Typing Checker

For the Sinclair QL

Version 1 of QTYP

This is the first version of QTYP. It is designed to be used in the Pointer Environment used by QRAM. As it is designed to be used as a resident extension to the QL, it requires to be installed. A simple installation program is provided that will install it in the BOOT file of any recent version of QRAM. This is QTYP_ADD, described on page two of the manual. For earlier versions of QRAM (those without the BOOT_MAKE utility), and for some versions of QRAM supplied free with the Sandy SuperQBoard, the installation will need to be done by hand as follows.

If you only have one drive, you will first need to copy the QTYP and QTYP_SPELL files to RAM disk, otherwise put QTYP in drive 2. Then execute the QRAM HOT_MAKE utility and put your normal BOOT disk or cartridge in drive 1 (if it is not already there). Now answer HOT_MAKE questions as follows:

```
New Hotkey file (Y or N)> N                 add to existing
Hotkey filename> mdv1_hotkey                Hotkey file
Program file to add to Hotkey> mdv2_qtyp    or ram1_qtyp
Hotkey for this program> t                  ALT t is normal
Another program> N
```

Then copy QTYP_SPELL to your BOOT disk or cartridge. Pop up Qram to find the new size of the Hotkey file, and the size of QTYP_SPELL (5424), and load your BOOT program to edit it. Change the RESPR for the Hotkey file to the new size, and add a RESPR/LBYTES/CALL statement for the QTYP_SPELL file immediately before the RESPR for the Hotkey file. Save your BOOT program back to your BOOT disk or cartridge. There are a number of example BOOT files in the accompanying leaflet.

BEWARE: the manual gives the ALT keystroke for QTYP as "T", this should be "t" which is easier to type, but will not be available if CAPSLOCK has been set. The QTYP_ADD program installs a version of QTYP with Hotkey ALT t.

The dictionary has turned out 15% over target, so that it now occupies about 75 kbytes. Rebuilding a dictionary using DED should not normally be required. In this version it takes about twelve hours to rebuild our standard dictionary. We advise sending the rebuild log to a printer: you can then see if it has stopped altogether.

There are two facilities which have been added to the QTYP_SPELL extensions. If you SPELL_ADD a zero length string (or FS.SAVE a zero length string), QTYP will add the current word in its buffer to the temporary word list. The IO.SBYTE IO call returns a code (in D1.b) giving its interpretation of the character checked: -ve cursor or edit keystroke, 0 separator or odd character, +ve printable character.

The Dictionary Editor and the Pointer Interface

The cursor control in QTYP_DED is very slippery (not to say slimy) with earlier versions of the Pointer Interface. Recent versions of PTR_GEN (which replaces PTR_IMI and PTR_KBD) and WMAN are enclosed for those who wish to install them. SuperQboard mouse owners will need to install the file PTR_GEN and remove the POINTER command from their BOOT file. For general instructions on installing PTR and WMAN files you should read the QRAM manual and the accompanying leaflet on BOOT files.

You should normally use the most recent versions of the Pointer Interface file (PTR_GEN, PTR_IMI or PTR_KBD) and the Window Manager file (WMAN). To find the version number of a the Pointer Interface files, VIEW the files with QRAM.

If you do not feel strong enough to tackle the installation of new copies of PTR_GEN and WMAN, it is better to use ALT UP and ALT DOWN when editing a dictionary, rather than the UP and DOWN keys on their own.

CONTENTS

The QTYP Package

The QTYP package provides several levels of access to a comm spelling checking utility. For instant use the QTYP program provid real-time checking of words as they are typed. At lower levels t spelling utility can be accessed by programs to check spelling charact by character or a word at a time. As the spelling utility uses the standa QL IO interface, it can be accessed directly from SuperBASIC or from a compiled language. This means that it is now possible for the sa spelling checker (and dictionary) to be used by programs from any vendor.

Checking spelling from within a program is much more satisfacto than using QTYP, as the program will always know the context of a keystroke, whereas QTYP needs to guess. While we have tried to make QT fairly intelligent, it will sometimes guess incorrectly, But until t authors of text editors for the QL have had a chance to incorporate t necessary code into their programs, you will have to make do with QTYP.

We must acknowledge that QTYP would not have come into existence Sector Software had not produced SPELLBOUND. This product establish that real time spelling checking was not only practical, but useful as we QTYP was produced for two main reasons. The first is that SPELLBOU had to do some rather clever things to produce pulldown windows on Q which did not have the extended console driver of the QL Point Interface. This made it incompatible with the QL Pointer Interface whi can get somewhat confused if programs alter the system variables access the display directly rather than using the QDOS calls provided f the purpose. The second reason is that SPELLBOUND provided no means f other software suppliers to access the dictionary, and such a clos system runs contrary to QJUMP's commitment to promoting open syste and co-operation between software vendors.

QTYP Files

The QTYP cartridge (or disk) has a number of files with both complet and kit parts of the spelling checker.

QTYP is the real time spelling checker.

QTYP_DICTIONARY is the QTYP dictionary.

QTYP_ADD adds a pre-configured version of QTYP to a QRAM ty BOOT_REXT file.

QTYP_CONFIG configures QTYP to set preferences for checking, a dictionary filenames and devices.

QTYP_SPELL is the file with both the SuperBASIC and machine co spelling checker extensions.

QTYP_DED is the QTYP dictionary editor.

QTYP_DED_HELP is the help file for the QTYP dictionary editor.

QTYP_FILE is used to check the spelling in plain text or _DOC files.

Before using QTYP you will need to install the QTYP program and t QTYP_SPELL spelling checker into your normal BOOT file. This can eith be done by using the simple QTYP_ADD program, or you may put your o BOOT file together as you wish.

Installing QTYP

QTYP is designed as a resident extension to the QL. Before using it, it is necessary to install it into your BOOT program.

The program QTYP_ADD adds a preconfigured version of QTYP to a BOOT_REXT file such as is provided with QRAM or produced by the BOOT_MAKE facility of QRAM. It also updates the BOOT program to increase the space allocated in the resident procedure area and adds the command QTYP to the end of the BOOT program to activate the QTYP Hotkey. To run this program, have a Microdrive or disk with your normal BOOT and BOOT_REXT files handy, ensure that your QL is BOOTed up with the pointer interface from QRAM, put the QTYP Microdrive or disk into a drive and execute QTYP_ADD.

```
    EXEC MDV1_QTYP_ADD          Microdrive in drive 1
or  EXEC MDV2_QTYP_ADD          Microdrive in drive 2
or  EXEC FLP1_QTYP_ADD          floppy in drive 1
or  EX QTYP_ADD                 for those with Toolkit II
or  use QRAM FILES menu to EXECUTE QTYP
```

Then follow the instructions. When QTYP has been added to BOOT files, it will be available the next time the QL is reset. It is invoked by the key combination ALT T. Beware of the difference between ALT T to invoke QTYP and CTRL T which pulls down the QTYP control menu after it has been invoked.

For various reasons, it is more efficient to add the QTYP program to the HOTKEY file and include the QTYP_SPELL extensions in your BOOT file. Once you have gained a feel for QTYP, you can configure the original files to set the default drive and dictionary filename (in QTYP_SPELL) and the control options (in QTYP). These can both be set by the configuration program.

```
EXEC MDV1_QTYP_CONFIG       etc.
```

When you have done this you should execute the HOT_MAKE program to add QTYP to the HOTKEY file and then remake the BOOT files using BOOT_MAKE. The BOOT files should include the Pointer Interface file PTR_IMI (or the POINTER command), the Window Manager WMAN and the QTYP_SPELL files before the HOTKEY file. The QTYP command which is added to the BOOT file by the QTYP_ADD program is not required.

The configured defaults and options will be available after the QL has been re-booted.

QTYP_CONFIG can also be used to reconfigure the BOOT_REXT and HOTKEY files directly. Note that the configured options will only become available if the actual file loaded by your BOOT program (usually BOOT_REXT) has been configured or includes a configured file and the QL been re-booted.

The QTYP Program

QTYP is the real time spelling checker for the QL Pointer Interface. It provides spelling checking for any text editing program which uses standard edit control keys. It is suitable for use with Quill, The Editor, ED and other QL programs and it will work reasonably well with Front Page although this has non-standard edit keys.

QTYP requires the QL Pointer Interface and Window Manager as well as the QTYP_SPELL extensions to be loaded into the QL before it is invoked.

Although QTYP is a normal executable program and can be EXECed (or EXed), QTYP will normally be invoked using a hotkey. Each invokation of QTYP will monitor the current input console of the job that was at the top of the "pile" of jobs visible on the QL's screen when QTYP was invoked. Each QTYP job is owned by the job being monitored: when that job is removed, its copy of QTYP will also be removed.

There can be many copies of QTYP in the QL at one time, each monitoring a different console. In the QRAM JOBS menu, the QTYP jobs will be shown as "daughters" of the jobs being monitored.

User Interaction with QTYP

QTYP communicates with the user through a variety of BEEPs and pull-down menus. QTYP will try to pull down a menu over the current cursor position but as cursors tend to be disabled and moved around, it may not always succeed.

The main control menu is normally pulled down by typing "CTRL T". If this keystroke upsets your programs, you can change it using the control menu itself.

Within the control menu, you can select or deselect items by either pointing to them and hitting them, or by pressing the first letter (or number!) of the item. Selected items are shown highlighted in the menu.

The ACTIVE and INACTIVE items control the state of QTYP. If inactive, QTYP does not do any spelling checking, but does check for the control menu key. The ACTIVE or INACTIVE state is selected by pressing "A" or "I" or hitting the ACTIVE or INACTIVE items.

The BEEP length can be set to short (press "1" or hit the 1), normal (2) or long (3). While QTYP is active, it can beep every time a character is CHECKed, only when a QTYP is WARNING you of a (potential) spelling error or not at all. The warning beep is a much longer and lower note than the cheery little "OK so far" beep. QTYP will not beep at all if checking has been temporarily suppressed by unrecognised keystrokes.

Alternatively (or additionally) QTYP will PULL DOWN a warning window whenever it has a complete word that it cannot find in the dictionary. Pressing "P" or hitting the PULL DOWN item will select or deselect this option. When a warning window is pulled down, you can ignore the warning by pressing "ESC" (or hitting the ESC item) or save the word in the temporary word list by moving the QTYP pointer to the SAVE item and pressing SPACE, ENTER or either button on the mouse.

You can also SAVE the current word from within the control menu by pressing "S" or hitting the SAVE item. If you have the pull down window facility disabled, then this is the only way of saving words using QTYP.

The control menu can also be used as a normal dictionary by LISTing all the words that start with the current (part) word. Hitting the list itself, will list another page of words as will hitting the MORE item, or pressing "M". Hitting the AGAIN item, or pressing "A", will start the list again. If you have no current (part) word, then LIST will list all of the temporary word list. Press "ESC", or hit the ESC item to leave the LIST menu.

Finally you can set the control menu KEY by hitting the KEY item, or pressing "K" and then pressing any letter.

When you have finished with the control menu, leave it by pressing "ESC" or hitting the ESC item.

Dictionary Notes and QTYP Failures

When a word is encountered that is qualified by a note in the dictionary, a window is pulled down with an appropriate message. This facility can only be removed by removing the notes from the dictionary or by editing the dictionary.

It is possible that QTYP may abort. In this case it will produce a low pitch rude noise, and, if possible, an explanatory pull down window. The most likely cause of failure is the absence of the QTYP dictionary from the default drive.

QTYP Dictionary and Word List

When QTYP is first invoked, it will check for the presence of the dictionary in memory, and, if necessary, will load the dictionary from disk or Microdrive. During use, you can add words to a temporary word list. Dictionaries occupy a considerable amount of memory, and the space may be reclaimed using the SuperBASIC functions SPELL_CLEAR to clear out the memory, or SPELL_SAVE to save the temporary word list to a file before clearing out the memory.

A list may be loaded back in using the function SPELL_FILE. Alternatively, a word list in a file can be merged into your dictionary using the dictionary editor QTYP_DED.

A Warning

Current versions of the QL Pointer Interface do not notify a program that its window has been buried. This means that you must ESCape from a QTYP pull down window rather than burying it. If you do bury QTYP's window under some other job's window, it will just sit there waiting.

Character by Character Checking

The SPELL extensions provide two separate mechanisms for spelli checking. The mechanism used by QTYP is character by charact checking. This has to allow for cursor movements and character delet as well as normal letters. The current (part of a) word being checked held in an internal buffer. No more than 128 characters should be put in this buffer.

When it is checking character by character any letters are added an internal buffer; cursor left, cursor right, delete left and delete rig are used to edit the internal buffer. QTYP assumes that text editing is "insert" mode rather than "overwrite" mode. If at any time the characte in the buffer, to the left of the cursor, do not form the start of a wor then SPELL will notify an error. When a separator (space, ENTER, carria return, null or any printable non-letter) completes the check, the wo must be complete. Any other character stops the checking process unt SPELL is reset by a separator.

This means that character by character checking can be used check words as they are typed. To a certain extent, it can tolerate t user going back to correct errors in the word he has just typed. suppressing the checking when strange characters (such as cursor up down) are encountered, spurious checking errors are avoided when text being altered by moving around a document. As normal typing is resume the checking is automatically re-enabled.

Word Checking

When the SPELL extensions are checking complete words, a word regarded as a group of letters separated from other groups of letters anything that is not a letter.

Valid letters

SPELL has a regular, but unusual, letter order which includes all the QL's letter set. The case of a letter is ignored. SPELL dictionarie are stored in this order.

'aäáàâãbcçdeëéèêfghiïíìîjklmnñoöóòôõpqrsßtuüúùûvwxyzæøå

Because of the peculiarities of the English language, apostrophe is treated as a "pseudo-letter". The apostrophe is often used to "quote text, where it acts as a punctuation mark. It is also used as a possessiv or as an abbreviation symbol (it's two o'clock and the cup's handle broken).

If an apostrophe is found at the beginning or end of a word, it i ignored. If an apostrophe is found within a word, then if the word end with "'s" then the "'s" is ignored, otherwise the word must appear in th dictionary including the apostrophe.

Hyphens are treated as separators and the words before and afte the hyphen must both exist in the dictionary.

The QTYP English Dictionary

The QTYP dictionary has been produced specially for QTYP. A dictionary for spelling checking has rather different requirements from those of ordinary use. An ordinary dictionary does not need to spell out all the many forms of verbs, or the plurals of nouns. For checking spelling, it is necessary to do this. The aim of the QTYP dictionary has been regularity and minimisation. As a dictionary size increases, the chances of any additional word being of use decreases rapidly, but the chances of any additional word being a potential misspelling of another word increases. Therefore, greater size does not necessarily mean better. We have decided on a dictionary size of about 40,000 words which occupies about 60 kilobytes of memory.

As there is no point in having alternative spellings for words we have had to standardise (rather than standardize) on what appears to be common British English. This has *colour* rather than *color*, *levelled* rather than *leveled* and *-ise* rather than *-ize*. You can use the dictionary editor to change these. In some cases alternative spellings are difficult to avoid: the British English printers' *fount* is the same as the *fount* of all knowledge, while the American English printers' *font* is indistinguishable from a *font* for Christening people.

We have tried to avoid specialist words as these are of little use outside the specialist area, and within the specialist area far more words would be required than you would ever find in a standard or "concise" dictionary. Instead, we have provided the QTYPE_FILE utility and the SPELL_FILE command which you may use to extract your own specialist dictionary from documents or letters you have already written.

The dictionary contains a few notes. These are mainly to do with the spellings of related words which sound similar but have different meanings or usages. E.g. *practice* and *practise*, *therefor* and *therefore*.

Extending the Dictionary

QTYP does not include a facility for updating the dictionary. It can, however, save words in a temporary word list which can then be written to a file for future use, or for incorporating into your dictionary using the dictionary editor.

This approach may seem more cumbersome than the instant update option of some spelling checkers, but in the long term it helps to preserve the integrity and regularity of the dictionary. If a rock climber wishes to add "rappelling", then the dictionary editor makes it easy to ensure that "rappel", "rappels" and "rappelled" are all added at the same time.

Even more important is the facility to remove words which are frequently typed in error for other words (putting up with the occasional false warning in order to catch the common error).

The Dictionary Editor

QTYP_DED is an editor for QTYP dictionaries. With DED you can set up a new dictionary, edit an existing dictionary or rebuild an existing dictionary.

For reasons of efficiency, DED edits a dictionary in memory. This can use a considerable amount of memory, so it is advisable to clear the memory as much as possible by using the SPELL_CLEAR (or SPELL_SAVE) function.

When DED prompts for a filename, you can either retype the name or edit the default provided.

Editing a Dictionary

There are two ways of extending a dictionary. The first is to edit the dictionary directly. The second is to load words into a "secondary" word list which will hold just the new words you wish to add. Then edit in any additional forms of the words (like plurals of nouns, other tenses of verbs, negatives etc.). Finally, merge the secondary word list into the main dictionary.

DED will display a section of the dictionary, with the words in order. It can only display words starting with the same letter. To alter a word just delete any unwanted characters using the normal delete keys, and type any new characters you wish. To add a word, position the cursor on a similar word and press ENTER or SHIFT ENTER (there is a slight difference) and edit the resulting copy. Changes to a word are not saved until the cursor is moved off the line or a command is invoked. Until the changes are saved, the word may be recovered by pressing ESC.

If editing a word puts it out of order, then the display will be adjusted to move it back into order. If the start of the word is changed (e.g. by putting "un" in front), the word will disappear and be put in the appropriate place. To remove a word, delete all the characters.

Edit Keystrokes

The behaviour of the cursor will depend on the version of the QL Pointer Interface which is resident in your QL. Later versions have better cursor handling. Otherwise DED uses the standard keystrokes:

F1	HELP
F3	Invoke commands menu
CTRL F3	Change window size
CTRL F4	Move window
LEFT / RIGHT / UP / DOWN	Move cursor
SHIFT LEFT / SHIFT RIGHT	Move cursor by word
SHIFT UP / SHIFT DOWN	Move cursor to top or bottom
CTRL LEFT / CTRL RIGHT	Delete Characters
ALT UP / ALT DOWN	Scroll window
SHIFT ALT UP / SHIFT ALT DOWN	Scroll window by page

Notes on Words

Notes may be added to words by just typing them after the word, separated by a space. Notes should not be longer than 40 characters and preferably shorter.

Dictionary Editor Commands

The commands are invoked by hitting the CMD item or pressing F3. The commands available depend on whether you are editing a secondary word list or the main dictionary.

FIND is always available. With this you can start editing anywhere in the dictionary. If you are editing the main dictionary, you may LOAD a secondary word list. If, on the other hand, you are editing a secondary word list, you can MERGE this into the main dictionary.

If you are editing the main dictionary, you can SAVE the dictionary back to disk or Microdrive. When you do this there are two options. The first is to save the dictionary uncompressed, this is faster but takes about twice as much space on the disk or Microdrive. The second is to save it compressed which may take several minutes.

Rebuilding a Dictionary

The QTYP dictionary format incorporates two methods of reducing the size of the dictionary. The first is a form of differential encoding that stores only the difference between two successive words. Because of the nature of a dictionary, the difference between two successive words is usually quite small. Typically, this reduces the space taken by the dictionary to about three bytes per word. The second stage is a compression process that encodes several bytes into one. The benefits of this second stage depend on the frequency with which certain sequences of bytes (e.g. "tion") occur. This is language dependent.

When a new dictionary is created, an existing dictionary is used as a model to predefine the compression tables. If these compression tables are not appropriate for the language or for the particular words that occur in a dictionary, then the compression will be inefficient. To correct this the dictionary may be rebuilt. Rebuilding involves a complete analysis of the structure of the dictionary, followed by re-compression. This is a very long process.

QTYP_FILE

QTYP_FILE is used to check the spellings in one or more files. It can process either plain text or Quill document files. As each misspelling is found, the user is asked whether he wishes to ignore it (press "I"), mark it with a "<" (press "M") or add it to the temporary dictionary (press "A"). Pressing ESC will stop the checking.

Where a misspelling is marked in a file the "<" overwrites the character after the offending word. This avoids problems with adjusting the sizes of the files.

SuperBASIC Interface to QTYP_SPELL

The SuperBASIC interface to the SPELL device is provided to enab simple spelling checking from interpreted or compiled SuperBAS programs. The interface is consists mostly of functions which retu error codes. In addition there are three functions which return strings.

Before using the dictionary, it is necessary to open a channel to i There are two functions provided to open a SPELL channel. Each of thes has two optional parameters. The first parameter is the channel numbe If this is not given, then the function will find a vacant hole in th SuperBASIC channel table and if the open call is successful, will retu the channel number. The second parameter is the name of a file. This nee not usually include the drive name as the SPELL open will use th SuperToolkit II PROGRAM default device if possible, or the default devic built into the SPELL device.

In order to minimise system overheads, the SPELL device does no load its standard dictionary until requested. It is possible that a application may need to do this itself. The recommended procedure is fo a program to try to open the channel using SPELL_OPEN. If this fails wit ERR.NI then the dictionary has not been loaded. If this happens, the use should be asked to ensure that his dictionary cartridge or disk is in th appropriate drive and then the program should try to open the channe using SPELL_NEW. If this fails the application can either give up, o request that the user specifies a filename, and puts the correct medium i the correct drive.

When a SPELL channel has been opened, the spelling of words may b checked character by character, or a whole word at once. It is als possible to add words to the temporary word list, and to extract lists o words from the dictionary. When you have finished with it, just CLOSE th SPELL channel. Note that if you try to use the SPELL functions with channel other than a SPELL channel, you may get some bizarre error code returned from the functions.

SuperBASIC Functions

SPELL_NEW	open channel to new dictionary
SPELL_OPEN	open channel to dictionary
SPELL_CLEAR	clear out dictionary and temp word list
SPELL_CHECK	check word or words against dictionary
SPELL_CHAR	check character by character
SPELL_ADD	add word to temporary word list
SPELL_SAVE	save temporary word list to file
SPELL_FILE	check / load words from file
SPELL_NOTE$	returns note as a string
SPELL_BUFF$	returns contents of SPELL buffer
SPELL_WORD$	gets word from dictionary

Note that these are all functions and return values. If you wish t type them as commands, then you should PRINT or (much better if you hav Toolkit II) REPORT the result:

```
    PRINT SPELL_CLEAR      to clear out the dictionary
or  REPORT SPELL_CLEAR
```

SPELL_NEW

SPELL_NEW sets up a new dictionary within the spelling checker. Normally this function will only need to be called once at the start of a session. If no file name is given, the default dictionary name built into the SPELL device (usually "QTYP_DICTIONARY") will be used. A channel number may be specified, or SPELL_NEW can allocate its own.

```
channel = SPELL_NEW
error = SPELL_NEW (#channel)
channel = SPELL_NEW (name)
error = SPELL_NEW (#channel, name)
```

Successful calls will return either 0 or the channel number. For both SPELL_NEW and SPELL_OPEN, only the file name need be given, the device will be assumed to be WIN1_, FLP1_ or MDV1_ as appropriate. The following error codes can be expected:

ERR.BN	-12	"name" not found or it is not a dictionary
ERR.OM	-3	out of memory
ERR.IU	-9	dictionary file is in use
ERR.EX	-8	standard dictionary already set
ERR.NF	-7	SPELL device not present
ERR.NO	-6	channel table full

SPELL_OPEN

SPELL_OPEN is the normal function for opening a SPELLing checking channel. If a name is given, this is the name of a specialist dictionary. The words in this specialist dictionary are not accessible to other channels. If a SPELL channel is already open with a specialist dictionary of the same name, then the new channel will use that same dictionary. When all the channels using a particular specialist dictionary have been closed, the specialist dictionary is thrown away.

```
channel = SPELL_OPEN
error = SPELL_OPEN (#channel)
channel = SPELL_OPEN (name)
error = SPELL_OPEN (#channel, name)
```

Successful calls will return either 0 or the channel number. For both SPELL_NEW and SPELL_OPEN, only the file name need be given, the device will be assumed to be WIN1_, FLP1_ or MDV1_ as appropriate. The following error codes can be expected:

ERR.BN	-12	"name" not found or it is not a dictionary
ERR.OM	-3	out of memory
ERR.IU	-9	dictionary file is in use
ERR.NI	-19	standard dictionary is not set
ERR.NF	-7	SPELL device not present
ERR.NO	-6	channel table full

SPELL_CLEAR and SPELL_SAVE

SPELL_CLEAR removes the current standard dictionary and temporary word list. SPELL_SAVE does the same, but saves the word list to file first.

```
error = SPELL_CLEAR
error = SPELL_SAVE (name)
```

Successful calls will return 0. The following error codes can be expected:

ERR.OM	-3	out of memory
ERR.IU	-9	the standard dictionary is in use or the file is in use (SPELL_SAVE)
ERR.NF	-7	SPELL device not present
ERR.NO	-6	channel table full
ERR.DF	-11	Drive full (SPELL_SAVE)

SPELL_CHECK, SPELL_NOTE$ and SPELL_BUFF$

The SPELL check function checks a complete word held in a string. There are three forms of this function. If it is just called with a string, then it will check the first word it can find in the string. If however, it is called with an additional pointer, it will look for each word in the string, starting at the character of the string which is pointed to. On return, the pointer is updated to point to the character after the word just checked. If there are no erroneous words in the string, the pointer will be returned equal to the string length plus 1.

```
error = SPELL_CHECK (#channel, string)
error = SPELL_CHECK (#channel, string, pointer)
error = SPELL_CHECK (#channel, string, start, end)
```

There are four error codes that can be expected to be returned from these calls:

0		no error, the word in the buffer is valid
ERR.NO	-6	channel not open!!!
ERR.NF	-7	not found, the word in the buffer is not found
special		there is a note attached to this word

Special error messages are large negative numbers. A special error message may be converted to a string by the function SPELL_NOTE$, and the function SPELL_BUFF$ will return the current word in the SPELL internal buffer.

```
note$ = SPELL_NOTE$ (error)
word$ = SPELL_BUFF$ (#channel)
```

The second and third forms of SPELL_CHECK make it relatively easy to check multiple word strings. For example, the following routines check all the words in the string "words$". The difference is that using the third form, you can easily find the start of the offending word to change or mark it. In both cases the pointer (or end pointer) is used to indicate where the check should start, and, on return, where the check has reached. The pointer must be preset by the program, but thereafter it is maintained by SPELL_CHECK.

SPELL_CHECK Examples

This first example uses the second form of SPELL_CHECK. It scans the whole of a string (words$) writing out an error for each misspelt word, and any notes associated with any of the words found.

```
word_ptr = 1
REPeat check_word
  sp_err = SPELL_CHECK (#sp_chan, words$, word_ptr)
  SELect ON sp_err
    = 0: EXIT check_word
    =-7: PRINT SPELL_BUFF$ (#sp_chan); '...is bad'
    =-6: PRINT 'SPELL channel not open': EXIT check_word
    = REMAINDER
      PRINT SPELL_BUFF$ (#sp_chan)! SPELL_NOTE$ (sp_err)
  END SELect
END REPeat check_word
```

This second example does the same operations as the first, but uses the third form of the SPELL_CHECK function.

```
w_end = 1
REPeat check_word
  sp_err = SPELL_CHECK (#sp_chan, words$, w_start, w_end)
  SELect ON sp_err
    = 0: EXIT check_word
    =-7: PRINT words$ (w_start TO w_end); '...is bad'
    =-6: PRINT 'SPELL channel not open': EXIT check_word
    = REMAINDER
      PRINT SPELL_BUFF$ (#sp_chan)! SPELL_NOTE$ (sp_err)
  END SELect
END REPeat check_word
```

SPELL_CHAR

SPELL_CHAR checks one character at a time. The spelling check is reset by a separator (space, newline, carriage return, null or any printable non-letter). Non-zero error codes will only be produced when the first separator is sent after a word. At this point cursor left or delete left will recover the internal buffer contents for further modification. Any letters are added to an internal buffer; cursor left, cursor right, delete left and delete right are used to edit the internal buffer. SPELL only checks the characters to the left of the cursor. Any other character stops the checking process until SPELL is reset by a separator.

error = SPELL_CHAR (*#channel, character*)

The character should be a string or substring or the internal CODE value of a character (0 to 255).

There are six error codes that to be expected from SPELL_CHAR

	1	character not checked
	0	no error, the word in the buffer is valid
ERR.NC	-1	not complete, the word is incomplete
ERR.NO	-6	channel not open!!!
ERR.NF	-7	not found, the word in the buffer is not found
special		there is a note attached to this word

If SPELL_CHAR is called with a letter, cursor left, cursor right or delete character, then it will return the error code ERR.NC if the characters in the buffer to the left of the cursor form the start of a word in the dictionary, but are not, in themselves, a complete word. If SPELL_CHAR is called with a separator, the error code ERR.NF will be returned if the word is either incorrect or incomplete.

This can be used for spelling checking "on the fly". The following function checks input to a SuperBASIC program. To use it, INCHK$(channel) should be used in place of INKEY$(#channel,-1).

```
DEFine FuNction inchk$ (in_chan)
  LOCAL char$, sp_err
  char$ = INKEY$ (#in_chan, -1)
  sp_err = SPELL_CHAR (#sp_chan, char$)
  SELect ON sp_err
    = -7: BEEP 5000,100
    = -1: BEEP 2000,20
    = 0: BEEP 500,3
  END SELect
  RETURN char$
END DEFine inchk$
```

SPELL_ADD

Words may be added to the temporary word list with the SPELL_ADD function. Normally, the only error codes returned from this function are fatal operating system errors.

error = SPELL_ADD (*#channel, word*)

0		no error
ERR.OM	-3	out of memory
ERR.NO	-6	channel not open!!!

SPELL_WORD$

SPELL_WORD$ is used to find all the words beginning with the letters in the internal SPELL buffer. Each call will get one more word, the words will be in alphabetical order and will come from all three dictionaries. When there are no more words to be read, SPELL_WORD$ returns an empty string. The next call will start from the beginning of the list again.

word$* = SPELL_WORD$ (#channel*)

There is a slight variation on this call which, if the internal buffer is empty, will read all the words from the temporary word list only. (The buffer may be emptied using the SPELL_CHECK function to reset SPELL, as shown in the following example. The error code should be ignored.)

```
sp_err = SPELL_CHECK (#channel,' ')
REPeat words
  word$ = SPELL_WORD$ (#channel)
  ....
```

To illustrate the use of SPELL_WORD$, the INCHK$ function (listed above), is extended to provide a list of possible words when the user types "?". It stops listing the words when the user presses a key.

```
DEFine FuNction inchk$ (in_chan, list_chan)
  LOCAL char, char$, word, word$, sp_err

  REPeat char
    char$ = INKEY$ (#in_chan, -1)
    IF char$<>"?": EXIT char
    REPeat word
      word$ = SPELL_WORD$ (#sp_chan)
      IF word$='' OR INKEY$ (#in_chan,10)<>'': EXIT word
      PRINT #list_chan, word$
    END REPeat word
  END REPeat char

  sp_err = SPELL_CHAR (#sp_chan, char$)
  IF sp_err = -7: BEEP 5000,100
  RETURN char$
END DEFine inchk$
```

SPELL_FILE

SPELL_FILE is used to check the spellings in a file. It can process either plain text or Quill document files. As each misspelling is found, the user is asked whether he wishes to ignore it (press "I"), mark it with a "<" (press "M") or add it to the temporary word list (press "A"). Pressing ESC will stop the checking. You can force it to add all words, or mark all words by including a third parameter "A" or "M" in the call arguments.

Where a misspelling is marked in a file the "<" overwrites the character after the offending word. This avoids problems with adjusting the sizes of the files.

If SPELL_FILE is unable to open the file it will return the appropriate error code (ERR.IU, ERR.NF etc.)

```
    PRINT SPELL_FILE (#sp_chan, mdv1_fred_doc)
or  REPORT SPELL_FILE (#sp_chan, fred_doc)  with Toolkit II
or  REPORT SPELL_FILE (#sp_chan, fred_doc, A)  Add all words
or  REPORT SPELL_FILE (#sp_chan, fred_doc, "M")  mark all
```


Device Driver Interface to QTYP_SPELL

The SPELL device is a special device driver which is used to perform spelling checks for applications programs. The SPELL device uses up to three dictionaries at a time. The first is the "standard" dictionary which is loaded from a file. This is used by all channels open to SPELL. The second is the "temporary" word list which is created during use of the spelling checker and is also used by all channels open to SPELL. The third is the "specialist" dictionary. This is loaded from file, and each channel may have its own specialist dictionary.

SPELL Open Calls

The SPELL open calls not only open a channel to the SPELL device, but they can also open a dictionary file, read the dictionary and close the file again.

The SPELL device can be configured to use any device (and standard dictionary filename), but this will not usually be necessary as the SPELL device will search all the devices WIN1_, FLP1_ and MDV1_ in that order.

It is strongly recommended that the standard dictionary should be in the file QTYP_DICTIONARY on WIN1_, FLP1_ or MDV1_.

The SPELL channels may be opened with a standard 'OPEN' call. If the access is new NEW or OVER a new standard dictionary is set up. If there is already a standard dictionary available, open NEW will return error ERR.IU, while OVER will remove the dictionary (and temporary word list) before opening the new one. If a file name is appended to the SPELL device name, then this file will be used as the standard dictionary in place of the default.

SPELL	set internal default dictionary (NEW)
SPELL_*name*	set up dictionary "name" as the standard dictionary

If the access key is OLD or SHARE, the name can specify a specialist dictionary in addition to the standard dictionary set up by a NEW or OVER open call. The words in this specialist dictionary are not accessible to other channels unless they are opened with the same specialist dictionary. If a SPELL channel is already open with a specialist dictionary of the same name, then the new channel will use that same dictionary. When all the channels using a particular specialist dictionary have been closed, the specialist dictionary is thrown away.

SPELL	use standard dictionary
SPELL_*name*	use specialist dictionary "name" as well as the standard

The open call can return all the normal QDOS error codes, and there are some additional error returns specific to the SPELL device.

ERR.BN	-12	"name" not found or it is not a dictionary	
ERR.OM	-3	out of memory	
ERR.IU	-9	standard dictionary in use (OVER)	
ERR.IU	-9	specialist dictionary file in use (OLD or SHARE)	
ERR.EX	-8	standard dictionary already set (NEW)	
ERR.NI	-19	standard dictionary not set (OLD or SHARE)	
ERR.NF	-7	SPELL device not present	)
ERR.NJ	-2	invalid job ID	) QDOS error returns
ERR.NO	-6	channel table full	)

It is the responsibility of application programs to ensure that the spell device is opened correctly, and the the dictionary is defined. The recommended procedure is to try to open the device with a SHARE key. If this fails with ERR.NI, the user should be asked to ensure that his dictionary cartridge or disk is in the appropriate drive and then the program should try to open NEW with the name "SPELL". If this fails, the application can either give up, or request that the user specifies a filename, and puts the correct medium in the correct drive.

SPELLing Checking

Normal spelling checking is carried out by sending characters to the checker. Characters may be sent one at a time using IO.SBYTE, or whole words may be checked using IO.SSTRG. In either case, the current (part of a) word being checked is held in an internal buffer. If more than 128 characters are put into this buffer, the call will return ERR.BF (buffer full).

SPELL treats all letters (English and non-English) as being parts of words.

SPELL has two different spelling checking mechanisms. In the case of checking complete words (IO.SSTRG), SPELL will skip to the first letter in the string, and check up to the first non-letter. If this word is found in the dictionary the next word will be checked in the same way. The process continues until either the string is exhausted, or the check fails. The string pointer (A1) will be updated and the most significant word of D1 will have the number of characters up to the start of the last word checked and the least significant word of D1 will be returned equal to the total number of characters checked. If no letters are found, the call will return without error.

This makes it relatively easy to check multiple word strings:

```
        MOVE.L  BUFFER,A1       set buffer pointer
        MOVE.W  BUFLEN,D2       ... and length
LOOK_WORD
        MOVE.L  SP_CHN,A0       send string to SPELL channel
        MOVEQ   #IO.SSTRG,D0
        TRAP    #3
        TST.L   D0              OK
        BEQ.S   END_WORD        ... yes
        BSR.S   SPELL_ERR       ... no, do error action
        SUB.W   D1,D2           reduce amount left to check
        BGT.S   LOOK_WORD
END_WORD
```

Within the routine SPELL_ERR the offending word may be read from the buffer using the IO.FSTRG call (see below)

The alternative mechanism is to check one byte at a time. The spelling check is reset by a separator (space, newline, carriage return, null or any printable non-letter). Non-zero error codes will only be produced when the first separator is sent after a word. At this point cursor left or delete left will recover the internal buffer contents for further modification. Any letters are added to an internal buffer; cursor left, cursor right, delete left and delete right are used to edit the internal buffer. SPELL only checks the characters to the left of the cursor. Any other character stops the checking process until SPELL is reset by a separator. If this character follows a letter which was checked, the call will return the error code appropriate to the contents of the buffer.

There are four error return codes from spelling checking.

0		no error, the word in the buffer is valid
ERR.NC	-1	not complete, the word is incomplete
ERR.NF	-7	not found, the word in the buffer is not found
special		there is a note attached to this word

The special error return will only occur when a separator is sent, or when checking words in a string, and the word in the buffer has a note in the dictionary.

If a letter, cursor left, cursor right or delete character is sent to SPELL using IO.SBYTE, then it will return the error code ERR.NC if the characters in the buffer to the left of the cursor form the start of a word in the dictionary, but are not, in themselves, a complete word. If separator or other character is sent, then the error code ERR.NF will be returned if the word is either incorrect or incomplete. SPELL will not return error code ERR.NC from any other call.

When sending single bytes to SPELL, the timeout must be 0.

Adding to the Dictionary Contents

A word may be saved in a temporary word list held within SPELL by sending an FS.SAVE command with A1 pointing to the word, and the length in D2. This temporary word list is shared with all other SPELL channels, and is maintained after the channel is closed. It is only cleared when a SPELL channel is opened with an OVERwrite key.

Viewing the Dictionary Contents

In normal operation, reading a string of bytes (IO.FSTRG) from SPELL will read the current word from the buffer.

In normal operation, reading a line (IO.FLINE) from SPELL will read the next word in the dictionary which starts with the characters in the internal buffer. Subsequent IO.FLINE calls will read subsequent matching words. If there are no more words which match, the call will return "end of file". These words can come from the temporary word list, the specialist dictionary or the standard dictionary.

If the internal buffer is clear, then IO.FLINE calls will fetch only from the temporary word list.