Sharing Data with Memory Mapped Files

DennisMcK

Sharing data between applications

Communicating between Liberty BASIC applications is a subject that comes up every so often in the different LB groups and some clever work-arounds have been tried with varying degrees of success. A recent post prompted some more research on the subject and led to this article. It turns out that the means to accomplish this in Liberty BASIC have been available for years. It's called File Mapping. As the name suggests this method uses a file that contains the data to be shared.Others have used files to pass information back and forth before but this is different in two major ways. One, the file doesn't have to exist on the hard drive, it can exist only in memory. Two, the file data can be accessed with a pointer, allowing very fast reads.

One rule you have to follow is to synchronize access to the filememory. In other words, don't allow more than one application to read or write to the file at the same time. This access control can accomplished by using something called a Mutex object, whose "name comes from the fact that it is useful in coordinating mutually exclusive access to a shared resource."

File Mapping Overview

A mapped file is created by calling CreateFileMappingA(). The file must have a unique name just like any other file path and the name of the file is used when any application wants to access it. CreateFileMappingA either creates the file-mapping object and returns a handle to it, or if the mapped file has already been created, returns a handle to the existing file-mapping object.

To get to the data in the file a call is made to MapViewOfFile() using the handle obtained from CreateFileMappingA. MapViewOfFile creates a 'view' of the file in the program's address space and returns a pointer to it.

The data in the view can then be copied into a structure and used. If your program changes any of the data, it can be saved in the mapped file by copying the structure back into the file-view and calling UnmapViewOfFile().

The mapped file handle must be be closed with a call to CloseHandle(). Failing to do this will cause a memory leak as Windows will not remove the mapped file object from memory until all open handles to it are closed.

Mutex Overview

A mutex object must have a unique name and that name is used when any application wants to use it. A mutex object is created by calling CreateMutex() and a handle to the mutex is returned. If the mutex has already been created, a handle to the mutex can be obtained by calling OpenMutex(). To use a mutex a program can request ownership of the mutex object by calling WaitForSingleObject(). Only one program may own the specified mutex at one time. If the mutex object is owned by another program, the WaitForSingleObject function blocks the requesting program until the owning program releases the mutex object or the function times out. An owning program releases a mutex by calling ReleaseMutex(). The mutex handle must be be closed with a call to CloseHandle().

Program Design

When the program starts:

- Define the strings for the file-mapping and mutex names. These strings are limited to _MAX_PATH number of characters and should be as unique as you can make them. The names can contain any character except the backslash character (\). Using the names from the example code for your program is not a good idea.
- Define one struct with an element for each variable to be shared.
- The string names and structure design must be identical in each program.
- Create the file-mapping object.
- Create or open the mutex object. Only open it if it already exists.

To read shared data:

- Obtain ownership of the mutex.
- Map a view of the file-mapping object.
- Copy the view into the struct.
- Unmap the view.
- Release the mutex.

To save shared data:

- Obtain ownership of the mutex.
- Map a view of the file-mapping object.
- Copy the struct into the view.

- Unmap the view.
- Release the mutex.

When the program ends:

- Close the handle to the file-mapping object.
- Close the handle to the mutex object.

Additional Info

The example code with this article shows how do each of these steps in detail. You should read the documentation for File Mapping as the example code is simplified. There are limits to the size of the file-mapping object and the mapped view does not have to be a view of the entire file. More than one view of the file-mapping object may be created and used at the same time. Also, additional steps are required when using a regular file that is stored on the hard drive. Other factors need to be considered when the mapped-file exists on one computer and is accessed by programs on different computers in a network environment.

Documentation

Hopefully this article has presented enough information to give you a good start toward building your own data sharing programs. Additional information can be found in the Borland Win32 Programmer's Reference (Win32.hlp) and at MSDN.

Dennis McKinney

September 2, 2003

Demo

```
' Sharing data with other applications.
' Author: Dennis McKinney. August 30, 2003.

' Run several instances of this program and arrange the windows
' so they are all visible. Click the Change Data button on different
' windows and observe the results.

    nomainwin

    ERROR.ALREADY.EXISTS = 183
```

```
    ' The same mapped-file name and mutex name must be used in
    ' all programs that will share data.
    ' Use any strings you like and try to make them unique.
    MappedFileName$ = "my_file_mapping_object"
    MutexName$      = "my_file_access_control_mutex"

    ' Define a common data structure to hold the shared data.
    ' The structure's elements will be the shared data variables.
    ' NOTE WELL - You must use type char[x] for strings and you must
    ' allow one additional character for the terminating null or "0"
    ' character that Windows appends to all strings.
    struct Shared, _
      numA  AS LONG, _
      numB  AS LONG,_
      strVar as char[255] 'String variable with up to 254 characters.

    sizeofShared = len(Shared.struct)

    g.flag = 1 'Allows the data to be printed on the first pass.
    g.caption$ = "Local string = NULL"

    WindowWidth = (DisplayWidth/2)-20
    WindowHeight = DisplayHeight/2-50
    UpperLeftX=int((DisplayWidth-WindowWidth)/2)
    UpperLeftY=int((DisplayHeight-WindowHeight)/2)

    teHeight = (DisplayHeight/2)-120
    texteditor #main.te, 5, 12, 210, teHeight
    button #main.changeData, "Change Data", [
change.shared.data], UL, 217, 12, 75, 25

    menu #main, "Edit", " ", [x]
    open " " for window as #main
    #main, "trapclose [quit]"
    hWnd = hwnd(#main)

    gosub [init.shared.data]
    goto [get.shared.data]

[quit]
    calldll #kernel32, "CloseHandle", hMutex as ulong, ret as long
    ' You must close the file-mapping object or you will
    ' cause a memory leak. This call will succeed even if other
    ' processes are still using the file-mapping object.
    calldll #kernel32, "CloseHandle", hMappedFile as ulong, ret as
```

```
long
    close #main

END

'*****************************************************************
[init.shared.data]
    ' Calling CreateFileMappingA with a file handle of -1
    ' creates a file-mapping object backed by the operating-system
    ' paging file rather than by a named file on the hard drive.
    ' The return is a handle to the file-mapping object.
    ' If the object existed before the function call, the return value
    ' is a valid handle to the existing file-mapping object.

    calldll #kernel32, "CreateFileMappingA", -1 as long, 0 as long, _
        _PAGE_READWRITE as ulong, 0 as ulong, sizeofShared as ulong, _
        MappedFileName$ as ptr, hMappedFile as ulong

    ' See if the file-mapping object has already been created.
    ' If it has then GetLastError() will return ERROR.ALREADY.EXISTS
    LastError = GetLastError()

    if hMappedFile > 0 then
        if LastError = ERROR.ALREADY.EXISTS then
          ' By design in this code, if the file-mapping object
          ' already exists then the mutex object has already been
          ' created by a previous process. So open the mutex object
          ' and get the handle to it for this process to use.
            calldll #kernel32, "OpenMutexA",
 _MUTEX_ALL_ACCESS as ulong,_
                0 as long, MutexName$ as ptr, hMutex as ulong
        else
          ' This is the first process so create the mutex object for
          ' file-mapping object access control purposes.
            calldll #kernel32, "CreateMutexA", 0 as ulong,_
                0 as long, MutexName$ as ptr, hMutex as ulong
        end if
    end if
return

[change.shared.data]
    timer 0
    ' Request ownership of the mutex object and
    ' wait up to 30 seconds for the request to be granted.
    ' Choose your own waiting time.
    ' Once ownership is granted to this process, no other
```

```
    ' process can 'own' the mutex until this process
    ' releases it with a call to ReleaseMutex.
    signalState = WaitForSingleObject(hMutex, 30000)

    select case signalState
        case _WAIT_OBJECT_0, _WAIT_ABANDONED
            ' Ok. We own the mutex. Now we can proceed.

            'Get a pointer to the shared file-mapping object
            calldll #kernel32, "MapViewOfFile", hMappedFile as ulong,_
                _FILE_MAP_WRITE as ulong, 0 as ulong, 0 as ulong, _
                sizeofShared as ulong, BaseAddress as ulong

            ' Copy the data from the file-mapping object into
            ' the struct 'Shared'.
            calldll #kernel32,"RtlMoveMemory", Shared as ptr, _
                BaseAddress as ulong, sizeofShared as ulong, ret as
void

            ' Change the data
            Shared.numA.struct = Shared.numA.struct + 1
            Shared.numB.struct = Shared.numB.struct + 1
            j = j + 1
            Shared.strVar.struct = "Test string " + str$(j*100)

            g.caption$ = "Local string = " + str$(
Shared.strVar.struct)

           ' save the new data (struct Shared)
           ' in the file-mapping object.
            calldll #kernel32,"RtlMoveMemory", BaseAddress as ulong, _
                Shared as ptr, sizeofShared as ulong, ret as void

            ' Unmap the current view
            calldll #kernel32, "UnmapViewOfFile",
 BaseAddress as ulong,_
                ret as long


' Release ownership of the mutex so another process can use it
            calldll #kernel32, "ReleaseMutex", hMutex as ulong,
 ret as long

        case _WAIT_TIMEOUT
            ' Another process is hogging the mutex ownership.
            ' Try again later.
```

```
        case else
            r = GetLastError()
            errmsg$ = FormatMessage$(r)
            notice "Error on Wait for Mutex" + chr$(10) + errmsg$

    end select

[get.shared.data]
    timer 0
    signalState = WaitForSingleObject(hMutex, 30000)

    select case signalState
        case _WAIT_OBJECT_0, _WAIT_ABANDONED

            'Get a pointer to the shared file-mapping object
            calldll #kernel32, "MapViewOfFile", hMappedFile as ulong,_
                _FILE_MAP_WRITE as ulong, 0 as ulong, 0 as ulong, _
                sizeofShared as ulong, BaseAddress as ulong

            ' Copy the data from the file-mapping object into
            ' the struct 'Shared'.
            calldll #kernel32,"RtlMoveMemory", Shared as ptr, _
                BaseAddress as ulong, sizeofShared as ulong, ret as
void

            ' Do something with the data
            if g.flag <> Shared.numA.struct then
                g.flag = Shared.numA.struct
                #main.te, "!cls"
                #main.te, "Shared.numA.struct = ";Shared.numA.struct
                #main.te, "Shared.numB.struct = ";Shared.numB.struct
                #main.te,
"Shared.strVar.struct = ";Shared.strVar.struct
            end if

            calldll #user32, "SetWindowTextA", hWnd as long, _
                g.caption$ as ptr, ret as long

            ' Unmap the current view
            calldll #kernel32, "UnmapViewOfFile",
 BaseAddress as ulong,_
                ret as long


' Release ownership of the mutex so another process can use it
```

```
            calldll #kernel32, "ReleaseMutex", hMutex as ulong,
 ret as long

        case _WAIT_TIMEOUT
            ' Another process is hogging the mutex ownership.
            ' Try again later.

        case else
            r = GetLastError()
            errmsg$ = FormatMessage$(r)
            notice "Error on Wait for Mutex" + chr$(13) + errmsg$

    end select
    timer 500, [get.shared.data]
wait

'****************************************************************

function WaitForSingleObject(hHandle, dwMilliseconds)
    calldll #kernel32, "WaitForSingleObject", hHandle as ulong,_
        dwMilliseconds as ulong, WaitForSingleObject as ulong
end function

function GetLastError()
    calldll #kernel32, "GetLastError", GetLastError as ulong
end function

function FormatMessage$(msgID)
    str$ = space$(256)
    calldll #kernel32, "FormatMessageA",
 _FORMAT_MESSAGE_FROM_SYSTEM as ulong, _
        0 as ulong, msgID as ulong, 0 as ulong, str$ as ptr, _
        256 as ulong, 0 as ulong, r as ulong
    FormatMessage$ = trim$(str$)
end function
```