Contents: Spell.h

If you have any problems, want extra information or think you have found a bug you can contact me at **spellchk@quinion.demon.co.uk.**

You might also be interested in my home page which is kept up to date with latest information on this project. If there is sufficient interest a list server (either developer or just user) will be setup.

SPCHK_CheckWord (Full interface)

#include spell.h

BOOL SPCHK_CheckWord(lpchkw **)**

LPCHECKWORD lpchkw /* address of initialisation data */

The SPCHK_CheckWord function performs spelling checking on the data supplied in lpcheckword.

Parameter	Description
lpchkw	Points to a CHECKWORD structure that contains information to perform a spelling check. This structure will be passed to any callback functions.

Returns

The return is only valid if a single word is being checked. It is non zero if the word was found. It is zero if the word was not found.

Errors

Error handling is limited to the point of non-existence. Errors will be trapped, but at the moment are not passed back and instead a message is displayed to the user. Every attempt has been made to make these message understandable.

Comments

The exact way that this function works depends on the content of the CHECKWORD structure.

See Also

SPCHK_Options, CHECKWORD

SPCHK_Options (Full interface)

#include spell.h

void SPCHK_Options(lpchkw **)**

LPCHECKWORD lpchkw /* address of initialisation data */

The SPCHK_Options function provides access to the options dialog box.

Parameter	Description
lpchkw	Points to a CHECKWORD structure that contains information to perform a spelling check. This structure will be passed to any callback functions.

Returns

Nothing.

Errors

Error handling is limited to the point of non-existence. Errors will be trapped, but at the moment are not passed back and instead a message is displayed to the user. Every attempt has been made to make these message understandable.

See Also

SPCHK_CheckWord, CHECKWORD

CHECKWORD (Full interface)

```
#include spell.h

typedef struct {
    WORD          wSizeOfBlock;

    HWND          hWndParent,
                  hWndDlg;

    WORD          CheckWordOptions;
    char          szLanguage[13];
    HGLOBAL       hCustomDics;
    BYTE          NumCustom;
    BYTE          CurCustom;

    HINSTANCE     hInstance;
    DLGPROC       fpMainHook;
    DLGPROC       fpOptionsHook;
    LPSTR         lpMainDlg;
    LPSTR         lpOptionsDlg;

    DWORD         dwCustData;
    DWORD         dwCustData2;

    char          ToCheck[MAXSPELL];
    char          Changed[MAXSPELL];
    BOOL          bCurWordChanged;

    BYTE          Reserved[26+MAXSPELL];
} CHECKWORD, FAR * LPCHECKWORD;
```

The **CHECKWORD** structure contains information that the spellchk dll requires to check a document.

Member	Description
wSizeOfBlock	Specifies the length of the structure in bytes. This member is filled on input.
hWndParent	Identifies the window that owns the dialog box. This member must be a valid window handle. This is the window that will receive notification and request messages unless they are being sent to a hook (see later). This member is filled on input.
hWndDlg	Identifies the dialog window created. This member is filled on window creation.
CheckWordOptions	Initialisation flags, a combination of the following values:

Value	Meaning
CWO_ALLOWCHANGE	If a word is not found a dialog box will be display to ask what to do. If this flag is not specified then no box will be displayed.
CWO_AUTOSUGGEST	When the spell dialog box is displayed the program will automatically display a list of suggested words. Is set on

input and output.

CWO_SUGGESTCUST
: Look for suggestions in the custom dictionaries as well as the main dictionary. Set on input and output.

CWO_NOOPTIONS
: Hide the options box.

CWO_UNDO
: Send UNDO information messages and UNDO requests. If this flag is not present then the undo button will be hidden.

CWO_NOHELP
: Hide the help button. If this message is not specified then help requests will be sent.

CWO_USEMAINHOOK
: Enables the hook function specified in the **fpMainHook** member.

CWO_USEOPTIONSHOOK
: Enables the hook function specified in the **fpOptionsHook** member.

CWO_USECUSTOMMAINDLG
: Causes the program to use the dialog box template identified by the **lpMainDlg** member.

CWO_USECUSTOMOPTIONSDLG
: Causes the program to use the dialog box template identified by the **lpOptionsDlg** member.

CWO_SENDMSGTOMAINHOOK
: Sends the following notification and request messages to the hook function specified in the **fpMainHook**:
  SPELL_GETNEXT
  SPELL_WORDNOTFOUND
  SPELL_WORDCHANGED
  SPELL_CANUNDO
  SPELL_STOREUNDO
  SPELL_UNDOLAST
  SPELL_HELPMAIN
  Otherwise they are sent to **hWndParent**.

CWO_SENDMSGTOOPTIONSHOOK
: Sends the following notification and request messages to the hook function specified in the **fpOptionsHook**:
  SPELL_GETCUSTOMDEFPATH
  SPELL_HELPOPTIONS
  SPELL_HELPEDITDIC
  Otherwise they are sent to **hWndParent**.

CWO_CHECKMULTIPLE
: The program will check a series of words obtained by sending out SPELL_GETNEXT messages. If this flag is not present then only the word in **ToCheck** will be checked.

CWO_DONTUSEFULL
: Do not check the main dictionary for words. I can think of no good reason for needing this, but it is here for completion.

szLanguage The file name of the main dictionary to use. If it is not specified then the first

	available dictionary will be selected. This member is filled on input and output. The only exception to this is if CWO_DONTUSEFULL has been specified.
hCustomDics	Handle of a block of global memory containing a series of CUSTDIC structures, or **hCustomDics** can be set to 0 if no custom dictionaries are needed. The block of memory should be created with the GHND and GMEM_DDESHARE flags.
NumCustom	The number of custom dictionaries in the array to which **hCustomDics** is a handle.
CurCustom	Specified the currently selected index (into the array to which **hCustomDics** is a handle) which words should be added to. It should be a number in the range 0 to **NumCustom**-1.
hInstance	Identifies the hInstance of the program/dll which contains the dialog box resources pointed to by. **lpMainDlg** and **lpOptionsDlg**.
fpMainHook	Pointer to a hook function to handle message to the Main dialog box. Only required if the CWO_USEMAINHOOK flag is specified. The function may also handle other message if the CWO_SENDMSGTOMAINHOOK flag is specified. See these two flags for details.
fpOptionsHook	Pointer to a hook function to handle message to the Options dialog box. Only required if the CWO_USEOPTIONSHOOK flag is specified. The function may also handle other message if the CWO_SENDMSGTOOPTIONSHOOK flag is specified. See these two flags for details.
lpMainDlg	Pointer to a null-terminated string that specifies the name of the resource to be used in preference to the default dialog box. It must be present in the module specified by **hInstance**.
lpOptionsDlg	Pointer to a null-terminated string that specifies the name of the resource to be used in preference to the default dialog box. It must be present in the module specified by **hInstance**.
dwCustData	Custom data for use by your program.
dwCustData2	Custom data for use by your program.
ToCheck	A null-terminated string giving the word to be checked. It should be filled on initialisation unless the CWO_CHECKMULTIPLE flag is specified in which case it should be filled with the next word to check each time a SPELL_GETNEXT message is received.
Changed	If the word is changed by the use you program will be notified with a SPELL_WORDCHANGED message. The new word will be placed in this buffer as a null-terminated string.
bCurWordChanged	Is non zero if the current word has been changed.
Reserved	Private data used by the program.

See Also

SPCHK_CheckWord, SPCHK_Options, CUSTDIC

CUSTDIC (Full interface)

```
#include spell.h

typedef struct {
    char          DicFile[MAXPATH];
    char          DicTitle[MAXDICTITLE];
    BYTE          Options;
    BYTE          Reserved[8];
} CUSTDIC, far * LPCUSTDIC;
```

The **CUSTDIC** structure contains information on a custom dictionary.

Member	Description
DicFile	Full file path and name of the custom dictionary.
DicTitle	Title of the custom dictionary.
Options	Initialisation flags, a combination of the following values:

Value	Meaning
CD_READONLY	The dictionary is read-only. This flag is set on input and output.
CD_DISABLED	The dictionary is disabled. This flag is set on input and output.
CD_CHANGED	The custom dictionary has been changed. This flag is set on output.

Private	Private data used by the program.

See Also

SPCHK_CheckWord, SPCHK_Options, CUSTDIC

Messages (Full interface)

SPELL_GETNEXT	Ask for the next word
SPELL_WORDNOTFOUND	Notify word not found
SPELL_WORDCHANGED	Notify word changed
SPELL_CANUNDO	Ask if undo available
SPELL_STOREUNDO	Provide undo information
SPELL_UNDOLAST	Execute undo
SPELL_GETCUSTOMDEFPATH	Ask for the default custom dictionary path
SPELL_HELPMAIN	User has requested help
SPELL_HELPOPTIONS	User has requested help
SPELL_HELPEDITDIC	User has requested help

SPELL_GETNEXT (Full interface)

```
SPELL_GETNEXT
wParam  = 0;                        /* not used */
lParam  = (LPCHECKWORD) lpchkw;     /* address of initialisation data */
```

This message is sent to request the next word. The application should respond by filling the **ToCheck** member of the CHECKWORD structure.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return non zero if ToCheck has been filled or 0 to finish.

Example

This example sends 5 words to be checked one at a time.

```
LPCHECKWORD             lpchkw;
WORD                    wIndex;
char                    Words[6][15]={"thiss",
                                      "poeple",
                                      "aggrivation",
                                      "enviromental",
                                      "wheight",
                                      "nohting"};

case SPELL_GETNEXT:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;
    wIndex = (WORD) lpchkw->dwCustData;

    // Have we done all the words?
    if (wIndex>5)
        return FALSE;

    // Copy over the word
    lstrcpy(lpCheckWord->ToCheck, Words[wIndex]);

    // Position for the next
    lpchkw->dwCustData;
    return TRUE;
```

SPELL_WORDNOTFOUND (Full interface)

```
SPELL_WORDNOTFOUND
wParam  = 0;                        /* not used */
lParam  = (LPCHECKWORD) lpchkw;     /* address of initialisation data */
```

This message is sent to inform the program that the word in **ToCheck** has not been found.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return 0.

Example

Display a notice if a word is not found.

```
LPCHECKWORD             lpchkw;
char                    Text[100];

case SPELL_WORDNOTFOUND:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;

    // Display the word not found
    wsprintf(Text, "The word '%s' was not found", lpchkw->ToCheck);
    MessageBox(hDlg, Text, "Test Word", MB_ICONHAND);
    return 0;
```

See Also

SPELL_WORDCHANGED

SPELL_WORDCHANGED (Full interface)

```
SPELL_WORDNOTFOUND
wParam  = 0;                        /* not used */
lParam  = (LPCHECKWORD) lpchkw;     /* address of initialisation data */
```

This message is sent to inform the program that the word in **ToCheck** sound be changed to the word in **Changed**.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return 0.

Example

Display a notice if a word is changed.

```
LPCHECKWORD             lpchkw;
char                    Text[100];

case SPELL_WORDCHANGED:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;

    // Display the word to be changed
    wsprintf(Text, "The word '%s' should be changed to %s",
               lpchkw->ToCheck, lpchkw->Changed);
    MessageBox(hDlg, Text, "Test Word", MB_ICONHAND);
    return 0;
```

See Also

SPELL_WORDNOTFOUND

SPELL_CANUNDO (Full interface)

```
SPELL_CANUNDO
wParam  = 0;                        /* not used */
lParam  = (LPCHECKWORD) lpchkw;     /* address of initialisation data */
```

This message is sent to ask the application if it can currently undo.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return non zero if it can undo, 0 if it can't.

Example

Return the undo status.

```
static BOOL             bCanUndo;

case SPELL_CANUNDO:
    // Just return the bUndo flag
    return bCanUndo;
```

See Also

SPELL_STOREUNDO, SPELL_UNDOLAST

SPELL_STOREUNDO (Full interface)

```
SPELL_STOREUNDO
wParam  = (HUNDO)hUndo;            /* handle to undo data */
lParam  = (LPCHECKWORD) lpchkw;    /* address of initialisation data */
```

Inform the program that it should store an undo pointer.

Parameter	Description
hUndo	Handle to undo block (spellchk internal)
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return 0.

Example

Store the information needed to undo.

```
LPCHECKWORD             lpchkw;
static BOOL             bCanUndo;
static HUNDO            hUndoHandle;
static WORD             wUndoIndex;

case SPELL_STOREUNDO:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;

    // Yes we can now undo
    bCanUndo=TRUE;

    // Store the handle provided by spellchk
    hUndoHandle=(HUNDO)wParam;

    // Store the current word being checked
    wUndoIndex=lpchkw->dwCustData;
    return 0;
```

See Also

SPELL_CANUNDO, SPELL_UNDOLAST

SPELL_UNDOLAST (Full interface)

```
SPELL_UNDOLAST
wParam  = 0;                        /* not used */
lParam  = (LPCHECKWORD) lpchkw;     /* address of initialisation data */
```

The application should undo the last change it made (as a result of a SPELL_CHANGEWORD message) and reposition its pointers so that the next SPELL_GETNEXT will return the undone word.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns

An application should return non zero if it has repositioned for an undo else it should return 0. The DWL_MSGRESULT of the spellchk dialog box should be set to the undo handle previously supplied.

Example

```
LPCHECKWORD             lpchkw;
static BOOL             bCanUndo;
static HUNDO            hUndoHandle;
static WORD             wUndoIndex;

case SPELL_UNDOLAST:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;

    // Do the undo if we can - see the full example app for
    // more details
    if (bCanUndo)
    {
        // Reposition the index to restart at the pervious position
        lpchkw->dwCustData=wUndoIndex;

        // Store the undo handle in the dialog result register
        SetWindowLong(lpchkw->hWndDlg, DWL_MSGRESULT, hUndoHandle);

        // Since we have undone once we can't do it again
        bCanUndo=FALSE;

        // Return a positive answer
        return TRUE;
    }
    return FALSE;
```

See Also

SPELL_CANUNDO, SPELL_STOREUNDO

SPELL_GETCUSTOMDEFPATH (Full interface)

```
SPELL_GETCUSTOMDEFPATH
wParam  = 0;                        /* not used */
lParam  = (LPSTR) lppath;           /* buffer for file path */
```

The application should copy the default file path for custom dictionaries into the buffer pointed to by lParam.

Parameter	Description
lppath	Points to a string buffer.

Returns

An application should return non zero if it can supply a default directory, else it should return 0.

Example

Return a file path stored as a global string.

```
case SPELL_GETCUSTOMDEFPATH:
     lstrcpy((LPSTR)lParam, g.szProgDir);
     return TRUE;
```

SPELL_HELPMAIN (Full interface)

`SPELL_HELPMAIN`

The application should display help for the main window.

Parameters

This message has no parameters.

Returns

An application should return 0.

Example

Call winhelp.

```
case SPELL_HELPMAIN:
    WinHelp(hWnd, "spell.hlp", HELP_CONTEXT, HELP_SPELLMAIN);
    return 0;
```

SPELL_HELPOPTIONS (Full interface)

`SPELL_HELPOPTIONS`

The application should display help for the options window.

Parameters

This message has no parameters.

Returns

An application should return 0.

Example

Call winhelp.

```
case SPELL_HELPMAIN:
    WinHelp(hWnd, "spell.hlp", HELP_CONTEXT, HELP_SPELLOPTIONS);
    return 0;
```

SPELL_HELPEDITDIC (Full interface)

`SPELL_HELPEDITDIC`

The application should display help for the custom dictionary window.

Parameters

This message has no parameters.

Returns

An application should return 0.

Example

Call winhelp.

```
case SPELL_HELPMAIN:
    WinHelp(hWnd, "spell.hlp", HELP_CONTEXT, HELP_SPELLCUSTEDIT);
    return 0;
```

SPEDT_CheckEdit (Quick interface)

#include spell.h

BOOL SPEDT_CheckEdit(hwndEdit **)**

HWND hwndEdit /* window handle of the edit box to check */

The SPEDT_CheckEdit function checks a windows edit box (or compatible) for spelling.

Parameter	Description
hwndEdit	Identifies the edit box to be checked.

Returns

Returns non zero if successful, otherwise it will return 0.

Comments

To be compatible the edit box should respond to the following edit box messages:

EM_GETLINE
EM_GETLINECOUNT
EM_LINEFROMCHAR
EM_LINEINDEX
EM_LINELENGTH
EM_REPLACESEL
EM_SETSEL

And should be aware that the bitwise operator:

ES_MULTILINE

is checked for in the style of the control using GetWindowLong(hwndEdit, GWL_STYLE)

See Also

SPEDT_SetupBox

SPEDT_SetupBox (Quick interface)

#include spell.h

void SPEDT_SetupBox(hwndParent **)**

HWND hwndParent /* Parent window */

The SPEDT_SetupBox function displays the setup box.

Parameter	Description
hwndParent	Identifies the parent window of the setup box.

Returns

Nothing.

Comments

Because of the way this interface works options are displayed as if the setup program that comes with spell200.zip has been run. This merely provides an interface to run it from within your own program.

See Also

SPEDT_CheckEdit

Copyright and disclaimer

Spell checker for edit boxes has been written by and is copyright © 1994 by Brian Quinion. All rights reserved.

Whilst every care has been taken in the compilation of this application, it is provided 'as is' and neither the author nor the publishers shall be held responsible for any error, omission or consequential loss.

Spell Checker for Edit Boxes is freeware.