Contents: Spell.h

If you make use of this module PLEASE contact me so that I can let you know about updates as soon as possible.

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.

http://clever.net/quinion/

SPCHK_CheckWord (2.00: 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_Version (2.40: Full interface)

#include spell.h

WORD SPCHK_Version(void **)**

Returns the version number of the loaded module.

Returns

The return gives the version number in the format

MajorVersion*100+MinorVersion

eg for version 3.00 the return would be 300

SPCHK_Options (2.00: 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

SPCHK_GetValidLettersBlock (3.00: Full interface)

#include spell.h

void SPCHK_GetValidLettersBlock(lpValidChars **)**

LPSTR lpValidChars /* address to copy valid chars data to */

The SPCHK_GetValidLettersBlock function copies a 256 byte data block for the current language giving information on which letters will be checked by the program.

Parameter	Description
lpValidChars	Points to a 256 byte array to which the data will be copied.

Returns

Nothing.

Comments

Each byte of the array returned gives the status of the corresponding letter.

Parameter	Description
0	Letter ignored. Your program should not send words containing this letter
1	Completely valid, always checked. Should be considered part of a word.
2	Punctuation. Should be considered part of a word.
3	Other special character. Should be considered part of a word.

CHECKWORD (3.00: Full interface)

```
#include spell.h

typedef struct {
    WORD          wSizeOfBlock;

    HWND          hWndParent,
                  hWndDlg;

    DWORD         CheckWordOptions;

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

    HFILE         hDebugLog;

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

    DWORD         dwCustData;
    DWORD         dwCustData2;

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

    char          PrevWord[MAXSPELL];
    WORD          wFunctionType;

    BYTE          CurPosCheck;
    BYTE          CurPosChanged;
    char          CurWord[MAXSPELL];

    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.

CWO_USEASCIIACCENTS
Translate **a"**, **i"**, **o~** to **ä**, **ï**, **õ**.

CWO_PUNCTUATIONCANBREAK
Let full stops, commas and other punctuation break words, useful for text that contains taging, but disables checking for mistakes like failure to put in breaks after punctuation.

CWO_DEBUGLOG
Log debug information in the file with the handle provided in hDebugLog.

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.

hDebugLog File handle for the file to which debug information should be written.

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.

PrevWord	The last word that was checked. This is used only to look for repeated words. As a result checking for repeated words and be disabled using PrevWord[0]=0; during each SPELL_GETNEXT. You should disable repeated word checking after a new line.
wFunctionType	Records the type off error that was found

Value	**Meaning**
WORD_NOTFOUND	Word was not found in any of the dictionaries.
WORD_REPEATED	Word was the same as the previous word (as recorded in PrevWord).

CurPosCheck	The position of the letter after the last letter of CurWord in the ToCheck array. Change with caution.
CurPosChanged	The position in the Changed array which has been reached. For referance only, this value will be ignored if changed.
CurWord	The current word being checked the will be either the whole or a sub string of the current ToCheck.
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];
    char          DicLanguage[13];
    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.
DicLanguage	The filename of the language that this dictionary will be used with or 'all' to specify that it will be used with all languages.
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.
CD_BYLANGUAGE	The custom dictionary is disabled due to the language.

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_WORDREPEATED	Notify word repeated (3.00)
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_LANGUAGECHANGED	Notify that the user has change language (3.00)
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_WORDREPEATED (Full interface, 3.00)

```
SPELL_WORDREPEATED
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_WORDREPEATED:
    // Get the pointer
    lpchkw = (LPCHECKWORD)lParam;

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

See Also

SPELL_WORDCHANGED

SPELL_WORDCHANGED (Full interface)

```
SPELL_WORDCHANGED
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_LANGUAGECHANGED (Full interface, 3.00)

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

Inform the program that the language has been changed. You should reload the character validation table if you use it.

Parameter	Description
lpchkw	Points to a CHECKWORD structure.

Returns
Is ignored.

Example

```
static BYTE             CharTable[256];

case SPELL_LANGUAGECHANGED:
     SPCHK_GetValidLettersBlock(LetterStates);
     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_Version (2.40: Quick interface)

#include spell.h

WORD SPEDT_Version(void **)**

Returns the version number of the loaded module.

Returns

The return gives the version number in the format

MajorVersion*100+MinorVersion

eg for version 2.40 the return would be 240

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_CheckEditCustom, SPEDT_SetupBox, SPEDT_SetupCustom

SPEDT_CheckEditCustom (2.40: Quick interface)

#include spell.h

BOOL SPEDT_CheckEditCustom(hwndEdit, hInstance, lpDlgBox, lpOptBox**)**

HWND hwndEdit /* window handle of the edit box to check */
HINSTANCE hInstance /* hInstance of the module containing the Dlg template */
LPSTR lpDlgBox /* name of main dialog box template */
LPSTR lpOptBox /* name of options dialog box template */

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

Parameter	Description
hwndEdit	Identifies the edit box to be checked.
hInstance	Specifies the module containing the dialog box templates.
lpDlgBox	Must contain either a pointer to a string giving the dialog box resource name stored in module hInstance or be NULL to use the default box.
lpOptBox	Must contain either a pointer to a string giving the dialog box resource name stored in module hInstance or be NULL to use the default box. NOTE: This is the spellchk default NOT spelledt default box.

Returns

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

Comments

You should note that the dialog box templates MUST contain all the buttons and controls present in the original box, but they may be hidden or moved out of the visible area. The exception is the KEYSELECT box, this MUST be removed in version 2.40 or 3.00. 3.01 will work either way.

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_CheckEdit, SPEDT_SetupBox, SPEDT_SetupCustom

SPEDT_CheckEditClip (3.00: Quick interface)

#include spell.h

BOOL SPEDT_CheckEditClip(hwndEdit **)**

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

The SPEDT_CheckEditClip function checks a windows by copying the content to the clipboard checking it there and then pasting it back.

Parameter	Description
hwndEdit	Identifies the edit box to be checked.

Returns

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

Comments

There is a bug in the released version of spell 3.00 that makes this function unusable. It is correctly implemented in the 3.00 DEVELOPER dlls and will not be available until version 3.01 of the public dlls. If you use this function you MUST provide the dlls with your program.

See Also

SPEDT_CheckEditCustom, SPEDT_SetupBox, SPEDT_SetupCustom

SPEDT_CheckClipboard (3.00: Quick interface)

#include spell.h

WORD SPEDT_CheckClipboard(hwndEdit **)**

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

The SPEDT_CheckClipboard function checks the current windows clipboard.

Parameter	Description
hwndEdit	Identifies the parent window that should be used with the 'Spell' box.

Returns

Returns a word of flag bits, multiple bits may be set:

Value	Meaning
CHECKGLOBAL_UNRECOVERABLEERROR	The program was unable to start
CHECKGLOBAL_FORMATNOTKNOWN	The format specified in wType in unknown
CHECKGLOBAL_CHANGED	The global memory block has been changed
CHECKGLOBAL_FINISHED	The entire text was checked

Comments

This function is only implemented in the 3.00 DEVELOPER dlls and will not be available until version 3.01 of the public dlls. If you use this function you MUST provide the dlls with your program.

See Also

SPEDT_CheckEditCustom, SPEDT_SetupBox, SPEDT_SetupCustom

SPEDT_CheckGlobal (3.00: Quick interface)

#include spell.h

WORD SPEDT_CheckGlobal(hwndEdit, lpToCheck, wType, hwndParent **)**

HWND hwndEdit /* window handle of the text box to check */
HGLOBAL far * lpToCheck /* global memory item containing text to check */
WORD wType /* type of data */
HWND hwndParent /* window to which progress/information messages are sent */

The SPEDT_CheckGlobal function checks a block of global memory, including formatting.

Parameter	Description
hwndEdit	Identifies the window that will be used as the parent window.
lpToCheck	A far pointer to a handle to a block of global memory in the format identified by wType.
wType	Type of data:

Value	Meaning
'Rich Text Format'	
'Rich Text'	Text will be treated as formatted rich text. The actual value should be determined using RegisterClipboardFormat("Rich Text Format") or RegisterClipboardFormat("Rich text")
HTML Text Format	
HTML Format (3.01)	
CF_HTML (3.01)	
HTML (3.01)	Text till be treated as HTML tagging text within <> will be ignored, and commented text <!-- --> will be ignored.
CF_TEXT	Treated as plain text
CF_OEMTEXT	Text is converted to ANSI then processed. The return data is converted back.

hwndParent	Window to which notification message will be sent. (3.01)

Returns

Returns a word of flag bits, multiple bits may be set:

Value	Meaning
CHECKGLOBAL_UNRECOVERABLEERROR	The program was unable to start
CHECKGLOBAL_FORMATNOTKNOWN	The format specified in wType in unknown
CHECKGLOBAL_CHANGED	The global memory block has been changed
CHECKGLOBAL_FINISHED	The entire text was checked

Comments

Some clipboard formats will not be supported until 3.01.

See Also

SPEDT_CheckEditCustom, SPEDT_SetupBox, SPEDT_SetupCustom

SPEDT_CheckGlobalCustom (3.00: Quick interface)

#include spell.h

WORD SPEDT_CheckGlobalCustom(hwndEdit, lpToCheck, wType, hInstance, lpMainDlg, lpOptionsDlg, hwndParent **)**

HWND hwndEdit /* window handle of the text box to check */
HGLOBAL far * lpToCheck /* global memory item containing text to check */
WORD wType /* type of data */
HINSTANCE hInstance /* module in which the two dialog box templates will be found */
LPSTR lpMainDlg /* pointer to the name of the dialog box to use for Main*/
LPSTR lpOptionsDlg /* pointer to the name of the dialog box to use for Options*/
HWND hwndParent /* window to which progress/information messages are sent */

The SPEDT_CheckGlobal function checks a block of global memory, including formatting.

Parameter	Description
hwndEdit	Identifies the window that will be used as the parent window.
lpToCheck	A far pointer to a handle to a block of global memory in the format identified by wType.
wType	Type of data:

Value	Meaning
'Rich Text Format'	
'Rich Text'	Text will be treated as formatted rich text. The actual value should be determined using RegisterClipboardFormat("Rich Text Format") or RegisterClipboardFormat("Rich text")
HTML Text Format	
HTML Format (3.01)	
CF_HTML (3.01)	
HTML (3.01)	Text till be treated as HTML tagging text within <> will be ignored, and commented text <!-- --> will be ignored.
CF_TEXT	Treated as plain text
CF_OEMTEXT	Text is converted to ANSI then processed. The return data is converted back.

hInstance	The hInstance of the module which contains the dialog box templates.
lpMainDlg	Name of the dialog box contained in hInstance which will be used for the main spelling box.
lpOptionsDlg	Name of the dialog box contained in hInstance which will be used for the options spelling box. Can be ignored if the options box will not be accessible.
hwndParent	Window to which notification message will be sent. (3.01)

Returns

Returns a word of flag bits, multiple bits may be set:

Value	Meaning
CHECKGLOBAL_UNRECOVERABLEERROR	The program was unable to start

CHECKGLOBAL_FORMATNOTKNOWN	The format specified in wType in unknown
CHECKGLOBAL_CHANGED	The global memory block has been changed
CHECKGLOBAL_FINISHED	The entire text was checked

Comments

Some clipboard formats will not be supported until 3.01. You should ensure that all the controls in the dialog boxes are the same or exactly compatible to the ones in the original. Controls should not be deleted, although they can be moved outside the visible area or have their WS_VISABLE falg removed.

See Also

SPEDT_CheckEditCustom, SPEDT_SetupBox, SPEDT_SetupCustom

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 spell300.zip has been run. This merely provides an interface to run it from within your own program.

See Also

SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupCustom

SPEDT_SetupBoxLimited (3.00: Quick interface)

#include spell.h

void SPEDT_SetupBoxLimited(hwndParent **)**

HWND hwndParent /* Parent window */

The SPEDT_SetupBoxLimited function displays the setup box as it is shown when the options button is pressed during a check without all the configuration options..

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

Returns

Nothing.

See Also

SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupCustom

SPEDT_SetupCustom (Quick interface)

#include spell.h

void SPEDT_SetupBox(hwndParent, hInstance, lpDlgBox **)**

HWND hwndParent /* Parent window */
HINSTANCE hInstance /* hInstance of the module containing the Dlg template */
LPSTR lpDlgBox /* name of main dialog box template */

The SPEDT_SetupBox function displays the setup box.

Parameter	Description
hwndParent	Identifies the parent window of the setup box.
hInstance	Specifies the module containing the dialog box templates.
lpDlgBox	Must contain either a pointer to a string giving the dialog box resource name stored in module hInstance or be NULL to use the default box. NOTE: This is the spellchk default NOT spelledt default box.

Returns

Nothing.

Comments

This provides a method to partially customise the display of the setup box. As yet there is no easy way to add to the options, but they can now be removed if you do not want them to appear. You should note that the dialog box template MUST contain all the buttons and controls present in the original box, but they may be hidden or moved out of the visible area,

See Also

SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupBox

SPEDT_GetParameter (Quick interface)

#include spell.h

BOOL SPEDT_GetParameter(wParam, lpData **)**

WORD wParam /* Identify the parameter to retrieve */
LPVOID lpData /* buffer in which the data will be placed */

Get the current settings of various spelledt parameters.

Parameter	Description	
wParam	Identifies the particular parameter that will be retrieved.	
	Value	**Returns in lpData**
	SPEDT_LANGUAGE	The filename of the current dictionary (string)
	SPEDT_XPOS	The current X position of the spell window, or the position at which it will be created. (int)
	SPEDT_YPOS	The current Y position of the spell window, or the position at which it will be created. (int)
	SPEDT_STOREWINPOS	Store the position of the spell window on exit. (BOOL)
	SPEDT_AUTOSUGGEST	Auto suggest. (BOOL)
	SPEDT_SUGGESTFROMCUSTOM	Suggest from custom dictionaries. (BOOL)
	SPEDT_ASCIITOANSI	Use ASCII accents. (BOOL)
	SPEDT_IGNORELINES	The first byte returned is a Boolean flag indicating if lines are being ignored. The rest of the returned data is a zero terminated string giving the text that ignored lines should start with.
	SPEDT_DOAFTER	After a check if: 0 Show message box 1 Beep 2 Do nothing The return is an int.
	SPEDT_ONPUNCTUATION	A BOOL, indicates if punctuation can break words.
	SPEDT_DEBUG	Debug is in operation (BOOL)
	SPEDT_PROGRESSINDICATOR	Show the progress indicator (BOOL)
lpData	A pointer to the buffer in which the data will be placed.	

Returns

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

See Also

SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupBox

SPEDT_SetParameter (Quick interface)

#include spell.h

BOOL SPEDT_SetParameter(wParam, lpData **)**

WORD wParam /* Identify the parameter to retrieve */
LPVOID lpData /* buffer from which the data will be read */

Set various spelledt parameters.

Parameter	Description
wParam	Identifies the particular parameter that will be changed.

Value	Returns in lpData
SPEDT_LANGUAGE	The filename of the current dictionary (string)
SPEDT_XPOS	The current X position of the spell window, or the position at which it will be created. (int)
SPEDT_YPOS	The current Y position of the spell window, or the position at which it will be created. (int)
SPEDT_STOREWINPOS	Store the position of the spell window on exit. (BOOL)
SPEDT_AUTOSUGGEST	Auto suggest. (BOOL)
SPEDT_SUGGESTFROMCUSTOM	Suggest from custom dictionaries. (BOOL)
SPEDT_ASCIITOANSI	Use ASCII accents. (BOOL)
SPEDT_IGNORELINES	The first byte returned is a Boolean flag indicating if lines are to be ignored. The rest of the returned data is a zero terminated string giving the text that ignored lines should start with.
SPEDT_DOAFTER	After a check if: 0 Show message box 1 Beep 2 Do nothing The value should be in the form of an int.
SPEDT_ONPUNCTUATION	A BOOL, indicates if punctuation can break words.
SPEDT_DEBUG	Debug is in operation (BOOL)
SPEDT_PROGRESSINDICATOR	Show the progress indicator (BOOL)

lpData	A pointer to the buffer in which the data will be placed.

Returns

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

See Also

SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupBox

Copyright and disclaimer

License information

I prefer to use a royalty system for licensing, it seems fairer to everyone (myself included).

Number of copies	Cost per copy
50 - 99	£7
100 - 999	£4
1000 - 4999	£2
5000 +	£1

At my discretion these fees may be reduced if the program is either being sold cheaply or has some particular merit. Contacting me for more information (Brian@quinion.demon.co.uk).

About the Author

I was a second year A level student at the Castle School, Thornbury, which is near Bristol in the UK. I'm now waiting for my results before going to University hopefully Software Engineering at Sheffield. I'm already a keen programmer.

Other than computers cycling, squash and juggling are my main hobbies. I also read a fair bit of S.F.

Contact:

I can be contacted by Email at either:

Spellchk@quinion.demon.co.uk
Brian@quinion.demon.co.uk

By mail:

Brian Quinion
18 Pittville Close,
Thornbury,
BRISTOL
BS12 1SE
UK

If I am online I will probably be on IRC as Brique, (yes I know, very unimaginative)

The latest information will be on my home page:

http://clever.net/quinion/

Examples: MS Access (Quick interface)

Function definitions

Declare Function **SPEDT_CheckEdit** Lib "SPELLEDT.DLL" (ByVal hWnd As Integer) As Integer

Declare Sub **SPEDT_SetupBox** Lib "SPELLEDT.DLL" (ByVal hWnd As Integer)

A couple of methods for adding spell checker to MS Access, the first thanks to John W. Dickerson, but while you are free to thank him if it works, don't blame him if it does not!

I place event procedures on AfterUpdate, Enter, and on Exit of the control that I want to check. I have also placed a hidden field on the form that will hold text (the field is called "key" and the values can be either "updated" or " " (blank)). This field will let me know if the field has been updated or not. Based on whether or not the field has been updated, the code in the Exit event executes if Me![Key] = "Updated". In order to have the results returned to the screen after a spell check, I use a send keys to send a tab so I will move to the next field. Also, when I do the send keys that executes in the macro named "Spell" I send a "shift tab tab" which will highlight the whole field so that the spell checker will check the whole value of the field word by word. This was kind of quirky to figure out, I just kept trying different key combinations until it worked. Also I have set the hot key to your spell checker as Ctrl F1, which is reflected in the Spell Macro below.

```
Sub Trigger_AfterUpdate ()
    Me![Key] = "Updated"
End Sub
```

```
Sub Trigger_Enter ()
    Me![Key] = " "
End Sub
```

```
Sub Trigger_Exit (Cancel As Integer)
    If Me![Key] = "Updated" Then
        DoCmd RunMacro "Spell"
        Me![Key] = " "
        DoCmd RunMacro "Tab"
    End If
End Sub
```

Spell Macro contains...
SendKeys +{Tab}{Tab}^{F1},Yes

Tab Macro contains...
Sendkeys {Tab},Yes

Next my solution:

(Declarations)

```
Declare Function SPEDT_CheckEdit Lib "spelledt.dll" (ByVal HWnd As Integer) As Integer
Declare Function GetFocus Lib "user.exe" () As Integer
Dim TempText As String
```

```
Function CheckEdit ()
    Dim i As Integer
```

```
    If Len(TempText) = 0 Then
        i = SPEDT_CheckEdit(GetFocus())
        TempText = Screen.ActiveControl.Text
    End If
End Function


Function ExitEditBox ()
    If Len(TempText) > 0 Then
        Screen.ActiveControl.Text = TempText
        SendKeys "{ENTER}"
        TempText = ""
    End If
End Function
```

They should be setup on the edit control as:

```
AfterUpdate: =CheckEdit()
OnExit: =ExitEditBox()
```

Explanation:

When you hit return after editing the box the AfterUpdate function CheckEdit() is called, which does the check and sets TempText to the new text. Access then resets the text and calls OnExit which seeing that there is text in TempText resets the text in the edit box to the correct value, and in the process gives itself back the focus. It then send an ENTER key to quit the edit box and this time since TempText contains text CheckEdit does not do a check. OnExit is called and although it changes the text it is the same as the text already in it so no AfterUpdate message is sent and the function finally drops through, resetting TempText on the way out!

Examples: Visual Basic (Quick interface)

Function definitions

Declare Function **SPEDT_CheckEdit** Lib "SPELLEDT.DLL" (ByVal hWnd As Integer) As Integer

Declare Sub **SPEDT_SetupBox** Lib "SPELLEDT.DLL" (ByVal hWnd As Integer)

To use the quick interface in Visual Basic, you need to include these in the declarations section of your module. If SPELLEDT.DLL is not in your \Windows directory, \Windows\System directory, or the directory you ran the .EXE that calls SPELLEDT.DLL you need to change "SPELLEDT.DLL" in the declaration to include the path of SPELLEDT.DLL.

Usage

To call the spell checker, use the following code:

```
i = SPEDT_CheckEdit( EditBox.hWnd )
```

'i' is an integer and 'EditBox' is the name of the text box you want to spell check.

To call the setup screen use:

```
call SPEDT_SetupBox( EditBox.hWnd )
```

See Also

SPEDT_CheckEdit, SPEDT_CheckEditCustom, SPEDT_CheckEdit, SPEDT_SetupBox