameter of this message contains the code for the conversion result. Upon reception of the VS_RETURNCODE, VSTest16 writes the wParam value in the output static control of it’s own window. Here is the C code that actually starts Vox Studio in VSTest16 (this is an excerpt from the file vstest16.c):

```
// The user pressed the Go button, as a result WM_COMMAND message with
the wParam ID_GO is
// received by VSTest16.
case ID_GO:
// First, get the command line from the input edit control.
GetWindowText(hInputWnd,str,sizeof(str));
// Add to the end of this command line the special flag /hxxxx, where xxxx is the
HWND handle
// of the calling program (VSTest16 itself).
sprintf(str1," /h%d",(short)hwnd);
strcat(str,str1);
// Now, launch Vox Studio.
if (WinExec(str,SW_SHOW) > 31)
        {
// Success, Vox Studio is now running.
        strcpy(str,"Waiting for Vox Studio return code");
        SetWindowText(hOutputWnd,str);
        }
else
        {
// Something went wrong, probably an error in the command line just typed in the
input edit control.
// Notify the user of the problem through a message box.
MessageBox(hwnd,"Could not start Vox Studio.\r\nCheck the command
line.",
                "VSTest16",MB_OK);
        }
break;
// And now, VSTest16 simply waits for the VS_RETURNCODE message to be
received, or for the user to give it another try after correction of the command
line.
```

The code in VSTest16 to actually retrieve the return code from Vox Studio is as follows (see also vstest16.c):

```
// Vox Studio posted the VS_RETURNCODE message, and VSTest16 just got it.
case VS_RETURNCODE:
// wParam has the return code, just write it to the output static control.
sprintf(str,"%d",(short)wParam);
SetWindowText(hOutputWnd,str);
break;
```

Et voila, that’s all there is to it.

As is mentioned above, VSTest16 is a 16 bits (Win 3.1) program, but it will work equally well under Windows 95. Nevertheless, we also provide an additional 32 bits test program called **VSTest32** (and its source code in C), which will also retrieve the Vox Studio return codes. This program uses a different scheme to do this, using the 32 bits GetExitCodeProcess function. It does **not** use the /H flag. First there is the code to start Vox Studio (see VSTest32.c):

```
// The user pressed the Go button, as a result a WM_COMMAND message with
wParam == ID_GO is
// received by VSTest32.
case ID_GO:
// First, get the command line from the input edit control.
GetWindowText(hInputWnd,str,sizeof(str));
// Next, initialize the STARTUPINFO structure to be used by the CreateProcess
function. As we have no
// special requirements, the structure is initialized to all 0, except for the structure
size member.
memset(&StartupInfo,0,sizeof(STARTUPINFO));
StartupInfo.cb = sizeof(STARTUPINFO);
// Now, launch Vox Studio by a call to CreateProcess. This time, we do not
append the /H flag.
if
(CreateProcess(NULL,str,NULL,NULL,FALSE,0,NULL,NULL,&StartupInfo,&
ProcessInfo))
        {
// Launching Vox Studio succeeded. Save the handle of the Vox Studio process
for later use.
        hProcess = ProcessInfo.hProcess;
// In this program, we are going to test for the completion of the Vox Studio task
by regularly
// querying Windows 95 for the return code of the process. It is done through a
Timer procedure,
// which we start now.
        SetTimer(hwnd,1,200,NULL);
        }
else
        {
```

```
// Sorry, something went wrong and we could not start Vox Studio. It is probably
due to an error
// in the command line which was typed in the edit control. The complete
command line must be
// given, starting with the path and the filename of Vox Studio.
MessageBox(hwnd,"Could not start Vox Studio.\r\nCheck the syntax of the
command line.",
                "VSTest32",MB_OK);
        }
break;
```

Now for the code that retrieves the Vox Studio return code:

```
// To retrieve the return code, we regularly query Windows 95 for it.
// If Vox Studio is still running, the GetExitCodeProcess function returns
STILL_ACTIVE, otherwise it is
// the return code.
case WM_TIMER:
// Query for the return code
GetExitCodeProcess(hProcess,&dwCode);
if (dwCode == STILL_ACTIVE)
// Vox Studio has not finished yet
        SetWindowText(hOutputWnd,"Waiting...");
else
        {
// Vox Studio has completed, write the output code to the output static control,
and kill the timer.
        sprintf(str,"%ld",dwCode);
        SetWindowText(hOutputWnd,str);
        KillTimer(hwnd,1);
// Everything is now ready for a new command line to be typed and Vox Studio to
be restarted for a new
// conversion.
        }
break;
```

In this example, we used a timer to regularly query for the return code. Of course, your own program could query only if and when it needs to know the return code. For example, to know if it can send Vox Studio a new command line as Vox Studio accepts to run only one instance of itself. Because it is a 16 bits program, it cannot run two different tasks at the same time. The next release of Vox Studio will be a true 32-bit program, and will allow multiple simultaneous phone file conversions or manipulations.

Remember, this is the command-line documentation. There is also a conversion DLL that you can call from within your own applications. The DLL reference documentation is different. If you write a new program it would be wise to select the DLL instead. Happy programming.

The Xentec Vox Studio team.

Xentec nv-sa
De Helftwinning 2
3070 Kortenberg, Belgium.
Tel. +32 2 757 0666
Fax +32 2 757 0777
E-mail support@xentec.be
Web http://www.xentec.be