```
//
//browserthread.h : interface of the controlling function of the worker thread
//                  and of the CBrowserThread class
//
////////////////////////////////////////////////
#if !defined browserthread
#define browserthread

const int nMaxURL=9;

struct CBrowserThreadInfor {
       CString m_strServer;
       CString m_strPath;
       CString m_strKeyword;
       HANDLE m_hEventStartSave;
       CStdioFile* m_pFile;
};

// controlling function for the worker thread
UINT   BrowserThread( LPVOID pParam /* CBrowserThreadInfo ptr */);

#endif



//
//mythread.cpp
//
////////////////////////////////////////////////

#include "stdafx.h"
#include "browserthread.h"
#include "afx.h"
//#include "winbase.h"
//#include <time.h>

UINT   BrowserThread( LPVOID pParam ) {
       CBrowserThreadInfor* pBrowserInfo = (CBrowserThreadInfor*)pParam;
       SYSTEMTIME m_systimeFirst = {1999,11,6,18,12,12,12,12};
       SYSTEMTIME m_systimeSecond = {1999,11,6,18,12,12,12,12} ;
       LPSYSTEMTIME m_lpstFirstTime(&m_systimeFirst);
       LPSYSTEMTIME m_lpstSecondTime(&m_systimeSecond);

       // This thread runs in no loop and exit after saving the keyword
       // in the file "test.txt".

       //SetThreadPriority( GetCurrentThread(),THREAD_PRIORITY_HIGHEST );
    // Initialize the Internet DLL
       GetSystemTime(m_lpstFirstTime);
    CInternetSession* pSession = new CInternetSession("Thread HTML Reader");

    // See if the session is valid
    if (pSession == NULL)
    {
              ::MessageBeep(MB_ICONEXCLAMATION);
              AfxMessageBox("Internet session initialization failed!",
         MB_OK | MB_ICONEXCLAMATION);
              return 1;
       }
       // Initialize HTTP session
       // CTime t1 = CTime::GetCurrentTime();

       //get the first time

    CHttpConnection* pConnect = pSession->GetHttpConnection(pBrowserInfo->m_strServer);

    // See if connection is valid
    if (pConnect == NULL)
    {
              ::MessageBeep(MB_ICONEXCLAMATION);
        AfxMessageBox("Internet connection failed!",
         MB_OK | MB_ICONEXCLAMATION);

        // Close session
        pSession->Close();
        delete pSession;
        return 1;
       }
```

```
        // Open an HTTP request handle
        CHttpFile* pHttpFile = pConnect->OpenRequest("GET", pBrowserInfo->m_strPath);

        // See if HTTP request handle is valid
        if (pHttpFile == NULL)
        {
                ::MessageBeep(MB_ICONEXCLAMATION);
                AfxMessageBox("HTTP request failed!",
          MB_OK | MB_ICONEXCLAMATION);

                // Close session handles and clean up
                pConnect->Close();
                pSession->Close();
                delete pConnect;
                delete pSession;
                return 1;
        }

        // Send the request
        pHttpFile->AddRequestHeaders("Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
image/png, */*\r\nAccept-Language: en-us,zh-cn;q=0.5\r\n");
//      pHttpFile->AddRequestHeaders("Connection: keep-alive\r\n");
//      pHttpFile->AddRequestHeaders("User-Agent: Mozilla/4.61 [en] (WinNT; I)\r\n");
        pHttpFile->AddRequestHeaders("User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT;
DigExt; freenet)\r\n");
//      pHttpFile->AddRequestHeaders("Pragma: no-cache\r\n");
        pHttpFile->AddRequestHeaders("Accept-Encoding: gzip, deflate\r\n");
//      pHttpFile->AddRequestHeaders("Via: 1.0 csdalpha3.sbu.ac.uk:8080 (Squid/2.2.STABLE5)\r\n");
//      pHttpFile->AddRequestHeaders("X-Forwarded-For: unknown\r\n");
//      pHttpFile->AddRequestHeaders("Cache-Control: max-age=259200\r\n");
        BOOL bSendRequest = pHttpFile->SendRequest();

        if (bSendRequest) {
                /*
           // Get the size of the requested file
      char achQueryBuf[16];
      DWORD dwFileSize;
      DWORD dwQueryBufLen = sizeof(achQueryBuf);

      BOOL bQuery = pHttpFile->QueryInfo(HTTP_QUERY_CONTENT_LENGTH,
         achQueryBuf, &dwQueryBufLen, NULL);

      if (bQuery)
      {
         // The query succeeded, specify memory needed for file
         dwFileSize = (DWORD)atol(achQueryBuf);
                AfxMessageBox(dwFileSize);
      }
      else
      {

         // The query failed, so guess at a max file size
         dwFileSize = 10 * 1024;
      }
      // Allocate a buffer for the file data
      //char* lpszBuf = new char[dwFileSize + 1];
          */

                CString htmlText(_T("")), dhtmlText(_T(""));
                // Read the file
                //UINT uBytesRead = pHttpFile->Read(lpszBuf, dwFileSize + 1);
                while(pHttpFile->ReadString(htmlText)) {
                        dhtmlText+=htmlText+"\r\n";
                }

                // Getting the end time
                //CTime t2 = CTime::GetCurrentTime();
                // Subtract 2 CTimes
                //CTimeSpan ts = t2 - t1;
                //char buffer[20];
                //_ltoa(ts.GetTotalSeconds() , buffer, 10 );

                GetSystemTime(m_lpstSecondTime);

                //searching keyword
                if (dhtmlText.IsEmpty()||pBrowserInfo->m_strKeyword.IsEmpty())
                        dhtmlText = "no text!";
```

```
		else {
			// Check for a keyword
			int nSlash = dhtmlText.Find(pBrowserInfo->m_strKeyword);
			if (nSlash != -1)  // There's a specified, so grab it
			{
				// Strip off "keyWord"
				CString strTemp = dhtmlText.Mid(nSlash+(pBrowserInfo-
>m_strKeyword).GetLength());
				nSlash = strTemp.Find(pBrowserInfo->m_strKeyword);
				dhtmlText= strTemp.Left(nSlash);
			} else
				dhtmlText = "no time!";
		}

		//change unsigned time word type to signed time interger
		int nFirstHour=0, nFirstMinute=0, nFirstSecond=0, nFirstMilli=0;
		nFirstHour=m_lpstFirstTime->wHour;
		nFirstMinute=m_lpstFirstTime->wMinute;
		nFirstSecond=m_lpstFirstTime->wSecond;
		nFirstMilli=m_lpstFirstTime->wMilliseconds;

		int nSecondHour=0, nSecondMinute=0, nSecondSecond=0, nSecondMilli=0;
		nSecondHour=m_lpstSecondTime->wHour;
		nSecondMinute=m_lpstSecondTime->wMinute;
		nSecondSecond=m_lpstSecondTime->wSecond;
		nSecondMilli=m_lpstSecondTime->wMilliseconds;

		int m_nMillisec;
		m_nMillisec=(nSecondMilli - nFirstMilli);
		//if needing borrow one second from the second bit
		if(m_nMillisec<0) {
			m_nMillisec=nSecondMilli + 1000 - nFirstMilli;
			nSecondSecond=nSecondSecond - 1;
		}

		int m_nSecond;
		//adjust the second and calculate the second
		if(nSecondSecond<0) {
			nSecondSecond=nSecondSecond + 60;
			nSecondMinute=nSecondMinute - 1;
		}
		m_nSecond=nSecondSecond - nFirstSecond;
		if(m_nSecond<0) {
			m_nSecond=nSecondSecond + 60 - nFirstSecond;
			nSecondMinute=nSecondMinute - 1;
		}

		int m_nMinute;
		//adjusting the minute and calculate the minute
		if(nSecondMinute<0) {
			nSecondMinute=nSecondMinute + 60;
			nSecondHour=nSecondHour - 1;
		}
		m_nMinute=nSecondMinute - nFirstMinute;
		if(m_nMinute<0) {
			m_nMinute=nSecondMinute + 60 - nFirstMinute;
			nSecondHour=nSecondHour - 1;
		}

		int m_nHour;
		//adjusting the hour and never running beyond two days
		if(nSecondHour<0) {
			nSecondHour=nSecondHour + 24;
		}
		m_nHour=nSecondHour - nFirstHour;
		if(m_nHour<0) {
			m_nHour=nSecondHour + 24 - nFirstHour;
		}

		long int result=0;
		result=(m_nHour*60*60*1000+m_nMinute*60*1000+m_nSecond*1000+m_nMillisec);


		CString strTemp;
		char buffer[20];
		_ltoa(m_lpstFirstTime->wHour , buffer, 10 );
		strTemp=buffer;
		_ltoa(m_lpstFirstTime->wMinute , buffer, 10 );
```

```
            strTemp=strTemp + ":" + buffer;
            _ltoa(m_lpstFirstTime->wSecond , buffer, 10 );
            strTemp=strTemp + ":" + buffer;
            _ltoa(m_lpstFirstTime->wMilliseconds , buffer, 10 );
            strTemp=strTemp + ":" + buffer;

            //construct the writing string
            dhtmlText = pBrowserInfo->m_strServer + " , "+dhtmlText + " , ";
            dhtmlText = dhtmlText + strTemp + " ";

            _ltoa(m_lpstSecondTime->wHour , buffer, 10 );
            strTemp=buffer;
            _ltoa(m_lpstSecondTime->wMinute , buffer, 10 );
            strTemp=strTemp + ":" + buffer;
            _ltoa(m_lpstSecondTime->wSecond , buffer, 10 );
            strTemp=strTemp + ":" + buffer;
            _ltoa(m_lpstSecondTime->wMilliseconds , buffer, 10 );
            strTemp=strTemp + ":" + buffer;
            _ltoa(result, buffer, 10 );
            strTemp=strTemp + " " + buffer;

            dhtmlText = dhtmlText + strTemp + " ";
            dhtmlText = dhtmlText+"\r\n";

            WaitForSingleObject(pBrowserInfo->m_hEventStartSave, INFINITE);
            TRY {
                    //DWORD dwActual = (pBrowserInfo->m_pFile)->SeekToEnd();
                    (pBrowserInfo->m_pFile)->WriteString( dhtmlText );
            }  CATCH( CFileException, e ){
                    #ifdef _DEBUG
                    afxDump << "File could not be opened " << e->m_cause << "\n";
                    #endif
            }
                    END_CATCH

            //      AfxMessageBox( dhtmlText );
            SetEvent(pBrowserInfo->m_hEventStartSave);

            // Close all open Internet handles
            pHttpFile->Close();
            pConnect->Close();
            pSession->Close();

            // Clean up
            delete pHttpFile;
            delete pConnect;
            delete pSession;
    }

        return 0;
}
```

```
// testDlg.h : header file
//

#if !defined(AFX_TESTDLG_H__3A39F426_832B_11D3_B190_00600846ACA2__INCLUDED_)
#define AFX_TESTDLG_H__3A39F426_832B_11D3_B190_00600846ACA2__INCLUDED_

#include "browserthread.h"    // Added by ClassView
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CTestDlg dialog

class CTestDlg : public CDialog
{
// Construction
public:
	CBrowserThreadInfor m_browserThreadInfo[nMaxURL];
	CWinThread* m_pBrowserThread[nMaxURL];
//	CWinThread* m_pBrowserThread1;
	CTestDlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CTestDlg)
	enum { IDD = IDD_TEST_DIALOG };
	CEdit	m_editUrl;
	CEdit	m_editHtml;
	CButton	m_btnGo;
	CButton	m_btnClose;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CTestDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
//	LPSYSTEMTIME m_lpstTime[nMaxURL];
//	LPSYSTEMTIME lpst;
	CStdioFile m_strFilename;
	HICON m_hIcon;
   CString  m_strServer[nMaxURL];
   CString  m_strPath[nMaxURL];

   CString ParseURLserver(CString& strUrl);
   CString ParseURLpath(CString& strUrl);
   void DisplayRawHtml(char* lpszBuffer);
   void DisplayStringHtml(CString lpszBuffer);

	// Generated message map functions
	//{{AFX_MSG(CTestDlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnBtnGo();
	afx_msg void OnBtnClose();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
private:
	CString SearchTime(CString& htmlStr, CString& keyWord);
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_TESTDLG_H__3A39F426_832B_11D3_B190_00600846ACA2__INCLUDED_)




// testDlg.cpp : implementation file
//

#include "stdafx.h"
```

```
#include "test.h"
#include "testDlg.h"
#include "browserthread.h"
#include <stdlib.h>
#include "winbase.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestDlg dialog

CTestDlg::CTestDlg(CWnd* pParent /*=NULL*/)
	: CDialog(CTestDlg::IDD, pParent)
{
//	m_strServer="";
//	m_strPath="";
	//{{AFX_DATA_INIT(CTestDlg)
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CTestDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CTestDlg)
	DDX_Control(pDX, IDC_URL_EDIT, m_editUrl);
	DDX_Control(pDX, IDC_HTML_EDIT, m_editHtml);
	DDX_Control(pDX, ID_GO, m_btnGo);
```

```
	DDX_Control(pDX, ID_CLOSE, m_btnClose);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CTestDlg, CDialog)
	//{{AFX_MSG_MAP(CTestDlg)
	ON_WM_SYSCOMMAND()
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(ID_GO, OnBtnGo)
	ON_BN_CLICKED(ID_CLOSE, OnBtnClose)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CTestDlg message handlers

BOOL CTestDlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Add "About..." menu item to system menu.

	// IDM_ABOUTBOX must be in the system command range.
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
			pSysMenu->AppendMenu(MF_SEPARATOR);
			pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

//Set the default URL
	m_editUrl.SetWindowText("http://www.sbu.ac.uk/index.html");
	m_editUrl.SetFocus();

	return TRUE;  // return TRUE  unless you set the focus to a control
}

void CTestDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CTestDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
```

```
                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
                dc.DrawIcon(x, y, m_hIcon);
        }
        else
        {
                CDialog::OnPaint();
        }
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CTestDlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

void CTestDlg::OnBtnGo()
{
        // Reading URL, time from the input file
        int nLoop=0, i=0, j=0;
        int nTime[nMaxURL];
        CString strURL[nMaxURL];
    CString keyWord(_T("+"));

        char* pFileName = "testinput.txt";
        CStdioFile fileInput( pFileName, CFile::modeRead );

        char buf[256];
        fileInput.ReadString( buf, 255 );
        nLoop=atoi( buf);

        fileInput.ReadString( buf, 255 );
        i=atoi( buf);
        if(i>nMaxURL){
                AfxMessageBox("You are not allowed to have so many URLs!!!");
                return;
        }
        for(j=0; j<i; j++) {
                fileInput.ReadString( buf, 255 );
                strURL[j]= buf;
                fileInput.ReadString( buf, 255 );
                nTime[j]=(int)(1000*atof( buf));
        }
        fileInput.Close();

        for(j=0; j<i; j++) {
                m_strServer[j]=ParseURLserver(strURL[j]);
                m_strPath[j]=ParseURLpath(strURL[j]);
        }
   // Disable the user interface sending & processing request
   m_editUrl.EnableWindow(FALSE);
   m_btnGo.EnableWindow(FALSE);
   m_btnClose.EnableWindow(FALSE);

   // Display wait cursor
   CWaitCursor wait;

   pFileName = "test.txt";
//        TRY {
        m_strFilename.Open( pFileName, CFile::modeCreate | CFile::modeWrite );

        for(j=0; j<i; j++) {
                // Initial browserThreadInfo object, Event: auto reset and initial signal
                m_browserThreadInfo[j].m_hEventStartSave=CreateEvent(NULL, FALSE, TRUE, NULL);
                m_browserThreadInfo[j].m_pFile=&m_strFilename;
                // The above two paras should be global variant, I don't want to chang it
                m_browserThreadInfo[j].m_strServer=m_strServer[j];
                m_browserThreadInfo[j].m_strPath=m_strPath[j];
                m_browserThreadInfo[j].m_strKeyword=keyWord;
        }
```

```
        while(nLoop) {
                //test my thread
                for(j=0; j<i; j++) {
                        m_pBrowserThread[j]=AfxBeginThread(BrowserThread, &m_browserThreadInfo[j]);
                        Sleep(nTime[j]);
                }
                nLoop--;
        }


   // Get the current URL
   CString strUrl, strServer, strPath;
   m_editUrl.GetWindowText(strUrl);

   // See if the URL looks valid
   if (strUrl.IsEmpty() || strUrl.Left(7) != "http://")
   {
      ::MessageBeep(MB_ICONASTERISK);
      AfxMessageBox("Sorry, http address required...",
         MB_OK | MB_ICONINFORMATION);
      return;
   }

/*
   // Parse the URL to get server name and file path (if any)
   strServer=ParseURLserver(strUrl);
   strPath=ParseURLpath(strUrl);

   // Initialize the Internet DLL
   CInternetSession* pSession = new CInternetSession("Raw HTML Reader");

   // See if the session is valid
   if (pSession == NULL)
   {
      ::MessageBeep(MB_ICONEXCLAMATION);
      AfxMessageBox("Internet session initialization failed!",
         MB_OK | MB_ICONEXCLAMATION);
      return;
   }
   // Initialize HTTP session
   CHttpConnection* pConnect = pSession->GetHttpConnection(strServer);

   // See if connection is valid
   if (pConnect == NULL)
   {
      ::MessageBeep(MB_ICONEXCLAMATION);
      AfxMessageBox("Internet connection failed!",
         MB_OK | MB_ICONEXCLAMATION);

      // Close session
      pSession->Close();
      delete pSession;
      return;
   }
   // Open an HTTP request handle
   CHttpFile* pHttpFile = pConnect->OpenRequest("GET", strPath);

   // See if HTTP request handle is valid
   if (pHttpFile == NULL)
   {
      ::MessageBeep(MB_ICONEXCLAMATION);
      AfxMessageBox("HTTP request failed!",
         MB_OK | MB_ICONEXCLAMATION);

      // Close session handles and clean up
      pConnect->Close();
      pSession->Close();

      delete pConnect;
      delete pSession;

      return;
   }


   // Send the request
        pHttpFile->AddRequestHeaders("Accept: text/*\r\nAccept-Language: en-us,zh-cn;q=0.5\r\n");
```

```
        pHttpFile->AddRequestHeaders("Accept: image/gif\r\n");
        pHttpFile->AddRequestHeaders("Connection: keep-alive\r\n");
        pHttpFile->AddRequestHeaders("User-Agent: Mozilla/4.0 (compatible; MSIE 5.0; Windows NT;
DigExt; freenet)\r\n");
//      pHttpFile->AddRequestHeaders("Pragma: no-cache\r\n");
        pHttpFile->AddRequestHeaders("Accept-Encoding: gzip, deflate\r\n");
//      pHttpFile->AddRequestHeaders("X-Forwarded-For: unknown\r\n");
//      pHttpFile->AddRequestHeaders("Host: webdb.sbu.ac.uk\r\n");
        BOOL bSendRequest = pHttpFile->SendRequest();

   if (bSendRequest)
   {
      // Get the size of the requested file
      char achQueryBuf[16];
      DWORD dwFileSize;
      DWORD dwQueryBufLen = sizeof(achQueryBuf);

      BOOL bQuery = pHttpFile->QueryInfo(HTTP_QUERY_CONTENT_LENGTH,
         achQueryBuf, &dwQueryBufLen, NULL);

      if (bQuery)
      {
         // The query succeeded, specify memory needed for file
         dwFileSize = (DWORD)atol(achQueryBuf);
      }
      else
      {
         // The query failed, so guess at a max file size
         dwFileSize = 10 * 1024;
      }
      // Allocate a buffer for the file data
      //char* lpszBuf = new char[dwFileSize + 1];
          CString htmlText(_T("")), dhtmlText(_T("")), copyHtmlText(_T(""));

      // Read the file
      //UINT uBytesRead = pHttpFile->Read(lpszBuf, dwFileSize + 1);
          while(pHttpFile->ReadString(htmlText)) {
                  dhtmlText+=htmlText+"\r\n";//+copyHtmlText;
          }
          DisplayStringHtml((dhtmlText+SearchTime(dhtmlText, keyWord)));

      // Close all open Internet handles
      pHttpFile->Close();
      pConnect->Close();
      pSession->Close();
      // Clean up
      delete pHttpFile;
      delete pConnect;
      delete pSession;
   }


*/
   // Enable the user interface
   m_btnGo.EnableWindow(TRUE);
   m_btnClose.EnableWindow(TRUE);
   m_editUrl.EnableWindow(TRUE);

//   Sleep(5000);
//   f.Close();

}

CString CTestDlg::ParseURLserver(CString& strUrl)
{
   if (strUrl.IsEmpty())
      return "www.sbu.ac.uk";

   // Strip off "http://"
   CString strTemp = strUrl.Mid(7) ;

   // Check for a path after the host name
   int nSlash = strTemp.Find("/");

   if (nSlash != -1)  // There's a path specified, so grab it
   {
          return (strTemp.Left(nSlash));
   }
```

```
   else
          return (strTemp);
}

CString CTestDlg::ParseURLpath(CString& strUrl){
   if (strUrl.IsEmpty())
      return "/";

   // Strip off "http://"
   CString strTemp = strUrl.Mid(7) ;

   // Check for a path after the host name
   int nSlash = strTemp.Find("/");

   if (nSlash != -1)  // There's a path specified, so grab it
   {
      return( strTemp.Mid(nSlash));
   }
   else
          return "/";
}

void CTestDlg::DisplayRawHtml(char* lpszBuffer)
{
   m_editHtml.SetWindowText((LPCTSTR)lpszBuffer);
}

void CTestDlg::DisplayStringHtml(CString strBuffer)
{
   m_editHtml.SetWindowText(strBuffer);
}

void CTestDlg::OnBtnClose()
{
       // TODO: Add your control notification handler code here
       // in this applicaiton, the program owns the worker thread.
       // the program destructor is responsible for killing the active worker thread.

       // this stu.pid way. Reading maxURLnumber and loop from the input file
       int nLoop=0, i=0, j=0;

       char* pFileName = "testinput.txt";
       CStdioFile fileInput( pFileName, CFile::modeRead );

       char buf[10];
       fileInput.ReadString( buf, 9 );
       nLoop=atoi( buf);
       fileInput.ReadString( buf, 9 );
       i=atoi( buf);
       fileInput.Close();

       DWORD dwExitCode;
       for(j=0;j<i;j++){
              if(m_pBrowserThread != NULL &&
              GetExitCodeThread(m_pBrowserThread[j]->m_hThread, &dwExitCode) &&
              dwExitCode==STILL_ACTIVE) {
              //kill the worker thread by setting the "kill thread" event.
              Sleep(2000); //wait for 2 secs.
              j--;
              }
       }
       m_strFilename.Close();
       CDialog::OnCancel();

}

CString CTestDlg::SearchTime(CString & htmlStr, CString & keyWord)
{
   if (htmlStr.IsEmpty()||keyWord.IsEmpty())
      return "no time!";

   // Check for a keyword
   int nSlash = htmlStr.Find(keyWord);

   if (nSlash != -1)  // There's a specified, so grab it
   {
          // Strip off "keyWord"
         CString strTemp = htmlStr.Mid(nSlash+keyWord.GetLength());
```

```
            nSlash = strTemp.Find(keyWord);
         return strTemp.Left(nSlash);
      }
      else
         return "no time!";

}
```