AppLocalizer DLL

Version 1.09

Software Documentation

www.southsoftware.com
support@southsoftware.com

1 Contents

2 Abstract

AppLocalizer DLL is a localization solution for your applications. It allows your application to store all its text strings in a single language file. AppLocalizer DLL provides easy way to load and use language files which have either INI or XML format. Using AppLocalizer DLL your translator will only need to translate a single file, which helps cut costs and speeds up the localization process. Translator does not need to care that the length of the translated text can be increased or decreased and affect the layout of dialog box controls. AppLocalizer DLL solves this problem by replacing standard Windows dialog box templates by the XML-formatted dialog box templates, which format is very similar to HTML pages. In other words, with AppLocalizer DLL every dialog box is an HTML table where each cell may contain a control, a group of controls or an enclosed table. So there is no matter how translated text is expanded because it will never affect the layout of the dialog box. AppLocalizer DLL comes with an XML dialog editor which provides an easy way for creating and editing the XML dialogs. AppLocalizer DLL also supports XML based templates for menus and XML based templates for pages of Property Sheets. Using a sample project, which comes with AppLocalizer DLL, you can better understand how to integrate AppLocalizer DLL into your Visual C++ application. Integrating AppLocalizer DLL requires you only to replace menu and dialog box templates so in most cases you will not need even to modify dialog box procedures. AppLocalizer DLL supports all 32 and 64-bit Windows platforms including Windows 95, 98, Me, NT 4.0, 2000, XP, 2003, 2008 and Vista.

3 Registration

Your trial of AppLocalizer DLL will expire in 30 days after installation. When your trial period is expired you will get error `1931` (`ERROR_CONTEXT_EXPIRED`) as a result of main AppLocalizer DLL functions.

If you wish to continue using AppLocalizer DLL or if you wish to redistribute it with your own application then you have to register your copy of AppLocalizer DLL with us.

To register AppLocalizer DLL follow this link:

http://applocalizer.southsoftware.com/purchase.html

After registering you will receive full version of AppLocalizer DLL and your registration key. The full version of AppLocalizer DLL has the same interface as trial version but it is smaller in size and works a lot faster. Use the `XMLLangSetKey` function to enter your registration key before using any other AppLocalizer DLL functions. Use the following source code example to find out how to enter the registration key:

```
XMLLangSetKey(_T("Your registration key here"));
```

4 How to use

AppLocalizer DLL package includes the following files:

Windows 32-bit	***Windows 64-bit***
`dll\` `applocx32a.dll` – ANSI DLL `applocx32a.lib` – ANSI LIB `applocx32w.dll` – UNICODE DLL `applocx32w.lib` – UNICODE LIB	`dll\` `applocx64a.dll` – ANSI DLL `applocx64a.lib` – ANSI LIB `applocx64w.dll` – UNICODE DLL `applocx64w.lib` – UNICODE LIB
`include\` `xmllng.h` – Multilanguage string table C/C++ header file `xmlmnu.h` – Multilanguage XML menus C/C++ header file `xmldlg.h` – Multilanguage XML dialogs C/C++ header file `xmlps.h` – Multilanguage XML property sheets C/C++ header file	

`applocalizer.pdf` – this documentation
`applocalizer.url` – URL to the webpage
`files.txt` – list of files
`license.txt` – license agreement
`readme.txt` – readme file
`register.txt` – registration information
`dlgedit\` – XML dialog editor
`sample\` – Microsoft Visual C++ sample

To use any functions from AppLocalizer DLL you need to link your application with one of the following files:
`applocx32a.dll` - 32-bit ASCII DLL;
`applocx32w.dll` - 32-bit UNICODE DLL;
`applocx64a.dll` - 64-bit ASCII DLL;
`applocx64w.dll` - 64-bit UNICODE DLL.

4.1 Creating string table

Before using AppLocalizer DLL functions you need to initialize a string table. String table is an array of `LOADSTRINGS` structures, which defines where in the language file the corresponding string is located and where must be placed the pointer to the loaded string.

The following example shows a string table definition:

```
typedef struct tagSTRINGTABLE {
    LPTSTR lpOK;
    LPTSTR lpCancel;
    LPTSTR lpApply;
    LPTSTR lpBuyNow;
    LPTSTR lpHelp;
    LPTSTR lpBack;
    LPTSTR lpNext;
    LPTSTR lpFinish;
} STRINGTABLE, FAR *LPSTRINGTABLE;

LOADSTRINGS loadstrings[] = {
{ _T("Buttons"), _T("OK"), NULL, &strtbl.lpOK, NULL },
{ _T("Buttons"), _T("Cancel"), NULL, &strtbl.lpCancel, NULL },
{ _T("Buttons"), _T("Apply"), NULL, &strtbl.lpApply, NULL },
{ _T("Buttons"), _T("BuyNow"), NULL, &strtbl.lpBuyNow, NULL },
{ _T("Buttons"), _T("Help"), NULL, &strtbl.lpHelp, NULL },
{ _T("Buttons"), _T("Back"), NULL, &strtbl.lpBack, NULL },
{ _T("Buttons"), _T("Next"), NULL, &strtbl.lpNext, NULL },
{ _T("Buttons"), _T("Finish"), NULL, &strtbl.lpFinish, NULL }
};
```

Where the language file contains the following:

```
[Buttons]
OK=OK
Cancel=Cancel
Apply=&Apply Now
BuyNow=Bu&y Now
Help=&Help
Back=&Back
Next=&Next
Finish=&Finish
```

4.2 Determining list of installed languages

AppLocalizer DLL does not enumerate language files internally so it is the application's responsibility to determine the list of installed languages. AppLocalizer DLL provides the `XMLLangDefMultilang` function to handle the list of installed languages. The following example shows how to determine the list of installed languages by enumerating the `.lng` files in the folder where the executable file is located:

```
MULTILANGAVAILABLE DefLangList = {0};

TCHAR szLangDirectory[MAX_PATH];
GetModuleFileName(g_hinst, szLangDirectory,
    sizeof(szLangDirectory)/sizeof(szLangDirectory[0]));
int i;
for(i=lstrlen(szLangDirectory);i>=0;i--)
    if (szLangDirectory[i]==_T('\\')) break;
_sntprintf(szLangDirectory+i, (sizeof(szLangDirectory)/
    sizeof(szLangDirectory[0]))-i, _T("\\*.lng"));

WIN32_FIND_DATA FindFileData;
HANDLE hSearch = FindFirstFile(szLangDirectory, &FindFileData);
if (hSearch != INVALID_HANDLE_VALUE)
{
    do {
        // Test if found file fits to a language file name.
        TCHAR szLangFileName[50];
        for (int i=0;i<MULTILANG_MAXLANG;i++)
        {
            XMLLangGetFileName(i, szLangFileName);
            _tcscat(szLangFileName, _T(".lng"));
            if (_tcsicmp(szLangFileName,
                FindFileData.cFileName)==0)
            {
                XMLLangDefMultilang(DefLangList,
                    MULTILANG_ADDMULTILANG, i);
                break;
            }
        }
    } while (FindNextFile(hSearch, &FindFileData));
    // Close the search handle.
    FindClose(hSearch);
}
```

The following example shows how to fill a combo box with the list of installed languages where the active language is selected:

```
for (int i=0;i<MULTILANG_MAXLANG;i++) {
    if (XMLLangDefMultilang(DefLangList,
        MULTILANG_CHECKMULTILANG, i)!=0) {
        LRESULT idx = SendDlgItemMessage(hwndDlg,
            ID_PREF_COMBO, CB_ADDSTRING, 0,
            (LPARAM)g_strtbl.LanguageNameTable[i]);
        SendDlgItemMessage(hwndDlg, ID_PREF_COMBO,
            CB_SETITEMDATA, idx, (LPARAM)i);
        if (g_nCurrLang==i) SendDlgItemMessage(hwndDlg,
            ID_PREF_COMBO, CB_SETCURSEL, idx, 0);
    }
}
```

4.3 Loading string table

Stringtable is loaded by `XMLLangLoadStrings` function. `XMLLangLoadStrings` loads strings from a file specified in the `lpFile` parameter of the `LOADLANGFILESTRUCT` structure, if not all strings are loaded, it tries to load remaining strings from a file specified in the `lpDefFile` parameter of the `LOADLANGFILESTRUCT` structure. The following example shows how to use the `XMLLangLoadStrings` function to load a stringtable from a language file:

```
BOOL IsFileExist(LPCTSTR lpszFileName)
{
    HANDLE hSearch;
    WIN32_FIND_DATA FileData;

    hSearch = FindFirstFile(lpszFileName, &FileData);
    if (hSearch != INVALID_HANDLE_VALUE)
    {
        FindClose(hSearch);
        return TRUE;
    }
    return FALSE;
}

BOOL InitLanguage(void)
{
    // get lang file name
    TCHAR szLangFileName[50];
    XMLLangGetFileName(g_nCurrLang, szLangFileName);
    _stprintf(g_szLangFileName, _T("%s\\%s.lng"),
        g_szStartupDirectory, szLangFileName);
    if (g_nCurrLang!=MULTILANG_ENGLISH_US &&
        !IsFileExist(g_szLangFileName))
    {
        // if lang file does not exist, use enlish by default
        XMLLangGetFileName(MULTILANG_ENGLISH_US,
            g_szLangFileName);
        _stprintf(g_szLangFileName, _T("%s\\%s.lng"),
            g_szStartupDirectory, szLangFileName);
    }

    LOADLANGFILESTRUCT LoadLangFile;
    LoadLangFile.lpFile = g_szLangFileName;
    LoadLangFile.FileSource = LANGFILESOURCE_FILE;
    LoadLangFile.FileType = LANGFILETYPE_INI;
    LoadLangFile.lpDefFile =
        MAKEINTRESOURCE(DEFAULT_LNGTABLE_RESOURCE_ID);
    LoadLangFile.DefFileSource = LANGFILESOURCE_RES;
    LoadLangFile.DefFileType = LANGFILETYPE_INI;
    LoadLangFile.DefFileExtra.lpDefResType =
        MAKEINTRESOURCE(RT_LNGTABLE);
    return (XMLLangLoadStrings(g_hinst, loadstrings,
        sizeof(loadstrings)/sizeof(loadstrings[0]),
        &LoadLangFile)>0);
}
```

When stringtable is not longer needed, it should be destroyed by calling the `XMLLangFreeStrings` function. Also the stringtable should be destroyed before it is re-loaded, when program changes its language. The following example shows how to use the `XMLLangFreeStrings` function to destroyed a stringtable:

```
int count = XMLLangFreeStrings(loadstrings,
    sizeof(loadstrings)/sizeof(loadstrings[0]));
```

4.4 Validating strings

To avoid some errors in format specification characters and eliminate possible memory violations all strings are validated when they are loaded. There is a `lpszValidate` member in the `LOADSTRINGS` structure for validate each string. The validation function just converts all % format specifiers to the form corresponding to the validation template - `lpszValidate`. Validation template may be `NULL`, then there should not be any specifiers in the string.
The following examples explain how the validation works:

Language file	***Validation template***	***Result***
Text1 %% and %% text2	_T("%s %d")	Text1 %s and %d text2
Text\n\ta\nb\sc\%d	NULL	Text a b c%%d
Text %%	_T("%")	Text %

4.5 Creating a menu

A menu can be created from a plain XML template or from a template that is created dynamically with the `XMLTemplateString`, `XMLTemplateStringPtr`, `XMLTemplateMenuBegin`, `XMLTemplateMenuItem` and `XMLTemplateMenuEnd` functions. Creating templates dynamically provides more control on such things as styles and identifiers. The following example shows how to create a simple menu:

```
int GetMenuXMLTemplate(TCHAR *xmldata, int xmlsize)
{
    int len = XMLTemplateMenuBegin(xmldata, xmlsize,
        NULL, NULL, 0);
    len += XMLTemplateMenuBegin(xmldata+len, xmlsize-len,
        g_strtbl.lpMenuFile, _T("enabled"), 0);
    len += XMLTemplateMenuItem(xmldata+len, xmlsize-len,
        g_strtbl.lpMenuExit, _T("enabled"), cm_exit);
    len += XMLTemplateMenuEnd(xmldata+len, xmlsize-len);
    len += XMLTemplateMenuEnd(xmldata+len, xmlsize-len);
    return len;
}

HMENU CreateMenuFromXMLTemplate(void)
{
    HMENU hMenu = NULL;
    LPTSTR xmldata = NULL;
    int xmlsize = 0;
    int len;
    do {
        xmlsize += 4096;
        LPTSTR newxmldata = (LPTSTR)realloc(xmldata,
            xmlsize*sizeof(TCHAR));
        if (newxmldata)
        {
            xmldata = newxmldata;
        }
        else
        {
            if (xmldata)
            {
                free(xmldata);
                xmldata = NULL;
            }
            break;
        }
        len = GetMenuXMLTemplate(xmldata, xmlsize);
    } while (len==xmlsize-1);

    if (xmldata)
    {
        hMenu = XMLCreateMenu(xmldata, len, NULL);
        free(xmldata);
    }
    return hMenu;
}
```

4.7 Creating a dialog box

A dialog box can be created from a plain XML template or from a template that is created dynamically with the `XMLTemplateString`, `XMLTemplateStringPtr`, `XMLTemplateDlgBegin`, `XMLTemplateDlgEnd`, `XMLTemplateDlgGroupBegin`, `XMLTemplateDlgGroupEnd`, `XMLTemplateDlgPanelBegin`, `XMLTemplateDlgPanelEnd`, `XMLTemplateDlgRowBegin`, `XMLTemplateDlgRowEnd`, `XMLTemplateDlgColBegin`, `XMLTemplateDlgColEnd` and `XMLTemplateDlgControl` functions. Creating templates dynamically provides more control on such things as styles and identifiers. The following example shows how to create a simple dialog box:

```
int GetDlgXMLTemplate(TCHAR *xmldata, int xmlsize)
{
    int len = XMLTemplateDlgBegin(xmldata, xmlsize,
        _T("Sample dialog box"), _T("MS Sans Serif, 10"),
        WS_DLGFRAME | WS_POPUP | WS_BORDER | WS_VISIBLE |
        WS_CAPTION | WS_SYSMENU | DS_MODALFRAME | DS_3DLOOK |
        DS_SETFONT, NULL, FALSE, FALSE, FALSE, 0, 0, 0, 0);
    len += XMLTemplateDlgRowBegin(xmldata+len, xmlsize-len);
    len += XMLTemplateDlgColBegin(xmldata+len, xmlsize-len,
        0, 0, 50, 14, 4, 4, 4, 4,
        _T("center"), _T("center"), 0);
    len += XMLTemplateDlgControl(xmldata+len, xmlsize-len,
        _T("BUTTON"), _T("OK"),
        WS_CHILD | WS_VISIBLE | WS_TABSTOP,
        NULL, IDOK, TRUE, FALSE, FALSE, 0, 0, 1, 1);
    len += XMLTemplateDlgColEnd(xmldata+len, xmlsize-len);
    len += XMLTemplateDlgRowEnd(xmldata+len, xmlsize-len);
    len += XMLTemplateDlgEnd(xmldata+len, xmlsize-len);
    return len;
}
```

```
INT_PTR CALLBACK DlgProc(HWND hwndDlg, UINT uMsg, WPARAM wParam,
    LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_INITDIALOG:
        {
            return TRUE;
        }
        case WM_COMMAND:
        {
            int cmd = GET_WM_COMMAND_ID(wParam, lParam);
            switch (cmd)
            {
                case IDCANCEL:
                case IDABORT:
                case IDOK:
                {
                    EndDialog(hwndDlg, cmd);
                    break;
                }
            }
            return TRUE;
        }

    }
    return FALSE;
}

LRESULT DisplayDialog(HINSTANCE hInstance, HWND hwnd)
{
    LPTSTR xmldata = NULL;
    int xmlsize = 0;
    LRESULT lResult;
    int len;
    do {
        xmlsize += 4096;
        LPTSTR newxmldata = (LPTSTR)realloc(xmldata,
            xmlsize*sizeof(TCHAR));
        if (newxmldata)
        {
            xmldata = newxmldata;
        }
        else
        {
            if (xmldata) free(xmldata);
            return IDABORT;
        }
        len = GetDlgXMLTemplate(xmldata, xmlsize);
    } while (len==xmlsize-1);
    lResult = XMLDialogBox(hInstance, hwnd, DlgProc,
        xmldata, len);
    free(xmldata);
    return lResult;
}
```

4.8 Creating a property sheet

Property sheet can be created with the `XMLPropertySheetCreate` function. Here is an example of a simple property sheet:

```
int GetDlgXMLTemplate(TCHAR *xmldata, int xmlsize)
{
    int len = XMLTemplateDlgBegin(xmldata, xmlsize,
        _T("Sample"), _T("MS Sans Serif, 10"),
        WS_CHILD | WS_VISIBLE | DS_SETFONT,
        NULL, FALSE, FALSE, FALSE, 0, 0, 0, 0);

    len += XMLTemplateDlgRowBegin(xmldata+len, xmlsize-len);
    len += XMLTemplateDlgColBegin(xmldata+len, xmlsize-len,
        0, 0, 50, 14, 4, 4, 4, 4,
        _T("center"), _T("center"), 0);
    len += XMLTemplateDlgControl(xmldata+len, xmlsize-len,
        _T("BUTTON"), _T("Btn"), WS_CHILD | WS_VISIBLE |
        WS_TABSTOP, NULL, 100, TRUE, FALSE, FALSE, 0, 0, 1, 1);
    len += XMLTemplateDlgColEnd(xmldata+len, xmlsize-len);
    len += XMLTemplateDlgRowEnd(xmldata+len, xmlsize-len);

    len += XMLTemplateDlgEnd(xmldata+len, xmlsize-len);
    return len;
}

INT_PTR CALLBACK PageDlgProc(HWND hwndDlg, UINT uMsg,
    WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_INITDIALOG:
        {
            return TRUE;
        }
    }
    return FALSE;
}

void DiaplaySamplePropertyPage(HINSTANCE hInstance, HWND hwnd)
{
    XMLPROPERTYSHEET XMLps;
    ZeroMemory(&XMLps, sizeof(XMLPROPERTYSHEET));
    XMLps.dwFlags = XMLPS_DEFAULT |
        XMLPS_NOAPPLYNOW | XMLPS_NOBUYNOW;
    XMLps.hwndParent = hwnd;
    XMLps.hInstance = hInstance;
    XMLps.hIcon = (HICON)LoadImage(hInstance,
        MAKEINTRESOURCE(1), IMAGE_ICON, 32, 32, NULL);
    XMLps.hIconSm = (HICON)LoadImage(hInstance,
        MAKEINTRESOURCE(1), IMAGE_ICON, 16, 16, NULL);
    XMLps.pszCaption = g_strtbl.lpSamplePSCaption;
    XMLps.pszFontInfo = _T("MS Sans Serif, 10");
    XMLps.x = 10; XMLps.y = 10;
    XMLps.cx = 100; XMLps.cy = 100;
    XMLps.propText.pszOK = g_strtbl.lpOK;
    XMLps.propText.pszCancel = g_strtbl.lpCancel;
    XMLps.propText.pszClose = g_strtbl.lpClose;
    XMLps.propText.pszApplyNow = g_strtbl.lpApply;
    XMLps.propText.pszBuyNow = g_strtbl.lpBuyNow;
    XMLps.propText.pszHelp = g_strtbl.lpHelp;
    XMLps.propText.pszBack = g_strtbl.lpBack;
    XMLps.propText.pszNext = g_strtbl.lpNext;
    XMLps.propText.pszFinish = g_strtbl.lpFinish;
    HANDLE hXMLps = XMLPropertySheetCreate(&XMLps);
    if (hXMLps)
    {
        LPTSTR xmldata = NULL;
        int xmlsize = 0;
        int len;
        do {
            xmlsize += 4096;
            LPTSTR newxmldata = (LPTSTR)realloc(xmldata,
                xmlsize*sizeof(TCHAR));
            if (newxmldata)
            {
                xmldata = newxmldata;
            }
            else
            {
                if (xmldata)
                {
                    free(xmldata);
                    xmldata = NULL;
                }
                break;
            }
            len = GetDlgXMLTemplate(xmldata, xmlsize);
        } while (len==xmlsize-1);

        if (xmldata)
        {
            XMLPROPERTYSHEETPAGE XMLpsp;
            ZeroMemory(&XMLpsp, sizeof(XMLPROPERTYSHEETPAGE));
            XMLpsp.hInstance = hInstance;
            XMLpsp.lpXMLdata = xmldata;
            XMLpsp.dwXMLsize = xmlsize;
            XMLpsp.CheckControlSizeEx = NULL;
            XMLpsp.pszIcon = MAKEINTRESOURCE(1);
            XMLpsp.pfnDlgProc = PageDlgProc;
            XMLpsp.lParam = NULL;
            XMLPropertySheetAdd(hXMLps, &XMLpsp);
            free(xmldata);
        }

        XMLPropertySheetShow(hXMLps, 0);
        XMLPropertySheetFree(hXMLps);
    }
}
```

5 Building samples

To either build or run sample projects you need to put the following files to their folders:

```
applocx32a.dll:
  sample\Releasex32a
applocx32w.dll:
  sample\Releasex32w
applocx64a.dll:
  sample\Releasex64a
applocx64w.dll:
  sample\Releasex64w
applocx32a.lib:
  sample\
applocx32w.lib:
  sample\
applocx64a.lib:
  sample\
applocx64w.lib:
  sample\
```

Another sample project for AppLocalizer DLL is Polsedit – a freeware utility to modify user policies on a local or remote system: http://polsedit.southsoftware.com/.
Polsedit has been created as an example to AppLocalizer DLL and is available with source code from the link above. You can find compilation instruction in the archive with the source code of Polsedit.

Note: If some characters in the sample program does not display properly, then the default font on your system does not contain the proper glyphs for those characters. You can install additional language support in the Control Panel \ Regional and Language Options dialog box.

6 XML Formats

This section describes the data format for INI and XML language files, XML menu and XML dialog box templates. All INI and XML data can be encoded in ANSI or UNICODE (UTF16) format.

6.1 Language file format INI

Language file contains multiple sections. Each section must have the following form:

```
[section]
key=string
       .
       .
       .
```

Special characters in INI language file:

Character	*Meaning*
&	Ampersand (&) is placed in front of the letter that is used as access key for menu or dialog items. For example, to use F as the access key for a menu named File, use &File. If the F in the menu name is underlined you can open the menu by pressing ALT+F.

Character	***Meaning***
\t	TAB character. For menu items \t is used to display the shortcut key combination for the menu item with right align.
\n	Line break character. You can add extra line breaks where you want to make some strings multilined so they fit better in their area.
%%	%% will be replaced with some test when string is displayed.

INI language file example:

```
[Buttons]
OK=OK
Cancel=Cancel
Apply=&Apply Now
BuyNow=Bu&y Now
Help=&Help
Back=&Back
Next=&Next
Finish=&Finish
```

6.2 Language file format XML

XML language file format:

- `stringtable` – starts a stringtable
  - `section` – starts a section
    - `name` – (string) name of a section
  - `entry` - entry
    - `name` – (string) name of entry
    - `value` – (string) value

Special characters in XML language file:

Character	***Meaning***
&	Ampersand (&) is placed in front of the letter that is used as access key for menu or dialog items. For example, to use F as the access key for a menu named File, use &File. If the F in the menu name is underlined you can open the menu by pressing ALT+F.
\t	TAB character. For menu items \t is used to display the shortcut key combination for the menu item with right align.
\n	Line break character. You can add extra line breaks where you want to make some strings multilined so they fit better in their area.
%%	%% will be replaced with some test when string is displayed.

XML language file example:

```
<stringtable>
  <section name="Buttons">
    <entry name="OK" value="OK" />
    <entry name="Cancel" value="Cancel" />
    <entry name="Apply" value="&#38;Apply Now" />
    <entry name="BuyNow" value="Bu&#38;y Now" />
    <entry name="Help" value="&#38;Help" />
    <entry name="Back" value="&#38;Back" />
    <entry name="Next" value="&#38;Next" />
    <entry name="Finish" value="&#38;Finish" />
  </section>
</stringtable>
```

6.3 Menu XML format

XML Menu template format:

`menu` – starts menu or submenu
  - `title` – (string) submenu title
  - `flags` – (string) menu item flags
  - `id` – (unsigned int) menu item id

`item`
  - `title` – menu item title
  - `flags` – menu item flags
  - `id` – menu item id

The flags parameter can be any combination of the following values (space delimited):

`checked`	`MF_CHECKED` Places a check mark next to the item.
`disabled`	`MF_DISABLED` Disables the menu item so that it cannot be selected, but this flag does not gray it.
`enabled`	`MF_ENABLED` Enables the menu item so that it can be selected and restores it from its grayed state.
`grayed`	`MF_GRAYED` Disables the menu item and grays it so that it cannot be selected.
`menubarbreak`	`MF_MENUBARBREAK` Functions the same as the `MF_MENUBREAK` flag for a menu bar. For a drop-down menu, submenu, or shortcut menu, the new column is separated from the old column by a vertical line.
`menubreak`	`MF_MENUBREAK` Places the item on a new line (for menu bars) or in a new column (for a drop-down menu, submenu, or shortcut menu) without separating columns.
`ownerdraw`	`MF_OWNERDRAW` Specifies that the item is an owner-drawn item. Before the menu is displayed for the first time, the window that owns the menu receives a `WM_MEASUREITEM` message to retrieve the width and height of the menu item. The `WM_DRAWITEM` message is then sent to the window procedure of the owner window whenever the appearance of the menu item must be updated.
`separator`	`MF_SEPARATOR` Draws a horizontal dividing line. This flag is used only in a drop-down menu, submenu, or shortcut menu. The line cannot be grayed, disabled, or highlighted.
`unchecked`	`MF_UNCHECKED` Does not place a check mark next to the item.

The following XML Menu template example is from the sample project:

```
<menu>
  <menu title="&#38;File" flags="enabled">
    <item title="E&#38;xit\tAlt-F4" flags="enabled" id="200" />
  </menu>
  <menu title="&#38;Sample" flags="enabled">
```

```
    <item title="Sample &#38;Property Sheet..." flags="enabled"
      id="201" />
    <item title="Sample &#38;Wizard..." flags="enabled"
      id="202" />  </menu>
  <menu title="&#38;Options" flags="enabled">
    <item title="&#38;Preferences..." flags="enabled"
      id="203" />
    <item flags="separator" />
    <menu title="&#38;Language" flags="enabled">
      <item title="Chinese Simplified" flags="enabled"
        id="217" />
      <item title="English US" flags="checked" id="223" />
      <item title="French" flags="enabled" id="229" />
      <item title="Italian" flags="enabled" id="239" />
      <item title="Russian" flags="enabled" id="260" />
    </menu>
  </menu>
  <menu title="&#38;Help" flags="enabled">
    <item title="&#38;Contents" flags="enabled" id="204" />
    <item title="&#38;About..." flags="enabled" id="205" />
  </menu>
</menu>
```

6.4 Dialog XML format

XML Dialog template format:

- `dialog` - starts a dialog
  - `title` – (string) title of a dialog box
  - `font` - (string) dialog font
  - `style` – (unsigned int) dialog style constant (format: "0xHEX" "#HEX" "DEC")
  - `exstyle` - (unsigned int) dialog extended style constant (format: "0xHEX" "#HEX" "DEC")
  - `scalable` - (boolean) scalable dialog flag (format: "1" "0")
  - `noscroll` - (boolean) hide scroll bars for scalable dialog (format: "1" "0")
  - `watchinput` - (boolean) send `DM_XMLDLGWATCHINPUT` message to dialog procedure to watch user input (format: "1" "0")
  - `left` - (int) x-coordinate of the upper-left corner of the dialog box, in dialog box units
  - `top` - (int) y-coordinate of the upper-left corner of the dialog box, in dialog box units
  - `width` - (int) width of the dialog box, in dialog box units, ignored for non-scalable dialog
  - `height` - (int) height of the dialog box, in dialog box units, ignored for non-scalable dialog
- `group` - starts a group box
  - `title` - (string) title of a group box
  - `id` - (int) control identifier for a group box
- `panel` - starts an enclosed table (panel)
- `tr` - starts cells row
- `td` - starts a new cell in the row
  - `colspan` - (unsigned int) extend column across multiple other columns
  - `rowspan` - (unsigned int) extend row across multiple other rows
  - `minwidth` - (unsigned int) minimal width of the cell, in dialog box units
  - `minheight` - (unsigned int) minimal height of the cell, in dialog box units
  - `leftmargin` - (unsigned int) left margin in the cell, in dialog box units
  - `topmargin` - (unsigned int) top margin in the cell, in dialog box units
  - `rightmargin` - (unsigned int) right margin in the cell, in dialog box units
  - `bottommargin` - (unsigned int) bottom margin in the cell, in dialog box units
  - `align` - (string) alignment inside the cell (format: "LEFT" " CENTER" "RIGHT")

  - `valign` - (string) alignment inside the cell (format: "TOP" " CENTER" "BOTTOM")
  - `storerect` - (unsigned int) store cell's bounding rectangle to the structure, which is available by `DM_XMLDLGGETINFO` message

- `control` - starts a control inside a cell
  - `type` - (string) type of a control (window class name)
  - `title` - (string) title of a control (window text)
  - `style` - (unsigned int) control style constant (format: "0xHEX" "#HEX" "DEC")
  - `exstyle` - (unsigned int) control extended style constant (format: "0xHEX" "#HEX" "DEC")
  - `id` - (int) control identifier
  - `forcesize` - (boolean) set control's size to full size of its cell even if control's original size is smaller (format: "1" "0")
  - `forcewidth` - (boolean) set control's width to full width of its cell even if control's original size is smaller (format: "1" "0")
  - `forceheight` - (boolean) set control's height to full height of its cell even if control's original size is smaller (format: "1" "0")
  - `addwidth` - (int) additional control's width, in dialog box units; for example, can be used to provide an extra space to display an elevated icon for a button.
  - `addheight` - (int) additional control's height, in dialog box unit; for example, can be used to specify height of drop-down list for a combo-box.
  - `relwidth` - (double) relative value for control's size
  - `relheight` - (double) relative value for control's size

The argument values must be specified in the defined format. For text format there are the following restrictions:

1) quote character is disallowed: use &#x22; instead
2) format characters are allowed: \n \r \t \s

Dialog font format: "facename[, font size[, charset[BOLD ITALIC UNDERLINE]]]"
where charset can be one of the following values: ANSI, DEFAULT, SYMBOL, SHIFTJIS, GB2312, HANGEUL, CHINESEBIG5, OEM, JOHAB, HEBREW, ARABIC, GREEK, TURKISH, THAI, EASTEUROPE, RUSSIAN, MAC, BALTIC.

The following XML dialog template example is the Preferences dialog from the sample project:

```
<dialog title="Preferences" font="MS Sans Serif, 10"
  style="0x90c800c4" watchinput="1">
  <tr>
    <td colspan="3" leftmargin="8" topmargin="4" rightmargin="8"
      bottommargin="4" align="left" valign="top">
      <group title="Language" id="101">
        <tr>
          <td leftmargin="8" topmargin="4" rightmargin="8"
            bottommargin="4" align="center" valign="bottom">
            <control type="STATIC" title="Select &#38;language:"
              style="0x50000000" id="102" />
          </td>
        </tr>
        <tr>
          <td minwidth="70" minheight="10" leftmargin="8"
            topmargin="4" rightmargin="8" bottommargin="8"
            align="center" valign="top">
            <control type="COMBOBOX" style="0x50210803" id="103"
              forcewidth="1" addheight="60" />
          </td>
        </tr>
      </group>
    </td>
  </tr>
  <tr>
    <td minwidth="50" minheight="14" leftmargin="4"
      topmargin="4" rightmargin="4" bottommargin="4"
      align="left" valign="top">
    </td>
    <td minwidth="50" minheight="14" leftmargin="4"
      topmargin="4" rightmargin="4" bottommargin="4"
      align="left" valign="top">
      <control type="BUTTON" title="OK" style="0x50010000"
        id="1" forcesize="1" />
    </td>
    <td minwidth="50" minheight="14" leftmargin="4"
      topmargin="4" rightmargin="4" bottommargin="4"
      align="left" valign="top">
      <control type="BUTTON" title="Cancel" style="0x50010000"
        id="2" forcesize="1" />
    </td>
  </tr>
  <tr>
    <td colspan="3" minheight="14" align="left" valign="top">
      <control type="msctls_statusbar32" style="0x50000006"
        id="100" forcesize="1" />
    </td>
  </tr>
</dialog>
```

7 XML Dialog editor

File management commands:
Open – Open an XML dialog template
Save – Save XML dialog template
Save As – Save XML dialog template with a new file name
Export – Export XML dialog template
Exit – Exit XML Dialog Editor

Edit commands:
Undo – Undo last edition
Redo – Redo the previously undone edition
Cell Properties – Edit Cell properties
Control Properties – Edit Control properties
Group/Panel Properties – Edit Group/Panel properties
Dialog Properties – Edit Dialog properties
Insert Group/Panel – Insert Group/Panel into the selected cell
Delete Group/Panel – Delete Group/Panel from the selected cell
Insert Row – Insert row
Append Row – Append row
Delete Row – Delete row
Insert Column – Insert column
Append Column – Append column

Delete Column – Delete column
Join Cell Left – Join selected sell with the cell left
Join Cell Above – Join selected sell with the cell above
Join Cell Right – Join selected sell with the cell right
Join Cell Below – Join selected sell with the cell below

Options:
Language – Language management commands

View management commands:
Plain XML – Show XML dialog template as plain XML
Source code – Show XML dialog template as source code
Grid – Show grid lines around cells of XML dialog template

8 Header files reference

Include files:

`xmllng.h` - Multilanguage string table

`xmlmnu.h` - Multilanguage XML Menus

`xmldlg.h` - Multilanguage XML Dialogs

`xmlps.h` - Multilanguage XML Property Sheets

Note: All structures in header files are 4-byte aligned.

8.1 Multilanguage string table

Include file: `xmllng.h`.

8.1.1 Language constants

Language constants

```
enum {
    MULTILANG_AFRIKAANS,           /*  0, Afrikaans */
    MULTILANG_ALBANIAN,            /*  1, Albanian */
    MULTILANG_ARABIC,              /*  2, Arabic */
    MULTILANG_ARMENIAN,            /*  3, Armenian */
    MULTILANG_ASSAMESE,            /*  4, Assamese */
    MULTILANG_AZERI,               /*  5, Azeri */
    MULTILANG_BASQUE,              /*  6, Basque */
    MULTILANG_BELARUSIAN,          /*  7, Belarusian */
    MULTILANG_BENGALI,             /*  8, Bengali */
    MULTILANG_BULGARIAN,           /*  9, Bulgarian */
    MULTILANG_CATALAN,             /* 10, Catalan */
    MULTILANG_CHINESE_SIMPLIFIED,  /* 11, Chinese Simplified */
    MULTILANG_CHINESE_TRADITIONAL, /* 12, Chinese Traditional */
    MULTILANG_CROATIAN,            /* 13, Croatian */
    MULTILANG_CZECH,               /* 14, Czech */
    MULTILANG_DANISH,              /* 15, Danish */
    MULTILANG_DUTCH,               /* 16, Dutch */
    MULTILANG_ENGLISH_US,          /* 17, English US */
    MULTILANG_ENGLISH_UK,          /* 18, English UK */
    MULTILANG_ESTONIAN,            /* 19, Estonian */
    MULTILANG_FAEROESE,            /* 20, Faeroese */
    MULTILANG_FARSI,               /* 21, Farsi */
    MULTILANG_FINNISH,             /* 22, Finnish */
    MULTILANG_FRENCH,              /* 23, French */
    MULTILANG_GEORGIAN,            /* 24, Georgian */
    MULTILANG_GERMAN,              /* 25, German */
    MULTILANG_GREEK,               /* 26, Greek */
    MULTILANG_GUJARATI,            /* 27, Gujarati */
    MULTILANG_HEBREW,              /* 28, Hebrew */
    MULTILANG_HINDI,               /* 29, Hindi */
    MULTILANG_HUNGARIAN,           /* 30, Hungarian */
    MULTILANG_ICELANDIC,           /* 31, Icelandic */
    MULTILANG_INDONESIAN,          /* 32, Indonesian */
    MULTILANG_ITALIAN,             /* 33, Italian */
    MULTILANG_JAPANESE,            /* 34, Japanese */
    MULTILANG_KANNADA,             /* 35, Kannada */
    MULTILANG_KASHMIRI,            /* 36, Kashmiri */
    MULTILANG_KAZAK,               /* 37, Kazak */
    MULTILANG_KONKANI,             /* 38, Konkani */
    MULTILANG_KOREAN,              /* 39, Korean */
    MULTILANG_LATVIAN,             /* 40, Latvian */
    MULTILANG_LITHUANIAN,          /* 41, Lithuanian */
    MULTILANG_MACEDONIAN,          /* 42, Macedonian */
    MULTILANG_MALAY,               /* 43, Malay */
    MULTILANG_MALAYALAM,           /* 44, Malayalam */
    MULTILANG_MANIPURI,            /* 45, Manipuri */
    MULTILANG_MARATHI,             /* 46, Marathi */
    MULTILANG_NEPALI,              /* 47, Nepali */
    MULTILANG_NORWEGIAN,           /* 48, Norwegian */
    MULTILANG_ORIYA,               /* 49, Oriya */
    MULTILANG_POLISH,              /* 50, Polish */
    MULTILANG_PORTUGUESE,          /* 51, Portuguese */
    MULTILANG_PUNJABI,             /* 52, Punjabi */
    MULTILANG_ROMANIAN,            /* 53, Romanian */
    MULTILANG_RUSSIAN,             /* 54, Russian */
    MULTILANG_SANSKRIT,            /* 55, Sanskrit */
    MULTILANG_SERBIAN,             /* 56, Serbian */
    MULTILANG_SINDHI,              /* 57, Sindhi */
    MULTILANG_SLOVAK,              /* 58, Slovak */
    MULTILANG_SLOVENIAN,           /* 59, Slovenian */
    MULTILANG_SPANISH,             /* 60, Spanish */
    MULTILANG_SWAHILI,             /* 61, Swahili */
    MULTILANG_SWEDISH,             /* 62, Swedish */
    MULTILANG_TAMIL,               /* 63, Tamil */
    MULTILANG_TATAR,               /* 64, Tatar */
    MULTILANG_TELUGU,              /* 65, Telugu */
    MULTILANG_THAI,                /* 66, Thai */
    MULTILANG_TURKISH,             /* 67, Turkish */
    MULTILANG_UKRAINIAN,           /* 68, Ukrainian */
    MULTILANG_URDU,                /* 69, Urdu */
    MULTILANG_UZBEK,               /* 70, Uzbek */
    MULTILANG_VIETNAMESE,          /* 71, Vietnamese */
    MULTILANG_MAXLANG              /* 72, Number of
                                      supported languages. */
};
```

8.1.2 XMLLangSetKey

SUMARY:
Sets registration key.

ARGUMENTS:
`lpszRegistrationKey` – pointer to a string with your registration key.

DECLARATION:

```
APPLOC_EXTERN void __stdcall XMLLangSetKey(
    TCHAR *lpszRegistrationKey);
```

RETURN VALUE:
This function has no return value.

8.1.3 XMLLangGetVersion

SUMMARY:
Retrieves current version of the library.

DECLARATION:

```
APPLOC_EXTERN DWORD __stdcall XMLLangGetVersion(void);
```

RETURN VALUE:
`MAKELONG(minor, major).`

8.1.4 XMLLangGetFileName

SUMMARY:
Retrieves language name from a `MULTILANG_` constant.

DECLARATION:

```
APPLOC_EXTERN bool __stdcall XMLLangGetFileName(int lang,
    TCHAR *lpszLangFile);
```

RETURN VALUE:
Returns `TRUE` if `lpszLangFile` buffer is filled with language name.

EXAMPLE:

```
TCHAR szLangFile[50];
bool err = GetLangFileName(MULTILANG_ENGLISH_US, szLangFile);
```

8.1.5 XMLLangId2Multilang

SUMMARY:
Converts a Windows LangId to a `MULTILANG_` constant.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLLangId2Multilang(
    unsigned int LangId);
```

RETURN VALUE:
Returns a `MULTILANG_` constant corresponding to the `LangId` value.
EXAMPLE:

```
// get the user default language identifier.
int MultilangId = LangId2Multilang(GetUserDefaultLangID());
```

8.1.6 XMLTemplateString

SUMMARY:
Writes formatted data to a string.

DECLARATION:
```
APPLOC_EXTERN int __cdecl XMLTemplateString(TCHAR *buffer,
    int count, TCHAR *format, ...);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
This function acts like the sprintf function. The only differences are:

1) All the strings specified as %s and %S will be prepared for XML format (for example, quote character (") will be replaced to "&#x22;" and so on).
2) To leave strings not XML formatted use the %su and %SU modifiers.
3) If the formatted string does not fit to the buffer, the function returns count-1.

8.1.7 XMLTemplateStringPtr

SUMMARY:
Writes formatted data to a string using a pointer to a list of arguments.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateStringPtr(TCHAR *buffer,
    int count, TCHAR *format, va_list argptr);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
See Remarks section of the `XMLTemplateString` function.

8.1.8 XMLLangLoadStrings

SUMMARY:
Loads string table from a language file.

ARGUMENTS:
`hInstance` – Handle to resource module.
`lpLoadStrings` – Pointer to array of `LOADSTRINGS` structures.
`nLoadStrings` – Number of items in the `lpLoadStrings`.
`lpLoadLangFile` – Pointer to a `LOADLANGFILESTRUCT` structure. See Remarks section for more information.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLLangLoadStrings(
    HINSTANCE hInstance, LPLOADSTRINGS lpLoadStrings,
    int nLoadStrings, LPLOADLANGFILESTRUCT lpLoadLangFile);
```

RETURN VALUE:
Returns the number of loaded strings.

REMARKS:

This function fills `lpszValue` field of items in the array of `LOADSTRINGS` structures. The `LOADSTRINGS` structure is declared as follows:

```
typedef struct tagLOADSTRINGS {
    TCHAR *lpszSection;     // the name of the section
                            // in the language file
    TCHAR *lpszKeyName;     // the name of the key whose
                            // value is to be retrieved
    TCHAR *lpszValidate;    // the validation string
    TCHAR **lpszValue;      // pointer to the retrieved
                            // and validated string
    LPARAM lParam;          // application-defined value
} LOADSTRINGS, *LPLOADSTRINGS;
```

Language file can be a file, a resource object or a memory object. Type of the language file and its format is defined in the `LOADLANGFILESTRUCT` structure:

```
typedef enum {
    LANGFILETYPE_INI,
    LANGFILETYPE_XML
} LANGFILE_TYPE;

typedef enum {
    LANGFILESOURCE_NONE,
    LANGFILESOURCE_MEM,
    LANGFILESOURCE_RES,
    LANGFILESOURCE_FILE
} LANGFILE_SOURCE;

typedef struct tagLOADLANGFILESTRUCT {
    TCHAR *lpFile;
    LANGFILE_SOURCE FileSource;
    LANGFILE_TYPE FileType;
    union {
        DWORD cbFile;
        TCHAR *lpResType;
    } FileExtra;
    TCHAR *lpDefFile;
    LANGFILE_SOURCE DefFileSource;
    LANGFILE_TYPE DefFileType;
    union {
        DWORD cbDefFile;
        TCHAR *lpDefResType;
    } DefFileExtra;
} LOADLANGFILESTRUCT, FAR *LPLOADLANGFILESTRUCT;
```

Actually, `XMLLangLoadStrings` loads strings from two files. First it loads strings from a file specified in the `lpFile` parameter of the `LOADLANGFILESTRUCT` structure, then if not all strings are loaded, it tries to load strings which are not loaded in the first step from a file specified in the `lpDefFile` parameter of the `LOADLANGFILESTRUCT` structure.

To get extended error information, call the `GetLastError()` function.

8.1.9 XMLLangFreeStrings

SUMMARY:
Frees string table loaded by `XMLLangLoadStrings`.

ARGUMENTS:
`lpLoadStrings` – Pointer to array of `LOADSTRINGS` structures.
`nLoadStrings` – Number of items in the `lpLoadStrings`.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLLangFreeStrings(
    LPLOADSTRINGS lpLoadStrings, int nLoadStrings);
```

RETURN VALUE:
Returns the number of freed strings.

REMARKS:
`XMLLangFreeStrings` must be called when program terminating and before calling `XMLLangLoadStrings` to reload the string table.

8.1.10 XMLLangDefMultilang

SUMMARY:
Executes operations which are useful to handle the list of supported languages.

ARGUMENTS:
`LangList` – Array of supported languages.
`opcode` – Operation code.
`op` – A `MULTILANG_` constant.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLLangDefMultilang(
    MULTILANGAVAILABLE LangList, int opcode, int op);
```

RETURN VALUE:
If `opcode` is `MULTILANG_CHECKMULTILANG`, returns `TRUE`, if the language specified by `op` is in the list specified by `LangList`, in all other cases returns `FALSE`.

REMARKS:
`MULTILANGAVAILABLE` is declared as follows:

```
typedef unsigned char MULTILANGAVAILABLE[16];
```

This function supports the following operations:

`MULTILANG_ADDMULTILANG`	Adds the language specified by `op` to the list specified by `LangList`.
`MULTILANG_DELETEMULTILANG`	Removes the language specified by `op` `from` the list specified by `LangList`.
`MULTILANG_CHECKMULTILANG`	Returns `TRUE`, if the language specified by `op` is in the list specified by `LangList`, otherwise returns `FALSE`.

8.2 Multilanguage XML Menus

Include file: `xmlmnu.h`.

8.2.1 XMLTemplateMenuBegin

SUMMARY:
Writes opening menu tag to the XML menu template.

ARGUMENTS:
`buffer` – XML menu template buffer.
`count` – size of `buffer` in characters.
`szTitle` – Menu title.
`szFlags` – Menu item flags.
`dwId` – Menu item ID.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateMenuBegin(TCHAR *buffer,
    int count, LPTSTR szTitle, LPTSTR szFlags, DWORD dwId);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateMenuBegin` output:

`int len = XMLTemplateMenuBegin(buffer, count, _T("&File"), _T("enabled"), 0);`	`<menu title="&x22;File" flags="enabled">`

8.2.2 XMLTemplateMenuItem

SUMMARY:
Writes menu item tag to the XML menu template.

ARGUMENTS:
`buffer` – XML menu template buffer.
`count` – size of `buffer` in characters.
`szTitle` – Menu item title.
`szFlags` – Menu item flags.
`dwId` – Menu item ID.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateMenuItem(TCHAR *buffer,
    int count, LPTSTR szTitle, LPTSTR szFlags, DWORD dwId);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateMenuItem` output:

`int len = XMLTemplateMenuItem(buffer, count, _T("&Check"), _T("enabled checked"), 100);`	`<item title="&x22;Check" flags="enabled checked" id="100" />`

8.2.3 XMLTemplateMenuEnd

SUMMARY:
Writes menu closing tag to the XML menu template.

ARGUMENTS:
`buffer` – XML menu template buffer.
`count` – size of `buffer` in characters.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateMenuEnd(TCHAR *buffer,
    int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateMenuEnd` output:

`int len = XMLTemplateMenuEnd(buffer, count);`	`</menu>`

8.2.4 XMLCreateMenu

SUMMARY:
Creates menu from the XML menu template.

ARGUMENTS:
`xmldata` – XML menu template.
`xmlsize` – XML menu template size in characters.
`pdwXMLMenuError` – XML menu template parsing error code. This argument can be NULL.

DECLARATION:
```
APPLOC_EXTERN HMENU __stdcall XMLCreateMenu(LPCTSTR xmldata,
    unsigned int xmlsize, XMLMNUPARSE_E *pdwXMLMenuError);
```

RETURN VALUE:
If menu successfully created, returns handle to the created menu, otherwise returns `NULL`.

REMARKS:
XML menu template parsing error code can be one of the following values:

```
typedef enum {
    XMLMNUPARSE_NOERR = 0, // no error
    XMLMNUPARSE_UNCLOSED,  // unclosed tag found
} XMLMNUPARSE_E;
```

8.3 Multilanguage XML Dialogs

Include file: `xmldlg.h`.

8.3.1 DM_XMLDLGGETINFO

SUMMARY:
An application sends `DM_XMLDLGGETINFO` message to retrieve XML dialog box information. To send this message, call the `SendMessage` function with the following parameters:

```
SendMessage(
    (HWND) hWnd,        // handle to destination window
    DM_XMLDLGGETINFO,   // message to send
    (WPARAM) wParam,    // not used, must be zero
    (LPARAM) lParam     // pointer to GETXMLDLGINFO structure
);
```

ARGUMENTS:
`wParam` – This parameter is not used.
`lParam` – Pointer to `GETXMLDLGINFO` structure.

RETURN VALUE:
Returns `TRUE` on success, otherwise returns `FALSE`.

REMARKS:
`GETXMLDLGINFO` structure contains information about of the dialog box dimensions.

```
typedef struct tagGETXMLDLGINFO {
    int nPanelLeft;           // the position of the horizontal
                              // scroll box if the dialog
                              // has scroll bars
    int nPanelTop;            // the position of the vertical
                              // scroll box if the dialog
                              // has scroll bars
    unsigned int nPanelWidth;  // the width, not scaled, of the
                              // client area of the dialog
    unsigned int nPanelHeight; // the height, not scaled, of the
                              // client area of the dialog
    XMLDLGPARSE_E parseerror;  // the last error in parsing of
                              // XML data
    unsigned int nPanelRects;  // the number of items in the
                              // lpPanelRects
    XMLRECT *lpPanelRects;     // the list of XMLRECT structures
                              // for all cell with storerect
} GETXMLDLGINFO, *LPGETXMLDLGINFO;
```

`XMLDLGPARSE_E` can be one of the following values:

```
typedef enum {
    XMLDLGPARSE_NOERR = 0, // no error
    XMLDLGPARSE_UNEXPDLG,  // unexpected "dialog" tag
    XMLDLGPARSE_UNCLOSED,  // unclosed tag found
} XMLDLGPARSE_E;
```

The `XMLRECT` structure defines the coordinates of the upper-left and lower-right corners of a rectangle.

```
typedef struct tagXMLRECT {
    int top;
    int left;
    int right;
    int bottom;
} XMLRECT, *LPXMLRECT;
```

8.3.2 DM_XMLDLGSETTEMPLATE

SUMMARY:

An application sends `DM_XMLDLGSETTEMPLATE` message to change the XML dialog template. To send this message, call the SendMessage function with the following parameters:

```
SendMessage(
    (HWND) hWnd,          // handle to destination window
    DM_XMLDLGSETTEMPLATE, // message to send
    (WPARAM) wParam,      // XML dialog template (LPCTSTR)
    (LPARAM) lParam       // XML dialog template size in TCHARs
);
```

ARGUMENTS:

`wParam` – Pointer to the XML dialog template.
`lParam` – Size of the XML dialog template, in characters.

RETURN VALUE:

This message has no return value.

REMARKS:

This message is used to change XML template for a dialog that is previously created with another template. It can be useful when you need to change language of the already created XML dialog box. In order to avoid any conflicts it is recommended to set different id value to all controls and groups in the XML dialog template.

8.3.3 DM_XMLDLGWATCHINPUT

SUMMARY:

The `DM_XMLDLGWATCHINPUT` message is sent to the dialog procedure when user moves mouse over some dialog control or when some dialog control receives keyboard focus. A dialog box receives this message through its dialog procedure.

```
SendMessage(
    (HWND) hWnd,         // handle to destination window
    DM_XMLDLGWATCHINPUT, // message to send
    (WPARAM) wParam,     // handle to control (HWND)
    (LPARAM) lParam      // notification code (BOOL)
);
```

ARGUMENTS:

`wParam` – Handle to the control.
`lParam` – Notification code (`TRUE` – mouse cursor is over the control, `FALSE` – the control has received keyboard focus).

RETURN VALUE:

This message has no return value.

REMARKS:
This message is sent only when "watchinput" is enabled in the XML dialog template.

8.3.4 XMLGetBorderSize

SUMMARY:
Fills `cx` and `cy` with width and height of the window border (detected from style attributes and `GetSystemMetrics`).

ARGUMENTS:
`dwStyle` – Window style.
`dwExStyle` – Window extended style.
`cx` – Receives width of the window border.
`cy` – Receives height of the window border.

DECLARATION:
```
APPLOC_EXTERN void __stdcall XMLGetBorderSize(DWORD dwStyle,
    DWORD dwExStyle, unsigned int *cx, unsigned int *cy);
```

RETURN VALUE:
This function has no return value.

8.3.5 XMLMapDialogRect

SUMMARY:
The `XMLMapDialogRect` function converts (maps) the specified dialog box units to screen units (pixels). The function replaces the coordinates in the specified `RECT` structure with the converted coordinates, which allows the structure to be used to create a dialog box or position a control within a dialog box.

ARGUMENTS:
`hDlg` – Handle to a dialog box.
`lpRect` – Pointer to a `RECT` structure that contains the dialog box coordinates to be converted.

DECLARATION:
```
APPLOC_EXTERN BOOL __stdcall XMLMapDialogRect(HWND hDlg,
    LPRECT lpRect);
```

RETURN VALUE:
Returns `TRUE` on success, otherwise returns `FALSE`.

REMARKS:
This function is used to replace the standard `MapDialogRect` function, because the standard function always uses the default system font.

8.3.6 XMLTemplateDlgBegin

SUMMARY:
Writes opening dialog tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.
`szTitle` – Dialog box title.
`szFontInfo` – Dialog box font.

`dwStyle` – Dialog box style.
`dwExStyle` – Dialog box extended style.
`fScalable` – "scalable" flag.
`fNoScroll` – "noscroll" flag.
`fWatchInput` – "watchinput" flag.
`left` – Specified the x-coordinate, in dialog box units, of the upper-left corner of the dialog box.
`top` – Specified the y-coordinate, in dialog box units, of the lower-right corner of the dialog box.
`width` – Specified the width, in dialog box units, of the dialog box.
height – Specified the height, in dialog box units, of the dialog box.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgBegin(TCHAR *buffer,
    int count, LPTSTR szTitle, LPTSTR szFontInfo, DWORD dwStyle,
    DWORD dwExStyle, BOOL fScalable, BOOL fNoScroll,
    BOOL fWatchInput, int left, int top, int width, int height);
```

RETURN VALUE:

Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:

Here is an example of `XMLTemplateDlgBegin` output:

`int len = XMLTemplateDlgBegin(buffer, count, _T("Sample dialog box"), _T("MS Sans Serif, 10"), WS_DLGFRAME | WS_POPUP | WS_BORDER | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | DS_MODALFRAME | DS_3DLOOK | DS_SETFONT, NULL, FALSE, FALSE, FALSE, 0, 0, 0, 0);`	`<dialog title="Sample dialog box" font="MS Sans Serif, 10" style="0x90c800c4">`

8.3.7 XMLTemplateDlgEnd

SUMMARY:

Writes closing dialog tag to the XML dialog template.

ARGUMENTS:

`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgEnd(TCHAR *buffer,
    int count);
```

RETURN VALUE:

Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgEnd` output:

`int len = XMLTemplateDlgEnd(buffer, count);`	`</dialog>`

8.3.8 XMLTemplateDlgGroupBegin

SUMMARY:
Writes opening group tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.
`szTitle` – group title.
`id` – group id.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateDlgGroupBegin(
    TCHAR *buffer, int count, LPTSTR szTitle, int id);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgGroupBegin` output:

`int len = XMLTemplateDlgGroupBegin(buffer, count, _T("Group"), 100);`	`<group title="Group" id="100">`

8.3.9 XMLTemplateDlgGroupEnd

SUMMARY:
Writes closing group tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:
```
APPLOC_EXTERN int __stdcall XMLTemplateDlgGroupEnd(
    TCHAR *buffer, int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgGroupEnd` output:

`int len = XMLTemplateDlgGroupEnd(buffer, count);`	`</group>`

8.3.10 XMLTemplateDlgPanelBegin

SUMMARY:
Writes opening panel tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgPanelBegin(
    TCHAR *buffer, int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgPanelBegin` output:

`int len = XMLTemplateDlgPanelBegin(buffer, count);`	`<panel>`

8.3.11 XMLTemplateDlgPanelEnd

SUMMARY:
Writes closing panel tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgPanelEnd(
    TCHAR *buffer, int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgPanelEnd` output:

`int len = XMLTemplateDlgPanelEnd(buffer, count);`	`</panel>`

8.3.12 XMLTemplateDlgRowBegin

SUMMARY:
Writes opening tr tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgRowBegin(
    TCHAR *buffer, int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgRowBegin` output:

`int len = XMLTemplateDlgRowBegin(buffer, count);`	`<tr>`

8.3.13 *XMLTemplateDlgRowEnd*

SUMMARY:
Writes closing tr tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgRowEnd(TCHAR *buffer,
    int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgRowEnd` output:

`int len = XMLTemplateDlgRowEnd(buffer, count);`	`</tr>`

8.3.14 *XMLTemplateDlgColBegin*

SUMMARY:
Writes opening td tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.
`dwColspan` – number of joined columns.
`dwRowspan` – number of joined rows.
`dwMinWidth` – minimal width of the cell.
`dwMinHeight` – minimal height of the cell.
`dwLeftMargin` – left margin in the cell.
`dwTopMargin` – top margin in the cell.
`dwRightMargin` – right margin in the cell.
`dwBottomMargin` – bottom margin in the cell.
`szAlign` – horizontal alignment in the cell..
`szVAlign` – vertical alignment in the cell..
`dwStoreRect` – index of the stored rect.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgColBegin(
    TCHAR *buffer, int count, DWORD dwColspan, DWORD dwRowspan,
    DWORD dwMinWidth, DWORD dwMinHeight, DWORD dwLeftMargin,
    DWORD dwTopMargin, DWORD dwRightMargin,
    DWORD dwBottomMargin, LPTSTR szAlign,
    LPTSTR szVAlign, DWORD dwStoreRect);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgColBegin` output:

`XMLTemplateDlgColBegin(buffer, count, 0, 0, 50, 14, 4, 4, 4, 4, _T("left"), _T("top"), 0);`	`<td minwidth="50" minheight="14" leftmargin="4" topmargin="4" rightmargin="4" bottommargin="4" align="left" valign="top">`

8.3.15 XMLTemplateDlgColEnd

SUMMARY:
Writes closing td tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgColEnd(TCHAR *buffer,
int count);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgColEnd` output:

`int len = XMLTemplateDlgColEnd(buffer, count);`	`</td>`

8.3.16 XMLTemplateDlgControl

SUMMARY:
Writes control tag to the XML dialog template.

ARGUMENTS:
`buffer` – XML dialog template buffer.
`count` – size of `buffer` in characters.
`szType` – colntrol class name.
`szTitle` – control text.
`dwStyle` – control style.
`dwExStyle` – control extended style.
`id` – control identifier.

`fForceSize` – "forcesize" flag.
`fForceWidth` – "forcewidth" flag.
`fForceHeight` – "forceheight" flag.
`addwidth` – extra width.
`addheight` – extra height.
`relwidth` – relative width ratio.
`relheight` – relative height ratio.

DECLARATION:

```
APPLOC_EXTERN int __stdcall XMLTemplateDlgControl(
    TCHAR *buffer, int count, LPTSTR szType, LPTSTR szTitle,
    DWORD dwStyle, DWORD dwExStyle, int id, BOOL fForceSize,
    BOOL fForceWidth, BOOL fForceHeight, int addwidth,
    int addheight, double relwidth, double relheight);
```

RETURN VALUE:
Returns the number of characters stored in `buffer`, not counting the null terminating character.

REMARKS:
Here is an example of `XMLTemplateDlgColEnd` output:

`int len = XMLTemplateDlgControl(buffer, count, _T("BUTTON"), _T("OK"), WS_CHILD | WS_VISIBLE | WS_TABSTOP, NULL, IDOK, TRUE, FALSE, FALSE, 0, 0, 1, 1);`	`<control type="BUTTON" title="OK" style="0x50010000" forcesize="1" />`

8.3.17 XMLCreateDialogBox

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`dwInitParam` – Specifies the value to pass to the dialog box in the lParam parameter of the WM_INITDIALOG message.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.
`ProcessDialog` – process dialog flag. See Remark section for more information.
`CheckControlSizeEx` – callback function. See Remark section for more information.

DECLARATION:

```
APPLOC_EXTERN LRESULT __stdcall XMLCreateDialogBox(
    HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, LPARAM dwInitParam,
    TCHAR *lpXMLdata, DWORD dwXMLsize, BOOL ProcessDialog,
    CheckControlSizeExFn CheckControlSizeEx);
```

RETURN VALUE:
See Remarks section for the return value information.

REMARKS:
If `ProcessDialog` is `TRUE`, the function does not return control until the `lpDialogFunc` function terminates the modal dialog box by calling the `EndDialog` function. If the function succeeds, the return value is the `nResult` parameter in the call of the `EndDialog` function used to terminate the dialog box. If the function fails, the return value is -1.

If `ProcessDialog` is `FALSE`, the function creates a modeless dialog box. If the function succeeds, the return value is the handle to the dialog box window. If the function fails the return value is `NULL`.

Extra information of failure is available by `GetLastError()`:

`ERROR_SUCCESS`	The operation completed successfully.
`ERROR_NOT_ENOUGH_MEMORY`	Not enough storage is available.
`ERROR_REPARSE_TAG_INVALID`	Enclosed dialog found (`XMLDLGPARSE_UNEXPDLG`).
`ERROR_REPARSE_TAG_MISMATCH`	Tag parsing error (`XMLDLGPARSE_UNCLOSED`).
`ERROR_INVALID_REPARSE_DATA`	Unknown parsing error.
`ERROR_INVALID_HANDLE`	Invalid handle or function argument.

`CheckControlSizeEx` - pointer to the user-defined callback function. This function is used to determine size of dialog box controls.

```
// hwndParent -      [in] handle to the parent window
// DC -              [in] handle to the device
//                        context with font selected
// uID -             [in] child-window identifier
// lpClassName -     [in] pointer to the control class name
// lpCaption -       [in] pointer to the control text
// dwStyle -         [in] control style
// dwExStyle -       [in] extended style
// nMinWidth -       [in] minimal width, in pixels
// nMinHeight -      [in] minimal height, in pixels
// relwidth -        [in] relative width ratio
// relheight -       [in] relative height ratio
// pnWidth -         [out] minimal control width, in pixels
// pnHeight -        [out] minimal control height, in pixels
// pdwStyleUpdate -  [out] style update for the specified
//                         window class
typedef BOOL (__stdcall *CheckControlSizeExFn)(
    HWND hwndParent, HDC DC, UINT uID,
    LPCTSTR lpClassName, LPCTSTR lpCaption,
    DWORD dwStyle, DWORD dwExStyle,
    unsigned int nMinWidth, unsigned int nMinHeight,
    double relwidth, double relheight,
    unsigned int *pnWidth, unsigned int *pnHeight,
    LPDWORD pdwStyleUpdate);
```

8.3.18 XMLDialogBox

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.

`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.

DECLARATION:

```
int XMLDialogBox(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, TCHAR *lpXMLdata, DWORD dwXMLsize);
```

RETURN VALUE:

The function does not return control until the `lpDialogFunc` function terminates the modal dialog box by calling the `EndDialog` function. If the function succeeds, the return value is the `nResult` parameter in the call of the `EndDialog` function used to terminate the dialog box. If the function fails, the return value is -1.

8.3.19 XMLCreateDialog

SUMMARY:

Creates a dialog box from an XML dialog template.

ARGUMENTS:

`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.

DECLARATION:

```
HWND XMLCreateDialog(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, TCHAR *lpXMLdata, DWORD dwXMLsize);
```

RETURN VALUE:

The function creates a modeless dialog box. If the function succeeds, the return value is the handle to the dialog box window. If the function fails the return value is `NULL`.

8.3.20 XMLDialogBoxEx

SUMMARY:

Creates a dialog box from an XML dialog template.

ARGUMENTS:

`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.
`CheckControlSizeEx` – callback function. See Remarks section of the `XMLCreateDialogBox` function for more information.

DECLARATION:

```
int XMLDialogBoxEx(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, TCHAR *lpXMLdata, DWORD dwXMLsize,
    CheckControlSizeExFn CheckControlSizeEx);
```

RETURN VALUE:
The function does not return control until the `lpDialogFunc` function terminates the modal dialog box by calling the `EndDialog` function. If the function succeeds, the return value is the `nResult` parameter in the call of the `EndDialog` function used to terminate the dialog box. If the function fails, the return value is -1.

8.3.21 XMLCreateDialogEx

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.
`CheckControlSizeEx` – callback function. See Remarks section of the `XMLCreateDialogBox` function for more information.

DECLARATION:
```
HWND XMLCreateDialogEx(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, TCHAR *lpXMLdata, DWORD dwXMLsize,
    CheckControlSizeExFn CheckControlSizeEx);
```

RETURN VALUE:
The function creates a modeless dialog box. If the function succeeds, the return value is the handle to the dialog box. If the function fails the return value is `NULL`.

8.3.22 XMLDialogBoxParam

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`dwInitParam` – Specifies the value to pass to the dialog box in the `lParam` parameter of the `WM_INITDIALOG` message.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.

DECLARATION:
```
int XMLDialogBoxParam(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC  lpDialogFunc, LPARAM dwInitParam,
    TCHAR *lpXMLdata, DWORD dwXMLsize);
```

RETURN VALUE:
The function does not return control until the `lpDialogFunc` function terminates the modal dialog box by calling the `EndDialog` function. If the function succeeds, the return value is the `nResult` parameter in the call of the `EndDialog` function used to terminate the dialog box. If the function fails, the return value is -1.

8.3.23 XMLCreateDialogParam

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`dwInitParam` – Specifies the value to pass to the dialog box in the `lParam` parameter of the `WM_INITDIALOG` message.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.

DECLARATION:
```
HWND XMLCreateDialogParam(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, LPARAM dwInitParam,
    TCHAR *lpXMLdata, DWORD dwXMLsize);
```

RETURN VALUE:
The function creates a modeless dialog box. If the function succeeds, the return value is the handle to the dialog box. If the function fails the return value is `NULL`.

8.3.24 XMLDialogBoxParamEx

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`dwInitParam` – Specifies the value to pass to the dialog box in the `lParam` parameter of the `WM_INITDIALOG` message.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.
`CheckControlSizeEx` – callback function. See Remarks section of the `XMLCreateDialogBox` function for more information.

DECLARATION:
```
int XMLDialogBoxParamEx(HINSTANCE hInstance, HWND hWndParent,
    DLGPROC lpDialogFunc, LPARAM dwInitParam,
    TCHAR *lpXMLdata, DWORD dwXMLsize,
    CheckControlSizeExFn CheckControlSizeEx);
```

RETURN VALUE:
The function does not return control until the `lpDialogFunc` function terminates the modal dialog box by calling the `EndDialog` function. If the function succeeds, the return value is the `nResult` parameter in the call of the `EndDialog` function used to terminate the dialog box. If the function fails, the return value is -1.

8.3.25 XMLCreateDialogParamEx

SUMMARY:
Creates a dialog box from an XML dialog template.

ARGUMENTS:
`hInstance` – Handle to the module.
`hWndParent` – Handle to the window that owns the dialog box.
`lpDialogFunc` – Pointer to the dialog box procedure.
`dwInitParam` – Specifies the value to pass to the dialog box in the `lParam` parameter of the `WM_INITDIALOG` message.
`lpXMLdata` – XML dialog template.
`dwXMLsize` – size of XML dialog template in characters.
`CheckControlSizeEx` – callback function. See Remarks section of the `XMLCreateDialogBox` function for more information.

DECLARATION:
```
HWND XMLCreateDialogParamEx(HINSTANCE hInstance,
    HWND hWndParent, DLGPROC lpDialogFunc,
    LPARAM dwInitParam, TCHAR *lpXMLdata, DWORD dwXMLsize,
    CheckControlSizeExFn CheckControlSizeEx);
```

RETURN VALUE:
The function creates a modeless dialog box. If the function succeeds, the return value is the handle to the dialog box. If the function fails the return value is `NULL`.

8.4 Multilanguage XML Property Sheets

Include file: `xmlps.h`.

8.4.1 XMLPSN_SETACTIVE

SUMMARY:
Notifies a page that it is activated. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_SETACTIVE
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
This message has no return value.

REMARKS:
The `XMLPSN_SETACTIVE` notification message is sent when the page is visible. An application can use this notification to initialize data in the page.

8.4.2 XMLPSN_KILLACTIVE

SUMMARY:
Notifies a page that it is about to lose activation either because another page is being activated or the user clicked the OK button. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_KILLACTIVE
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
Returns `TRUE` to prevent the page from losing the activation, or `FALSE` to allow it.

REMARKS:
An application handles this notification to validate the information the user has entered.

To set a return value, the dialog box procedure for the page must call the `SetWindowLong` function with a `DWL_MSGRESULT` value set to the return value. The dialog box procedure must return `TRUE`.

If the dialog box procedure sets `DWL_MSGRESULT` to `TRUE`, it should display a message box to explain the problem to the user.

8.4.3 XMLPSN_APPLY

SUMMARY:
Sent to every page in the property sheet to indicate that the user has clicked the OK, Close, or Apply button and wants all changes to take effect. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_APPLY
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
Returns `FALSE` to indicate that the changes made to this page are valid and have been applied. If all pages return `FALSE`, the property sheet can be destroyed. To indicate that the changes made to this page are invalid and to prevent the property sheet from being destroyed, set `TRUE`.

To set the return value, the dialog box procedure for the page must call the `SetWindowLong` function with the `DWL_MSGRESULT` value, and the dialog box procedure must return `TRUE`.

REMARKS:
When the user clicks the OK, Apply, or Close button, the property sheet sends a `XMLPSN_KILLACTIVE` notification to the active page, giving it an opportunity to validate the user's changes. If the changes are valid, the property sheet sends a `XMLPSN_APPLY` notification message to each page, directing it to apply the new properties to the corresponding item.

Do not call the `EndDialog` function when processing this notification.

A modal property sheet is destroyed if the user clicks the OK button and every page returns the `FALSE` value in response to `XMLPSN_APPLY`. If any page returns `TRUE`, the Apply process is canceled immediately. Pages after the canceling page will not receive a `XMLPSN_APPLY` notification.

To receive this notification, a page must set the `DWL_MSGRESULT` value to `FALSE` in response to the `XMLPSN_KILLACTIVE` notification message.

8.4.4 XMLPSN_RESET

SUMMARY:
Notifies a page that the property sheet is about to be destroyed. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_RESET
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
This message has no return value.

REMARKS:
All changes made since the last `XMLPSN_APPLY` notification are canceled. An application can use this notification message as an opportunity to perform cleanup operations.

Do not call the `EndDialog` function when processing this notification.

8.4.5 XMLPSN_HELP

SUMMARY:
Notifies a page that the user has clicked the Help button. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_HELP
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
This message has no return value.

REMARKS:
An application should display Help information for the page.

8.4.6 XMLPSN_WIZBACK

SUMMARY:
Notifies a page that the user has clicked the Back button in a wizard. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_WIZBACK
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
Returns `TRUE` to prevent the wizard from changing pages. Returns `FALSE` to allow the wizard to go to the previous page.

REMARKS:
To set the return value, the dialog box procedure for the page must call the `SetWindowLong` function with the `DWL_MSGRESULT` value and return `TRUE`.

8.4.7 XMLPSN_WIZNEXT

SUMMARY:
Notifies a page that the user has clicked the Next button in a wizard. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_WIZNEXT
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
Returns `TRUE` to prevent the wizard from changing pages. Returns `FALSE` to allow the wizard to go to the next page.

REMARKS:
To set the return value, the dialog box procedure for the page must call the `SetWindowLong` function with the `DWL_MSGRESULT` value and return `TRUE`.

8.4.8 XMLPSN_WIZFINISH

SUMMARY:
Notifies a page that the user has clicked the Finish button in a wizard. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_WIZFINISH
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
Returns `TRUE` to prevent the wizard from finishing. Returns `FALSE` to allow the wizard to finish.

REMARKS:
To set the return value, the dialog box procedure for the page must use the `SetWindowLong` function with the `DWL_MSGRESULT` value, and the dialog box procedure must return `TRUE`.

8.4.9 XMLPSN_QUERYCANCEL

SUMMARY:
Indicates that the user has canceled the property sheet. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_QUERYCANCEL
    lpxmlpsqc = (LPXMLPSNQUERYCANCEL)lParam
```

ARGUMENTS:
`lpxmlpsqc` – Address of an `XMLPSNQUERYCANCEL` structure that contains information about this notification.

RETURN VALUE:
Returns `TRUE` to prevent the cancel operation, or `FALSE` to allow it.

REMARKS:
This message is typically sent when a user clicks the Cancel button. It is also sent when a user clicks the X button in the property sheet's upper right hand corner or presses the ESCAPE key. A property sheet page can handle this notification message to ask the user to verify the cancel operation.

To set a return value, the dialog box procedure for the page must call the `SetWindowLong` function with `DWL_MSGRESULT` set to the return value. The dialog box procedure must return `TRUE`.

REMARKS:
The `XMLPSNQUERYCANCEL` structure is declared as follows:

```
typedef struct tagXMLPSNQUERYCANCEL {
    NMHDR hdr;
    BOOL bButton; // TRUE if Cancel button is pressed.
                  // FALSE if something like [X] is pressed.
} XMLPSNQUERYCANCEL, FAR *LPXMLPSNQUERYCANCEL;
```

8.4.10 XMLPSN_BUYNOW

SUMMARY:
Notifies a page that the user has clicked the Buy Now button. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_BUYNOW
    lpnm = (LPNMHDR)lParam
```

ARGUMENTS:
`lpnm` – Address of an `NMHDR` structure that contains information about this notification.

RETURN VALUE:
This message has no return value.

REMARKS:
An application should display Buy Now information for the page.

8.4.11 XMLPSN_WATCHINPUT

SUMMARY:
The `DM_XMLDLGWATCHINPUT` message is sent to the dialog procedure when user moves mouse over some dialog control or when some dialog control receives keyboard focus. This notification message is sent in the form of a `WM_NOTIFY` message.

```
XMLPSN_WATCHINPUT
    lpxmlpswi = (LPXMLPSNWATCHINPUT)lParam
```

ARGUMENTS:

`lpxmlpswi` – Address of an `XMLPSNWATCHINPUT` structure that contains information about this notification.

RETURN VALUE:

This message has no return value.

REMARKS:

The `XMLPSNWATCHINPUT` structure is declared as follows:

```
typedef struct tagXMLPSNWATCHINPUT {
    NMHDR hdr;
    INT cursor;  // Index of a button.
} XMLPSNWATCHINPUT, FAR *LPXMLPSNWATCHINPUT;
```

The `cursor` can be one of the following values:

`XMLPSBTN_BACK`	Chooses the Back button
`XMLPSBTN_NEXT`	Chooses the Next button
`XMLPSBTN_FINISH`	Chooses the Finish button
`XMLPSBTN_OK`	Chooses the OK button
`XMLPSBTN_APPLYNOW`	Chooses the Apply Now button
`XMLPSBTN_CANCEL`	Chooses the Cancel button
`XMLPSBTN_HELP`	Chooses the Help button
`XMLPSBTN_BUYNOW`	Chooses the Buy Now button
`XMLPSBTN_CLOSE`	Chooses the Close button

8.4.12 XMLPSM_SETCURSELHWND

SUMMARY:

The `XMLPSM_SETCURSELHWND` message activates the specified page in a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetCurSelHwnd(hDlg, hwnd)` macro.

```
XMLPSM_SETCURSELHWND
    wParam = (WPARAM) (HWND) hwnd
    lParam = 0
```

ARGUMENTS:

`hwnd` – Handle to a page.

RETURN VALUE:

Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:

The window that is losing the activation receives the `XMLPSN_KILLACTIVE` notification message, and the window that is gaining the activation receives the `XMLPSN_SETACTIVE` notification message.

8.4.13 XMLPSM_SETCURSELID

SUMMARY:
The `XMLPSM_SETCURSELID` message activates the specified page in a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetCurSelId(hDlg, index)` macro.

```
XMLPSM_SETCURSELID
    wParam = (WPARAM) (int) index
    lParam = 0
```

ARGUMENTS:
`index` – Index of a page.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The window that is losing the activation receives the `XMLPSN_KILLACTIVE` notification message, and the window that is gaining the activation receives the `XMLPSN_SETACTIVE` notification message.

8.4.14 XMLPSM_ADDPAGE

SUMMARY:
Adds a new page to an existing property sheet. You can send this message explicitly or by using the `XMLPropSheet_AddPage(hDlg, index, lpXMLpsp)` macro.

```
XMLPSM_ADDPAGE
    wParam = (WPARAM) (int) index
    lParam = (LPARAM) (LPXMLPROPERTYSHEETPAGE)lpXMLpsp
```

ARGUMENTS:
`index` – Index of a page after which the new page should be inserted.
`lpXMLpsp` – Pointer to a `XMLPROPERTYSHEETPAGE` structure with information about the new page.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise. To get extended error information, call the `GetLastError()` function.

REMARKS:
The property sheet will be automatically resized to fit the new page.

The `XMLPROPERTYSHEETPAGE` structure is declared as follows:

```
typedef struct tagXMLPROPERTYSHEETPAGE {
    HINSTANCE hInstance;      // Handle to the instance from
                              // which to load the dialog box
                              // template, icon, or title
                              // string resource.
    LPCTSTR   lpXMLdata;      // Identifies dialog box XML
                              // template.
    DWORD     dwXMLsize;      // Size, in characters, of the
                              // dialog box XML template.
    CheckControlSizeExFn CheckControlSizeEx; // pointer to the
                              // user-defined callback function.
    HICON     hIcon;          // Handle of the icon to use as
                              // the small icon in the title
                              // bar of the property sheet
                              // dialog box.
    LPCTSTR   pszIcon;        // Icon resource to use as the
                              // small icon in the title bar of
                              // the property sheet dialog box.
    DLGPROC   pfnDlgProc;     // Pointer to the dialog box
                              // procedure for the page. The
                              // dialog box procedure must not
                              // call the EndDialog function.
    LPARAM    lParam;         // Application-defined data.
} XMLPROPERTYSHEETPAGE, FAR *LPXMLPROPERTYSHEETPAGE;
```

8.4.15 XMLPSM_REMOVEPAGE

SUMMARY:
Removes a page from a property sheet. You can send this message explicitly or by using the `XMLPropSheet_RemovePage(hDlg, index)` macro.

```
XMLPSM_REMOVEPAGE
    wParam = (WPARAM) (int) index
    lParam = 0
```

ARGUMENTS:
`index` – Index of a page after which the new page should be inserted.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The property sheet will be automatically resized to fit the new page.

Sending `XMLPSM_REMOVEPAGE` destroys the property sheet page that is being removed.

8.4.16 XMLPSM_CHANGED

SUMMARY:
Informs a property sheet that information in a page has changed. You can send this message explicitly or by using the `XMLPropSheet_Changed(hDlg, hwndPage)` macro.

```
XMLPSM_CHANGED
    wParam = (WPARAM) (HWND) hwndPage
    lParam = 0
```

ARGUMENTS:
`hwndPage` – Handle to a page that has changed.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The property sheet will enable the Apply button.

8.4.17 XMLPSM_CANCELTOCLOSE

SUMMARY:
Sent by an application when it has performed changes since the most recent `XMLPSN_APPLY` notification that cannot be canceled. You can send this message explicitly or by using the `XMLPropSheet_CancelToClose(hDlg)` macro.

```
XMLPSM_ CANCELTOCLOSE
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns TRUE if successful, or `FALSE` otherwise.

REMARKS:
`XMLPSM_CANCELTOCLOSE` disables the Cancel button and changes the text of the OK button to Close.

Use the `PostMessage` function to send this message.

8.4.18 XMLPSM_UNCHANGED

SUMMARY:
Informs a property sheet that information in a page has reverted to the previously saved state. You can send this message explicitly or by using the `XMLPropSheet_UnChanged(hDlg, hwndPage)` macro.

```
XMLPSM_UNCHANGED
    wParam = (WPARAM) (HWND) hwndPage
    lParam = 0
```

ARGUMENTS:
`hwndPage` – Handle to the page that has reverted to the previously saved state.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The property sheet disables the Apply button if no other pages have registered changes with the property sheet.

8.4.19 XMLPSM_APPLY

SUMMARY:
Simulates the selection of the Apply button, indicating that one or more pages has changed and the changes need to be validated and recorded. You can send this message explicitly or by using the `XMLPropSheet_Apply(hDlg)` macro.

```
XMLPSM_APPLY
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The property sheet sends the `XMLPSN_KILLACTIVE` notification message to the current page. If the current page returns `FALSE`, the property sheet sends the `XMLPSN_APPLY` notification message to all active pages.

8.4.20 XMLPSM_SETTITLE

SUMMARY:
Sets the title of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetTitle(hDlg, lpszText)` macro.

```
XMLPSM_SETTITLE
    wParam = 0
    lParam = (LPARAM) (LPCTSTR) lpszText
```

ARGUMENTS:
`lpszText` – Pointer to a buffer that contains the title string.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

8.4.21 XMLPSM_SETWIZBUTTONS

SUMMARY:
Enables or disables the Back, Next and Finish buttons in a wizard. You can send this message explicitly or by using the `XMLPropSheet_SetWizButtons(hDlg, dwFlags)` macro.

```
XMLPSM_SETWIZBUTTONS
    wParam = 0
    lParam = (LPARAM) (DWORD) dwFlags
```

ARGUMENTS:
`dwFlags` – Pointer to a buffer that contains the title string.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
Wizards display either three or four buttons below each page. This message is used to specify which buttons are enabled. Wizards normally display Back, Cancel, and either a Next or Finish

button. You typically enable only the Next button for the welcome page, Next and Back for interior pages, and Back and Finish for the completion page.

`dwFlags` can be a combination of the following values:

`XMLPSWIZB_BACK`	Enables the Back button. If this flag is not set, the Back button is displayed as disabled.
`XMLPSWIZB_NEXT`	Enables the Next button. If this flag is not set, the Next button is displayed as disabled.
`XMLPSWIZB_FINISH`	Displays an enabled Finish button.
`XMLPSWIZB_DISABLEDFINISH`	Displays a disabled Finish button.

Use the `PostMessage` function to send this message.

8.4.22 XMLPSM_PRESSBUTTON

SUMMARY:
Simulates the selection of a property sheet button. You can send this message explicitly or by using the `XMLPropSheet_PressButton(hDlg, iButton)` macro.

```
XMLPSM_ PRESSBUTTON
    wParam = (WPARAM) (INT) iButton
    lParam = 0
```

ARGUMENTS:
`iButton` – Index of the button to choose.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
`iButton` can be one of the following constants:

`XMLPSBTN_BACK`	Chooses the Back button
`XMLPSBTN_NEXT`	Chooses the Next button
`XMLPSBTN_FINISH`	Chooses the Finish button
`XMLPSBTN_OK`	Chooses the OK button
`XMLPSBTN_APPLYNOW`	Chooses the Apply Now button
`XMLPSBTN_CANCEL`	Chooses the Cancel button
`XMLPSBTN_HELP`	Chooses the Help button
`XMLPSBTN_BUYNOW`	Chooses the Buy Now button
`XMLPSBTN_CLOSE`	Chooses the Close button

Use the `PostMessage` function to send this message.

8.4.23 XMLPSM_SETWIZARDBTNTEXT

SUMMARY:
Sets the specified text to the specified wizard button. You can send this message explicitly or by using the `XMLPropSheet_SetWizardBtnText(hDlg, iButton, lpszText)` macro.

```
XMLPSM_ SETWIZARDBTNTEXT
    wParam = (WPARAM) (INT) iButton
    lParam = (LPARAM) (LPCTSTR) lpszText
```

ARGUMENTS:
`iButton` – Index of the button to change text.
`lpszText` – New text of the button.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
`iButton` can be one of the following values:

`XMLPSWIZBTN_BACK`	Changes text on the Back button
`XMLPSWIZBTN_NEXT`	Changes text on the Next button
`XMLPSWIZBTN_FINISH`	Changes text on the Finish button
`XMLPSWIZBTN_CANCEL`	Changes text on the Cancel button

8.4.24 XMLPSM_SETSTATUSTEXT

SUMMARY:
Sets the specified text to the status bar of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetStatusText(hDlg, lpszText)` macro.

```
XMLPSM_ SETSTATUSTEXT
    wParam = 0
    lParam = (LPARAM) (LPCTSTR) lpszText
```

ARGUMENTS:
`lpszText` – Text of the status bar.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
This message works only for property sheets which are created with the `XMLPS_HASSTATUSBAR` flag.

8.4.25 XMLPSM_GETTABCONTROL

SUMMARY:
Receives the tab control of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_GetTreeControl(hDlg)` macro.

```
XMLPSM_GETTABCONTROL
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns the handle to the tab control.

8.4.26 XMLPSM_ISDIALOGMESSAGE

SUMMARY:
Passes a message to a property sheet dialog box and indicates whether the dialog box processed the message. You can send this message explicitly or by using the `XMLPropSheet_IsDialogMessage(hDlg, pMsg)` macro.

```
XMLPSM_ISDIALOGMESSAGE
    wParam = 0
    lParam = (LPARAM) pMsg
```

ARGUMENTS:
`pMsg` – Address of an MSG structure that contains the message to be checked.

RETURN VALUE:
Returns `TRUE` if the message has been processed, or `FALSE` if the message has not been processed.

REMARKS:
Your message loop should use the `XMLPSM_ISDIALOGMESSAGE` message with modeless property sheets to pass messages to the property sheet dialog box.

If the return value indicates that the message was processed, it must not be passed to the `TranslateMessage` or `DispatchMessage` function.

8.4.27 XMLPSM_GETCURRENTPAGEHWND

SUMMARY:
Retrieves a handle to the window of the current page of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_GetCurrentPageHwnd(hDlg)` macro.

```
XMLPSM_GETCURRENTPAGEHWND
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns a handle to the window of the current page of a property sheet.

REMARKS:
Use the `XMLPSM_GETCURRENTPAGEHWND` message with modeless property sheets to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, `XMLPSM_GETCURRENTPAGEHWND` returns NULL, and you can then use the `DestroyWindow` function to destroy the dialog box.

8.4.28 XMLPSM_GETCURRENTPAGEID

SUMMARY:
Retrieves an index of the current page of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_GetCurrentPageId(hDlg)` macro.

```
XMLPSM_GETCURRENTPAGEID
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns an index of the current page of a property sheet. Returns -1 if failed.

REMARKS:
Use the `XMLPSM_GETCURRENTPAGEHWND` message with modeless property sheets to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, `XMLPSM_GETCURRENTPAGEHWND` returns `NULL`, and you can then use the `DestroyWindow` function to destroy the dialog box.

8.4.29 XMLPSM_GETPAGECOUNT

SUMMARY:
Retrieves number of pages in a property sheet. You can send this message explicitly or by using the `XMLPropSheet_GetPageCount(hDlg)` macro.

```
XMLPSM_GETPAGECOUNT
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns number of pages in a property sheet.

8.4.30 XMLPSM_SETFONT

SUMMARY:
Changes font on a page of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetFont(hDlg, index, lpxmlsf)` macro.

```
XMLPSM_SETFONT
  wParam = (int) index
  lParam = (LPARAM) (LPXMLPSSETFONT)  lpxmlsf
```

ARGUMENTS:
`index` – Index of the page to change font.
`lpxmlsf` – Address of a `XMLPSSETFONT` that contains the font information.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The `XMLPSSETFONT` structure is defined as follows:

```
typedef struct tagXMLPSSETFONT {
    HFONT hFont;
    BOOL fRedraw;
} XMLPSSETFONT, FAR *LPXMLPSSETFONT;
```

8.4.31 XMLPSM_SETTEMPLATE

SUMMARY:
Changes the template of a page of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetTemplate(hDlg, index, lpxmlst)` macro.

```
XMLPSM_SETTEMPLATE
    wParam = (int) index
    lParam = (LPARAM) (LPXMLPSSETTEMPLATE) lpxmlst
```

ARGUMENTS:
`index` – Index of the page to change template.
`lpxmlst` – Address of a `XMLPSSETTEMPLATE` that contains the template information.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The `XMLPSSETTEMPLATE` structure is defined as follows:

```
typedef struct tagXMLPSSETTEMPLATE {
    TCHAR *xmldata;
    DWORD xmlsize;
} XMLPSSETTEMPLATE, FAR *LPXMLPSSETTEMPLATE;
```

The property sheet will be automatically resized to fit the new page template.

8.4.32 XMLPSM_SETXMLPROPERTYTEXT

SUMMARY:
Changes the text of a property sheet. You can send this message explicitly or by using the `XMLPropSheet_SetXMLPropertyText(hDlg, lpxmlpt)` macro.

```
XMLPSM_SETXMLPROPERTYTEXT
    wParam = 0
    lParam = (LPARAM) (LPXMLPROPERTYTEXT) lpxmlpt
```

ARGUMENTS:
`lpxmlpt` – Address of a `XMLPROPERTYTEXT` that contains the text information.

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

REMARKS:
The `XMLPROPERTYTEXT` structure is defined as follows:

```
typedef struct tagXMLPROPERTYTEXT {
    LPCTSTR    pszOK;         // Text of the "OK" button.
    LPCTSTR    pszCancel;     // Text of the "Cancel" button.
    LPCTSTR    pszClose;      // Text of the "Close" button.
    LPCTSTR    pszApplyNow;   // Text of the "Apply Now" button.
    LPCTSTR    pszBuyNow;     // Text of the "Buy Now" button.
    LPCTSTR    pszHelp;       // Text of the "Help" button.
    LPCTSTR    pszBack;       // Text of the "Back" button.
    LPCTSTR    pszNext;       // Text of the "Next" button.
    LPCTSTR    pszFinish;     // Text of the "Finish" button.
} XMLPROPERTYTEXT, FAR *LPXMLPROPERTYTEXT;
```

8.4.33 XMLPSM_REMOVEBUYNOWBUTTON

SUMMARY:
Removes the Buy Now button from a property sheet. You can send this message explicitly or by using the `XMLPropSheet_RemoveBuyNowButton(hDlg)` macro.

XMLPSM_REMOVEBUYNOWBUTTON

```
    wParam = 0
    lParam = 0
```

RETURN VALUE:
Returns `TRUE` if successful, or `FALSE` otherwise.

8.4.34 XMLPSM_SELECTBUTTON

SUMMARY:
Selects a property sheet button. You can send this message explicitly or by using the `XMLPropSheet_SelectButton(hDlg, iButton)` macro.

```
XMLPSM_SELECTBUTTON
    wParam = (WPARAM) (INT) iButton
    lParam = 0
```

ARGUMENTS:
`iButton` – Index of the button to select.

RETURN VALUE:
Returns TRUE if successful, or FALSE otherwise.

REMARKS:
`iButton` can be one of the following constants:

`XMLPSBTN_BACK`	Chooses the Back button
`XMLPSBTN_NEXT`	Chooses the Next button
`XMLPSBTN_FINISH`	Chooses the Finish button
`XMLPSBTN_OK`	Chooses the OK button
`XMLPSBTN_APPLYNOW`	Chooses the Apply Now button
`XMLPSBTN_CANCEL`	Chooses the Cancel button
`XMLPSBTN_HELP`	Chooses the Help button
`XMLPSBTN_BUYNOW`	Chooses the Buy Now button
`XMLPSBTN_CLOSE`	Chooses the Close

Use the `PostMessage` function to send this message.

8.4.35 XMLPropertySheetCreate

SUMMARY:
Creates a property sheet from an XML dialog templates.

ARGUMENTS:
`lpXMLpss` – Pointer to a `XMLPROPERTYSHEET` structure.

DECLARATION:

```
APPLOC_EXTERN HANDLE __stdcall XMLPropertySheetCreate(
    LPXMLPROPERTYSHEET lpXMLpss);
```

RETURN VALUE:
Returns handle to the property sheet. If the function fails, the return value is `NULL`.

REMARKS:
The `XMLPROPERTYSHEET` structure is declared as follows:

```
typedef struct tagXMLPROPERTYSHEET {
    DWORD      dwFlags;       // A set of bit flags.
    HWND       hwndParent;    // Handle of the owner window.
    HINSTANCE  hInstance;     // Handle of the instance from
                              // which to load the icon
                              // resource.
    HICON      hIcon;         // Handle of the icon of the
                              // property sheet dialog box.
    HICON      hIconSm;       // Handle of the icon to use as
                              // the small icon in the title bar
                              // of the property sheet
                              // dialog box.
    LPCTSTR    pszCaption;    // Title of the property
                              // sheet dialog box.
    LPCTSTR    pszFontInfo;   // Info of the font of
                              // property sheet dialog box.
    UINT       x, y;          // Specifies coordinates, in
                              // dialog box units, of the
                              // upper-left corner of the
                              // dialog box.
    UINT       cx, cy;        // Specifies the width and height,
                              // in dialog box units,
                              // of the dialog box.
    XMLPROPERTYTEXT propText; // Text strings for property
                              // sheet buttons.
    XMLPSCallbackFn XMLPSCallback; // Callback function
                                   // for property sheet.
} XMLPROPERTYSHEET, FAR *LPXMLPROPERTYSHEET;
```

The `dwFlags` can be any combination of the following values:

`XMLPS_DEFAULT`	Uses the default meaning for all structure members.
`XMLPS_USESIZE`	Uses `cx` and `cy` as the initial size of a property sheet window. Otherwise the size will be taken from the largest page dialog.
`XMLPS_NOAPPLYNOW`	Removes the Apply Now button.
`XMLPS_NOBUYNOW`	Removes the Buy Now button.
`XMLPS_NOCANCEL`	Removes Cancel button.
`XMLPS_HASHELP`	Displays the property sheet Help button.
`XMLPS_HASSTATUSBAR`	Displays the status bar.
`XMLPS_MODELESS`	Creates the property sheet as a modeless dialog.
`XMLPS_NODISABLEPARENT`	Do not disable parent window.
`XMLPS_WIZARD`	Creates a wizard property sheet.
`XMLPS_RESIZABLE`	Creates the property sheet with resizable window frame.

`XMLPS_RTLREADING`	Displays the title of the property sheet dialog using right-to-left reading order.
`XMLPS_CONTEXTHELP`	Displays a question mark in the title bar of the window.
`XMLPS_CENTERSCREEN`	Displays property sheet window in the center of the screen.
`XMLPS_MAXIMIZEBOX`	Creates a window that has a Maximize button. Cannot be combined with the `XMLPS_CONTEXTHELP` style.
`XMLPS_MINIMIZEBOX`	Creates a window that has a Minimize button. Cannot be combined with the `XMLPS_CONTEXTHELP` style.
`XMLPS_TOPMOST`	Specifies that a window created with this style should be placed above all non-topmost windows and should stay above them, even when the window is deactivated.
`XMLPS_NOCLOSE`	Disables the Close command on the System menu.
`XMLPS_APPLYONCHANGE`	Send `XMLPSN_APPLY` notification message when property page is about to change.

The `XMLPROPERTYTEXT` structure is defined as follows:

```
typedef struct tagXMLPROPERTYTEXT {
    LPCTSTR    pszOK;          // Text of the "OK" button.
    LPCTSTR    pszCancel;      // Text of the "Cancel" button.
    LPCTSTR    pszClose;       // Text of the "Close" button.
    LPCTSTR    pszApplyNow;    // Text of the "Apply Now" button.
    LPCTSTR    pszBuyNow;      // Text of the "Buy Now" button.
    LPCTSTR    pszHelp;        // Text of the "Help" button.
    LPCTSTR    pszBack;        // Text of the "Back" button.
    LPCTSTR    pszNext;        // Text of the "Next" button.
    LPCTSTR    pszFinish;      // Text of the "Finish" button.
} XMLPROPERTYTEXT, FAR *LPXMLPROPERTYTEXT;
```

`XMLPSCallback` - pointer to the user-defined callback function. This function is used to process property sheet button notifications and draw custom frame on the top of a wizard window.

```
// hwnd -      [in] handle to the property sheet window
// iMsg -      [in] callback message
// wParam -    [in] first message parameter
// lParam -    [in] second message parameter
BOOL __stdcall XMLPSCallback(HWND hwnd, int iMsg,
    WPARAM wParam, LPARAM lParam)
{
    switch (iMsg)
    {
        case XMLPS_CALLBACKBUTTON:
        {
            // button – (INT)wParam
            // Index of button (XMLPSBTN_ constants)
            // not used - lParam
            return TRUE;
        }
        case XMLPS_CALLBACKWIZTOP:
        {
            // lpnHeight - (int *)wParam
            // Pointer to a variable that retrieves
            // height of the top frame of the wizard.
            // not used - lParam
            return TRUE;
        }
        case XMLPS_CALLBACKWIZERASE:
        {
            // hDC = (HDC)wParam
            // lprect = (LPRECT)lParam
            return TRUE;
        }
        case XMLPS_CALLBACKWIZDRAW:
        {
            // hDC = (HDC)wParam
            // lprect = (LPRECT)lParam
            return TRUE;
        }
    }
    return FALSE;
}
```

8.4.36 XMLPropertySheetAdd

SUMMARY:
Adds a page from an XML dialog template to a property sheet.

ARGUMENTS:
`hXMLpss` – Handle to the property sheet.
`lpXMLpsp` – Pointer to a `XMLPROPERTYSHEETPAGE` structure.

DECLARATION:

```
APPLOC_EXTERN BOOL __stdcall XMLPropertySheetAdd(
    HANDLE hXMLpss, LPXMLPROPERTYSHEETPAGE lpXMLpsp);
```

RETURN VALUE:
Returns `TRUE` if the page has been successfully added to the property sheet, otherwise returns `FALSE`.

REMARKS:
The `XMLPROPERTYSHEETPAGE` structure is defined as follows:

```
typedef struct tagXMLPROPERTYSHEETPAGE {
    HINSTANCE hInstance;      // Handle to the instance from
                              // which to load the dialog box
                              // template, icon, or title
                              // string resource.
    LPCTSTR   lpXMLdata;      // Identifies dialog box XML
                              // template.
    DWORD     dwXMLsize;      // Size, in characters, of the
                              // dialog box XML template.
    CheckControlSizeExFn CheckControlSizeEx; // pointer to the
                              // user-defined callback function.
    HICON     hIcon;          // Handle of the icon to use as
                              // the small icon in the title
                              // bar of the property sheet
                              // dialog box.
    LPCTSTR   pszIcon;        // Icon resource to use as the
                              // small icon in the title bar of
                              // the property sheet dialog box.
    DLGPROC   pfnDlgProc;     // Pointer to the dialog box
                              // procedure for the page. The
                              // dialog box procedure must not
                              // call the EndDialog function.
    LPARAM    lParam;         // Application-defined data.
} XMLPROPERTYSHEETPAGE, FAR *LPXMLPROPERTYSHEETPAGE;
```

The `CheckControlSizeEx` function has the same format as the `CheckControlSizeEx` function described in the Remarks section of the `XMLCreateDialogBox` function.

8.4.37 XMLPropertySheetShow

SUMMARY:
Shows a property sheet.

ARGUMENTS:
`hXMLpss` – Handle to the property sheet.
`nStartPage` – Active property sheet page.

DECLARATION:

```
APPLOC_EXTERN LRESULT __stdcall XMLPropertySheetShow(
    HANDLE hXMLpss, UINT nStartPage);
```

RETURN VALUE:
For modeless property sheets returns the property sheet's window handle. For modal property sheets returns a positive value.

For a modeless property sheet, your message loop should use XMLPSM_ISDIALOGMESSAGE to pass messages to the property sheet dialog box. Your message loop should use XMLPSM_GETCURRENTPAGEHWND to determine when to destroy the dialog box. When the user clicks the OK or Cancel button, XMLPSM_GETCURRENTPAGEHWND returns NULL. You can then use the DestroyWindow function to destroy the dialog box. The following example shows how to organize message loop for a modeless property sheet:

```
XMLps.dwFlags |= XMLPS_MODELESS;
HANDLE hXMLps = XMLPropertySheetCreate(&XMLps);
if (hXMLps)
{
    .
    .
    .
    MSG msg;
    HWND hwndPS;
    if (hwndPS = (HWND)XMLPropertySheetShow(hXMLps, 0))
    {
        while (GetMessage(&msg, NULL, 0, 0) != 0)
        {
            if (hwndPS==NULL ||
                !XMLPropSheet_IsDialogMessage(hwndPS, &msg))
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
            if (hwndPS &&
                XMLPropSheet_GetCurrentPageHwnd(hwndPS)==NULL)
            {
                DestroyWindow(hwndPS);
                hwndPS = NULL;
                PostQuitMessage(0);
            }
        }
    }
    XMLPropertySheetFree(hXMLps);
}
```

8.4.38 XMLPropertySheetLastResult

SUMMARY:
Returns last result.

ARGUMENTS:
`hXMLpss` – Handle to the property sheet.

DECLARATION:
`APPLOC_EXTERN int __stdcall XMLPropertySheetLastResult(HANDLE hXMLpss);`

RETURN VALUE:
Returns a positive value.

REMARKS:
For modal property sheets the `XMLPropertySheetLastResult` returns the same value as `XMLPropertySheetShow`.

8.4.39 XMLPropertySheetFree

SUMMARY:
Frees memory associated with a property sheet.

ARGUMENTS:
`hXMLpss` – Handle to the property sheet.

DECLARATION:

```
APPLOC_EXTERN BOOL __stdcall XMLPropertySheetFree(
    HANDLE hXMLpss);
```

RETURN VALUE:
Return `TRUE` on success, otherwise returns `FALSE`.