Allowing only one instance of your program

-

StPendl *May 8, 2011*

Table of Contents

The Basics

Every now and then one wants to make sure that only one instance of his application is running to prevent problems with shared resources when multiple instances are run concurrently.

This is called **mut**ually **ex**clusive access, in short Mut-Ex or mutex.

MSDN has a nice article about Mutex Objects, which you should read to get a general understanding of the subject.

The API functions involved

The CreateMutex function

The first function is the one we use to create a mutex, which is the CreateMutex Function
It receives the security attributes, initial owner and name of our mutex object to create.

We do not want to grant other processes or threads access to our mutex, so we use NULL for the security attributes.
We want to be the initial owner of the mutex to prevent subsequent instances of our program to be run, so we set that to TRUE = 1.
We select a unique name for our mutex to have it easily separated from other mutex objects.

In return we get the Windows handle of the mutex, so we can access it at a later stage.

Here is the code to do all of this.

```
    ' inherit the default security attributes
    lpMutexAttributes = _NULL

    ' optain initial ownership by setting this to 1
    bInitialOwner = 1


' use a unique name for the mutex, maximum _MAX_PATH = 260 characters
    lpName$ = "LBMutExDemo"

    calldll #kernel32, "CreateMutexA",_
        lpMutexAttributes as ulong,_
        bInitialOwner     as long,_
        lpName$           as ptr,_
        hMutex            as ulong
```

The GetLastError function

The second function is the one we use to check if our mutex is already in use, which is the GetLastError Function
It returns an error number indicating that the mutex is already present, which is ERROR_ALREADY_EXISTS = 183.

If the mutex already exists, we do want our program to notify the user of this situation and end gracefully.

The call to CreateMutex has already created another handle to the mutex, so we need to close that handle too.

Here is the code to do all of this.

```
' get the last error, so we know if there is already an instance runni
ng
```

```
    calldll #kernel32, "GetLastError",_
        LastError as ulong


' check if there is already an instance running (ERROR_ALREADY_EXISTS
= 183)
    if LastError = 183 then
        ' close the Windows handle obtained
        calldll #kernel32, "CloseHandle",_
            hMutex as ulong,_
            result as long

        ' exit if there is already an instance running
        notice "Exiting!"; chr$(13);_
            "Another instance of this program is already running!"
        end
    end if
```

The ReleaseMutex function

The third function is the one we use to free our mutex before we end the program, which is the ReleaseMutex Function
It removes the ownership from the mutex.

This function must be called at the end of your program to be able to debug and run your program from within the LB editor.
If it happens, that your program crashes, while the mutex is active, you need to restart the LB editor to free the mutex.

Here is the code to do all of this.

```
    ' free the mutex so it is no longer owned by this instance
    calldll #kernel32, "ReleaseMutex",_
        hMutex as ulong,_
        result as long
```

The CloseHandle function

The fourth function is the one we use to close the handle to our mutex before we end the program, which is the CloseHandle Function
It removes the mutex entirely from the system.

This function must be called at the end of your program to be able to debug and run your program from within the LB editor.
If it happens, that your program crashes, while the mutex is active, you need to restart the LB editor to free

the mutex.

Here is the code to do all of this.

```
    ' close the mutex handle to cleanup
    calldll #kernel32, "CloseHandle",_
        hMutex as ulong,_
        result as long
```

The full example

You can run the example code below multiple times from the LB editor, but only the first run will be fully executed.
Any attempt to run another instance will result in the notice, that there is already an instance running.

Only if you close all running instances, you will be able to fully run another one.

```
    ' inherit the default security attributes
    lpMutexAttributes = _NULL

    ' optain initial ownership by setting this to 1
    bInitialOwner = 1


' use a unique name for the mutex, maximum _MAX_PATH = 260 characters
    lpName$ = "LBMutExDemo"

    ' create the mutex
    calldll #kernel32, "CreateMutexA",_
        lpMutexAttributes as ulong,_
        bInitialOwner     as long,_
        lpName$           as ptr,_
        hMutex            as ulong


' get the last error, so we know if there is already an instance runni
ng
    calldll #kernel32, "GetLastError",_
        LastError as ulong


' check if there is already an instance running (ERROR_ALREADY_EXISTS
= 183)
    if LastError = 183 then
```

```
        ' close the Windows handle obtained
        calldll #kernel32, "CloseHandle",_
            hMutex as ulong,_
            result as long

        ' exit if there is already an instance running
        notice "Exiting!"; chr$(13);_
            "Another instance of this program is already running!"
        end
    end if

    nomainwin
    UpperLeftX = 1
    UpperLeftY = 1

    open "I am the first program instance" for window as #m
    #m "trapclose [quit]"
    wait

[quit]
    close #m

    ' free the mutex so it is no longer owned by this instance
    calldll #kernel32, "ReleaseMutex",_
        hMutex as ulong,_
        result as long

    ' close the mutex handle to cleanup
    calldll #kernel32, "CloseHandle",_
        hMutex as ulong,_
        result as long
    end
```