HTML Message Editor (M-Edit Control), Demo Version 1.0

DEVELOPER REFERENCE

Introduction

M-Edit is a HTML editor/viewer control. It supports most of HTML 4.0 and CSS 2.1, except scripts and some form elements.
Also the editor supports all typical features of text editor: copy/cut/paste, undo and redo.
M-Edit control is distributed in dynamic link library (DLL) file – medit.dll. M-Edit creates only one child window, and all its functionality is accessible through window messages. Only two functions are exported – CreateEditor and CreateViewer. Both they receive similar parameters and return window handle of created editor / viewer.
It guarantees that you may use M-Edit from any WinCE application built on any programming language.
M-Edit may also be used to set/retrieve formatted text in Microsoft Rich Text Format (RTF).

Brief description of files in the package

demo.exe	– simple demo application
medit.pdf	– this reference
medit.h	– C++ header file defining all available functions and window messages
medit.dll	– binary code of M-Edit, the only one file you need to redistribute
medit.lib	– import library, if you want to use automatic import of exported functions
license.txt	– license agreement

Exported functions

As mentioned above, meddll.dll exports two very similar functions, named CreateViewer and CreateEditor:

HWND **CreateViewer**(unsigned int **nStyle**, unsigned int **nExStyle**, const RECT* **pRect**, HWND **hParentWnd**);

HWND **CreateEditor**(unsigned int **nStyle**, unsigned int **nExStyle**, const RECT* **pRect**, HWND **hParentWnd**);

nStyle, nExStyle, pRect, hParentWnd – window style, extended window style, initial window rectangle and parent window handle – they are the same as in the CreateWindowEx API function.

You should always pass nonzero hParentWnd parameter because M-Edit cannot be top-level window.
Both these functions return window handle to the created window (if success) or zero value (if any error occurred).

Window messages

Once you get window handle to the created editor or viewer, you can interact with it using window messages. Below is the description of each control message.
If it is not specified, the message is used both for editor and viewer. Some messages (like paste) may be passed only to the editor.
All message functions accept only two parameters: wParam (type WPARAM) and lParam (type LPARAM), and return LPARAM type value.
Some messages also may return MES_OUTOFMEMORY status in case of the system does not have enough memory to process the request.

MEM_SETTEXT

Sets plaintext contents of the control.		
wParam	const wchar_t*	Pointer to wide-char string
lParam	unsigned int	Size (in chars) of the string
Return value	EditorStatus	MES_OK

MEM_GETTEXT

Used to obtain text contents of the control.		
wParam	const wchar_t**	Pointer to pointer to wide-char string allocated by the control.
lParam	unsigned int*	Pointer to variable that receives size (in chars) of the passed string
Return value	EditorStatus	MES_OK
To free returned memory block (wParam) please use MEM_FREETEXT message		

MEM_SETRTFTEXT

Converts RTF to HTML and sets contents of the control.		
wParam	const char*	Pointer to char string containing RTF text
lParam	unsigned int	Size (in chars) of the string
Return value	EditorStatus	MES_OK

MEM_GETRTFTEXT

Used to obtain RTF contents of the control. Converts HTML contents of the control to RTF.		
wParam	not used	
lParam	not used	
Return value	const char*	Pointer to null-terminated string containing the RTF.
To free returned memory block please use MEM_FREETEXT message		

MEM_GETBOLD

Retrieves the bold state of current selection		
wParam	not used	
lParam	not used	
Return value	EditorStatus	MES_TRUE if bold is set and MES_FALSE otherwise

MEM_SETBOLD

Sets the bold state of current selection		
wParam	EditorStatus	Sets bold state if passed MES_TRUE and clears if passed MES_FALSE
lParam	not used	
Return value	EditorStatus	MES_OK

MEM_GETITALIC

Retrieves the italic state of current selection		
wParam	not used	
lParam	not used	
Return value	EditorStatus	MES_TRUE if italic is set and MES_FALSE otherwise

MEM_SETITALIC

Sets the italic state of current selection		
wParam	EditorStatus	Sets italic state if passed MES_TRUE and clears if passed MES_FALSE
lParam	not used	
Return value	EditorStatus	MES_OK

MEM_GETUNDERLINE

Retrieves the underline state of current selection		
wParam	not used	
lParam	not used	
Return value	EditorStatus	MES_TRUE if underline is set and MES_FALSE otherwise

MEM_SETUNDERLINE

Sets the underline state of current selection		
wParam	EditorStatus	Sets underline state if passed MES_TRUE and clears if passed MES_FALSE
lParam	not used	
Return value	EditorStatus	MES_OK

MEM_FREETEXT

Frees previously allocated memory block		
wParam	pointer to the block	Pointer to memory block to free
lParam	not used	
Return value	EditorStatus	MES_OK

Other messages

The control also accepts some other messages (for example, retrieving HTML source text, getting/setting font face and size, etc) that are not available in demo version.

Notification messages

M-Edit sends notification messages to its parent in case of any event occurred.

MEN_SETFOCUS

Occurs when the control receives the focus. You can, for example, show input panel on this event.

MEN_KILLFOCUS

Occurs when the control loses the focus. You can, for example, hide input panel on this event.

MEN_LINK

M-Edit sends this message when the user clicks the mouse on hyperlink		
wParam	const wchar_t*	Pointer to null-terminated wide string containing the link address
lParam	unsigned int	Size of the string
Return value	not used	

Source code examples

C++ Source code example

This easy example demonstrates how to bound M-Edit into its parent control.
Also we need to add reference to library medit.lib into the project settings.

demo.h

```
#pragma once

#include "medit.h"

// simple child class (we may call it "view" in terms of MFC)
class CDemo : public CWnd
{
public:
	CDemo()
	{
	}

protected:
	// window handle of the editor
	HWND m_hwndEditor;

protected:
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg void OnChildKillFocus();
	afx_msg void OnChildSetFocus();

	DECLARE_MESSAGE_MAP();
};
```

demo.cpp

```
#include "stdafx.h"
#include "demo.h"

BEGIN_MESSAGE_MAP(CDemo, CWnd)
	ON_WM_CREATE()
	ON_WM_SIZE()
	ON_MESSAGE_VOID(MEN_KILLFOCUS, OnChildKillFocus)
	ON_MESSAGE_VOID(MEN_SETFOCUS, OnChildSetFocus)

int CDemo::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
	if (CWnd::Create(lpCreateStruct) == -1)
		return -1;

	// creates the editor
	m_hwndEditor = CreateEditor(WS_CHILD | WS_VISIBLE, 0, CRect(), m_hWnd);
	if (NULL == m_hwndEditor)
		return -1;

	return 0;
}

void CDemo::OnSize(UINT nType, int cx, int cy)
{
	CWnd::OnSize(nType, cx, cy);

	// resizes the editor to occupy entire parent window area
```



```
	::SetWindowPos(m_hwndEditor, NULL, 0, 0, cx, cy, SWP_NOZORDER);
}

void CDemo::OnChildKillFocus()
{
	// hides the input panel
	SHSipPreference(m_hwndEditor, SIP_DOWN);
}

void CDemo::OnChildSetFocus()
{
	// shows the input panel
	SHSipPreference(m_hwndEditor, SIP_UP);
}
```

C#.Net source code example

On .Net environment, we have some problems while working with pointers, but some special techniques will help us to avoid such problems.

demo.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;

namespace DemoApp
{
	/// <summary>
	/// Summary description for DemoForm.
	/// </summary>
	public class DemoForm : System.Windows.Forms.Form
	{
		private System.Windows.Forms.MainMenu mainMenu;

		// declare four required windows functions
		[DllImport("medit.dll")]
		static extern Int32 CreateEditor(Int32 Style, Int32 ExStyle, ref
			System.Drawing.Rectangle Rect, Int32 ParentHandle);
		[DllImport("coredll.dll")]
		static extern bool SetWindowPos(Int32 Handle, Int32 InsertAfter,
			Int32 x, Int32 y, Int32 cx, Int32 cy, UInt32 Flags);
		[DllImport("coredll.dll")]
		static extern Int32 GetFocus();
		[DllImport("coredll.dll")]
		static extern Int32 SendMessage(Int32 Handle, Int32 Message, Int32
			WParam, Int32 LParam);

		private const int
			WS_CHILD = 0x40000000,
			WS_VISIBLE = 0x10000000;

		private const int SWP_NOZORDER = 0x0004;

		private const int MEM_FIRST = 0x8000 + 100;
		private const int MEM_SETTEXT = MEM_FIRST;

		protected Int32 EditorHandle;
		protected Int32 ThisHandle;
```



```
public DemoForm()
{
      InitializeComponent();

      // the next two lines used to retrieve
      // window handle of the form
      this.Focus();
      this.ThisHandle = GetFocus();

      // create M-Edit control and save its
      // window handle into EditorHandle variable
      Rectangle rect = new Rectangle();
      this.EditorHandle = CreateEditor(WS_CHILD | WS_VISIBLE, 0,
            ref rect, this.ThisHandle);

      // call InitEditor in an unsafe
      // context to pass initial string to the editor
      InitEditor();
}

protected unsafe void InitEditor()
{
      String text = "Hello World!";
      char[] textBuf = text.ToCharArray();
      fixed (char* pBuf = textBuf)
      {
            SendMessage(this.EditorHandle, MEM_SETTEXT, ((IntPtr)
                  pBuf).ToInt32(), textBuf.Length);
      }
}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
      base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
      this.mainMenu = new System.Windows.Forms.MainMenu();
      this.Menu = this.mainMenu;
      this.Text = "Demo";
      this.Resize += new
            System.EventHandler(this.DemoForm_Resize);

}
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>

static void Main()
{
      Application.Run(new DemoForm());
}

private void DemoForm_Resize(object sender, System.EventArgs e)
```



```
            {
                  // resizes the editor to occupy entire parent window area
                  SetWindowPos(this.EditorHandle, 0, 0, 0, ClientSize.Width,
                        ClientSize.Height, SWP_NOZORDER);
            }
      }
}
```

