

```
*****
*****
*****

*****
*                                     *
*          SPEECH!  Advanced programming notes          *
*                                     *
*          (c) 1986 Superior Software Ltd.              *
*                                     *
*          written by:  R.C. Peterson                   *
*                                     *
*                                     *
*****
```

Introduction:

These notes will be useful if you want to use the SPEECH! routines in your own programs. I recommend that you study the Amstrad user guide and become familiar with Basic programming, the AMSDOS memory map and the use of resident system extensions before attempting to customise the program. An essential

familiar with Basic programming , the AMSDOS memory map and the use of resident system extensions before attempting to customise the program. An essential document for anyone interested in machine code programming is the Amstrad Firmware Specification manual (SOFT 158).

One of the first tasks you should undertake before attempting any alterations to the SPEECH! programs is to make a working copy of the files we supply (for your own use only !). Never make alterations to your original disc or cassette as it is possible to make mistakes if you do not fully understand what is going on. Feel free to look at the demonstration programs and adapt them for your own purposes.

Differences between Locomotive Basic V1.0 and V1.1

The CPC 464 computer is supplied with the Locomotive Basic interpreter V1.0 while the CPC 664 and CPC 6128 computers have Locomotive Basic V1.1. As far as is known all Basic programs written for V1.0 will run without alteration under V1.1. The reverse is not however true. This only effects SPEECH! in the following way: V1.1 allows the programmer to use strings directly in calls to resident system extensions (RSXs).

E.g.

E.g.

```
V1.0 - 100 a$="I think therefore I am"  
      110 òSAY,àa$
```

```
V1.1 - 100 òSAY,"I think therefore I am"
```

If you are writing programs which need to run under both Basic V1.0 and V1.1 then you must use the V1.0 form of the RSX Commands.

Converting a cassette version of SPEECH! to load from disc.

SPEECH! as supplied on cassette is configured to load and save files to and from cassette tape. If you wish to convert a cassette version to work with a disc system then the following notes may be of help. Only minor changes are required to make SPEECH! load from disc. After copying the programs and files to disc you will need to make the following changes:

Change line 90 in SPEECH.BAS to read:

60 @DISC

Change line 55 in NOTES.BAS to read:

55 @DISC

Similarly disc versions of the programs can be converted to operate from tape using the @TAPE command in place of the the @DISC commands.

Creating a version of SPEECH! to run at a fixed address.

The standard SPEECH! program loads into memory at the top of the block of memory reserved for the Basic system. The program requires 7880 bytes of memory and so the system variable HIMEM is reduced by this amount and the space available for Basic programs is reduced by the same amount. However the position of HIMEM is not fixed and depends on whether any resident system extensions (

language text into a phonetic form. This file can be edited using the dictionary editor EDIT.BAS (see later). The file SPCODE.BIN consists of three sections. These are 1) The English text to phonetics translator 2) The RSX operating system interface and 3) The phoneme pronouncer. This last routine can be used independently from either Basic or machine code programs.

The easiest way of incorporating the SPEECH! routines into a machine code program is to make a copy of the routine which has been relocated to the required address. To do this you must make the following temporary changes to the SPEECH program.

- 1) Initialise the machine so that no RSX routines are loaded and the maximum amount of memory is free. Do this by pressing the <escape> , <shift> and <control> keys simultaneously.
- 2) Load the SPEECH! loader program (SPEECH.BAS) .

- 2) Load the SPEECH! loader program (SPEECH.BAS) .
- 3) Choose the address at which you would like the routine to load and run at.
- 4) Set the variable relad in line 102 to the desired load address.
- 5) replace line 220 with the following line: 220 STOP
- 6) Now type RUN. SPEECH! will now be loaded and the routines relocated. The program will now stop and print the Break message.
- 7) Now save the block of code you want using the SAVE command.

E.g. SAVE "NSPCOD.BIN",B,loadad,len

The value of loadad will normally be 34740 . To save the whole routine len should be 7880 though this will be different if you have made any changes to the dictionary. To save only the SPEAK routine len should be 4493. If you have made any changes to the dictionary then then the value of len will be given by the expression lentab + length (where lentab and length are variables in SPEECH) .

You should now have a file on disc or cassette which can be loaded back into the machine at a fixed address and called from either Basic or

You should now have a file on disc or cassette which can be loaded back into the machine at a fixed address and called from either Basic or machine code. If you are loading the full routine you will need to introduce it to the AMSDOS operating system by CALLing the routine at an address = loadad + 4493 (where loadad is the routines load address). Here is a simple loader for a file that has been relocated to load at 32000.

```
10 start=32000: length= 7880: init = start + 4493
20 MEMORY start-1
30 LOAD "NSPCOD.BIN",start
40 CALL init ' initialise the routine
50 ùSAY," SPEECH! has now loaded "
:
:
9999 END
```

Using the SPEAK routine on its own.

If only the SPEAK routine is loaded then there is no need to initialise it. The SPEAK routine on its own cannot be called using the RSX commands ùSAY, ùPITCH, ùSPEAK etc. To use the SPEAK routine alone it is necessary for the main program to poke the required phonemes into an area of memory at the start of

0PITCH, 0SPEAK etc. To use the SPEAK routine alone it is necessary for the main program to poke the required phonemes into an area of memory at the start of the routine. This area occupies the first 255 bytes of SPEECH! The phonemes must be stored as upper case letters and digits and must be followed by a newline character chr\$(13). The routine is called at address start + 256 (where start is the address of the routine).

```
100 ' a mini version of SPEECH! has been loaded at address start
110 phonemes$ = "/HEHLL0W5 IY AE2M AY KOOMP0WTER"
130 FOR i = 1 TO LEN(phonemes$)
140 POKE i+start-1,ASC(MID$(phonemes$,i,1))
150 NEXT i
160 POKE i+start,13
170 CALL start + 256
:
:
:
```

Calling SPEECH! from machine code programs.

In this section the address start is the load address of the SPEECH! routine. The pitch of the sounds can be altered by poking a number corresponding to

In this section the address start is the load address of the SPEECH! routine. The pitch of the sounds can be altered by poking a number corresponding to the required pitch into the location start + 285 . This number must be between 3 and 255 , 3 being the highest pitch, 255 being a very low pitch.

The sound channel through which SPEECH! operates can be altered from its initial value i.e. through both channels, by poking one of the following values into the location start + 301 .

- 8 - sound through left channel only.
- 9 - sound through both channels.
- 10 - sound through right channel only.

If a copy of the full routine has been loaded and initialised then the full set of SPEECH! commands will be available from Basic. These commands can be called from machine code programs using a program similar to the following:

```
SAYCOM EQU start + 4725 ; define the start of the command
DESCRP DEFS 3           ; reserve space for string descriptor
STRPOS DEFM "/HEHLOW"   ; the string to pronounce
STRLEN EQU 8            ; the length of the string
PARLST DEFW 0           ; space for a parameter list

SPCALL LD HL,DESCRP     ; pointer to string descriptor
```

```

SPCALL LD HL,DESCRP      ; pointer to string descriptor
      LD A,STRLEN        ; length of string to be passed
      LD (HL),A          ;
      LD BC,STRPOS       ; get start of string
      INC HL             ;
      LD (HL),C          ; save address
      INC HL             ;
      LD (HL),B          ;
      LD HL,DESCRP       ; get address of descriptor
      LD IX,PARLST       ; pointer to parameter list
      LD (IX+0),L        ;
      LD (IX+1),H        ;
      LD A,1             ; only one parameter to be passed
      CALL SAYCOM        ; call the SAY routine
      .
      .
      .

```

The SPEAK routine can similarly be called using an identical routine to the one above except calling the routine at the address SPKCOM = start + 4684 . The pitch and channel can be set using the appropriate POKEs. It should be noted that the AF BC DE HL registers are not preserved by the SPEECH! routines and should be saved by the user's program. It should also be noted that all interrupts are disabled during the routine which actually

SPEECH! routines and should be saved by the user's program. It should also be noted that all interrupts are disabled during the routine which actually produces the sounds. The SPEECH! routine requires complete control over the sound chip and as a result any sounds in the sound queue or being played when the SPEECH! routines are called will be disrupted. If the sound chip is being used when SPEECH! is called then the sound queue should be flushed and the sound chip reset otherwise the state of the sound queue and sound chip will be indeterminate. The sound queues can be flushed from Basic using the following line:

```
SOUND 129,0 : SOUND 130,0 : SOUND 132,0
```

From machine code this can be achieved by the following call to the ROM:

```
CALL &BCA7 ; reset sound manager and clear all sound queues.
```

An alternative way of flushing the sound queues is by printing the the BEL character CHR\$(7) to the screen, this produces a "beep" sound.

The SPEECH! memory map.

```
0-----0 - start + 7880 = old HIMEM position
```

-----	- start + 7880 = old HIMEM position
SPDICN.BIN	
-----	- start + 5442
text translator	

RSX interface	
-----	- start + 4493 = initial entry point
Phoneme producer	
-----	- start = New HIMEM position + 1
Basic System	

the dictionary I have included the program EDIT which can be called from the main menu. EDIT is designed for editing the SPEECH! dictionary. EDIT allows you to play around with the dictionary adding new words and deleting ones you do not require. The program is a simple line-based editor written in Basic. The program has not been written to be foolproof but should trap most common errors.

Ideally you will have access to a floppy disc system for the storage of the data and program files. The cassette version is supplied so that it can be used with cassette files. Details of how to convert a cassette version to run from disc are included in the section of this help file dealing with converting SPEECH! for use with discs/cassettes.

To use the editor first load SPEECH! by typing RUN "SPEECH". When SPEECH! has loaded type RUN "EDIT" to load the editor. On loading, the program will ask you for the name of a dictionary file. For your first experiments you can use the basic dictionary we supply. The dictionary file will now be loaded in and

you are now in command mode and can add, delete or change any entry. Whenever the editor is in command mode it will display a prompt (" > ") at which point you can type a command.

All commands are entered as a single character followed by an optional string or number. One particularly useful command is the Help command, type the letter H followed by the return key when in command mode. This will give you a list of all the commands available.

The format of the dictionary.

When a line of English text is presented to the ÛSAY command it must be translated into a phonetic form before it can be pronounced. This process is carried out by a sub-routine referred to here as the translator. The translator scans through its input-buffer and produces a list of phoneme codes in the output-buffer.

In order to edit the dictionary successfully it is important to understand the format of each entry or line. Each line is laid out as follows:

çinput-text=output-textç

çinput-text=output-textç

The ç characters at the the start and end of the lines must always be present whenever a dictionary file is saved. The = sign separates the two substrings which comprise the dictionary entry. The general rule is that if the input-text is found in the input-buffer then the output-text is sent to the output-buffer. Thus the input-text always corresponds to a string of normal English characters and the output-text is always comprised of a list of phoneme codes. Entries in the table are stored in groups according to their first character. There are twenty-eight groups in the dictionary; for the letters A-Z, for space, and finally one for other miscellaneous characters. The entries in each group are ordered but not in an obvious way. The translator tries to match the buffer contents with all the entries in a group starting from the first and choosing the first which provides a match. The relevant number of characters in the input-buffer are then skipped over and the process is repeated until the end of the input-buffer is reached.

Two characters are treated in a non-standard way by the translator. They are the space character and the # (hash) character. The hash character can be used in the input-text of a dictionary entry to indicate a match with any character. If a hash character is found then the input-buffer character it matches is stored away. If the match is successful then the stored character can be used to take the place of another hash character supplied in the output-text.

can be used to take the place of another hash character supplied in the output-text.

If the first character of the input-text is a space then the entry will match with the start of a word. An input-text string with a space at the end will match with the end of a word. An input-text with spaces both at the start and at the end will only be matched by a whole word. In order to allow for other punctuation marks other than spaces at the end of words the translator will automatically match a space in the input-text with any character other than those in the range 0-9 and A-Z.

In order to translate correctly English sentences in which the words are separated by single spaces, the translator will match a single space with both the trailing space of the previous word and the leading space of the following word. As a result when a dictionary entry matches for a whole word or the end of a word you should not include a space at the end of the output-text.

If this seems complicated you can examine the standard dictionary which contains many examples. It is advisable not to make any changes to the main body of the dictionary as a single change could have complicated side-effects. The easiest ways to modify the dictionary are 1) by adding whole words and 2) by adding special characters.

Adding a new word to the dictionary.

Adding a new word to the dictionary.

- 1) First load and run EDIT as explained above.
- 2) Load in the dictionary you wish to work with.
- 3) First of all you want to find the words section.

Type: >F<space><return>

- 4) You will now see the following line displayed:

ç DAVID = DAYVIHDç

- 5) Now to insert a new entry (at the start of the words section).

Type: >I
new line :ç TOMB = TUWMç
ç TOMB = TUWMç
>

- 6) step 5) can now be repeated and more words added.

- 6) step 5) can now be repeated and more words added.
- 7) When all the words have been added use the S command to exit and save the new version of the dictionary.

Adding a special character to the dictionary.

The procedure for adding a new character is almost identical to that for adding a new word. First you need to find the special character section. You can do this using the following:

```
>Fç#  
ç#ç#ç  
>
```

Now you can add a special character. Any Ascii character can be added with the exception of the following: ç and = and # (hash) . For example you might wish to alter the % (percent) sign so that it is converted into the string of phonemes PERSEHNT . This can be done with the following line:

```
>I  
new line: ç%ç PERSEHNT ç
```

This can be repeated until all the characters have been entered. Then the S command can be used to exit and save the dictionary.

Saving the dictionary.

The dictionary can be saved using the S command. It is important not to overwrite your copy of the standard dictionary with an experimental version. If you are experimenting with changes to the dictionary it is advisable to make a working copy of the software and to use it. If the dictionary gets too badly corrupted it may become impossible for the editor to reload it. So never make changes to the disc or cassette we supply.

The s command takes around thirty seconds to pack the data ready to save it. When this is done a filename is requested and the dictionary is saved out to either disc or tape. When this has been done the length of the new version of the dictionary is displayed. This length should be noted as it must be incorporated in the Basic loader program (SPEECH.BAS) .

- 1) Change the filename in line 130 to that of the new dictionary file.
- 2) Change to value of length in line 60 to the new dictionary length.
- 3) save the program with a new name.

You now have a customised version of the SPEECH! loader program which will load SPEECH! together with your new dictionary. It is possible to add a large number of extra words to the dictionary. Each new word will reduce the space available to other programs and will slow down the translator program slightly. It is up to you to decide which words you will need for each application.

The editor will allow you to add as many entries to the dictionary as you like, subject to two constraints: The maximum number of entries is given by the variable tpmx in line 110, (in EDIT) and the entries when packed together to make up the stored dictionary must not exceed maxl bytes where maxl is a variable defined in line 105 (in EDIT). You can experiment with these values, increasing them to use up as much memory as you wish. One possible problem which

should be noted is that the SPEECH! routine must never be so long that the loader must reduce the top of the Basic system (HIMEM) below 16384 as this can lead to conflicts with the lower ROM.

EDIT command summary.

All commands begin with a single character which may be either upper or lower case. Some commands have parameters following the letter - either a number or a string. Other commands give special prompts when data is required.

- H - The H or Help command will display a brief summary of all the commands.
- X - The X or eXit command allows the user to leave the editor quickly without saving the dictionary file. It is possible to re-enter the editor using GOTO 1040 from Basic if the program has not been altered.
- S - The S or Save command will save the current dictionary file to either disc or cassette tape. The user is prompted for a filename. If the file has been saved successfully the length of the file in bytes and

disc or cassette tape. The user is prompted for a filename. If the file has been saved successfully the length of the file in bytes and the number of entries will be printed. The length must be noted if it is intended to use the dictionary with the standard loader program.

The following commands all refer to the current edit line i.e. the user's present position in the dictionary.

- T - The T or Top command moves the current edit line to the start of the dictionary.
- B - The B or Bottom command moves the current edit line to the end of the dictionary.
- U - The U or Up command allows the user to move the current edit line up a single line. If an optional number is included after the U, then the command line will move up by the specified number of lines or if this is impossible then to the top of the file.
- D - The D or Down command allows the user to move the current edit line down a single line. If an optional number is included after the D then the current edit line will be moved down by the specified number of lines or if this is impossible then to the end of the file.

then the current edit line will be moved down by the specified number of lines or if this is impossible then to the end of the file.

- L - The L or List command will list the current line. If it is followed by an optional number then it will list the specified number of dictionary lines from the current position or to the end of the file. Note that the current edit position is not changed by this command.
- I - The I or Insert command will insert a new line into the dictionary. The new line is always inserted before the current edit line. The inserted line becomes the new edit line. To use the command just enter I and press <return> ; the editor will prompt you for the line to insert.
- K - The K or Kill command will delete the current edit line.
- E - The E or Edit command allows you to edit the present edit line. Type E followed by <return>. The editor will now ask for two strings of characters. An attempt is made to find the first string in the present line and if it is present it is deleted and replaced with the second string. If the first string is not present in the line then no change is made.
- P - The P or Pronounce command enables you to hear the phoneme part of an entry that you have entered. If the SPEECH! RSX has not been loaded before EDIT is run then this command will not work as Basic will

entry that you have entered. If the SPEECH! RSX has not been loaded before EDIT is run then this command will not work as Basic will not recognise the commands. If you attempt to use the P command and this causes the editor to crash then you can re-enter the program from Basic using GOTO 1040. This command can also get confused if you use it on a line which is not correctly formatted.

- F - The F or Find command enables you to search through the dictionary from the current line for the string of characters which follows the F. e.g. F<space>GIVE<space><return> will search for the next occurrence of the word "give". If the string is not present then you will find that the current edit line is now the last line of the file. The ; and = characters which separate the two parts of the line can be used to find entries beginning or ending with a given string. When a line containing the string is found it becomes the new edit line.

