//

NoGUI

maXbox Starter70 - How to redirect form to a shell.

From Shell to Hell?
HellShell!

This tutor explains a solution to attach a console to your app. Basically we want an app to have two modes, a GUI mode and a non-GUI mode for any humans and robots. A NoGUI app provides a mechanism for storage and retrieval of data and functions in means other than the normal GUI used in operating systems.

From everything you've read this is supposed to work if we use *AttachConsole()*. First we generate the declaration of the 2 DLL's:

```
function AttachConsole(dwProcessID: Integer): Boolean;
              external 'AttachConsole@kernel32.dll stdcall';

function FreeConsole(): Boolean;
              external 'FreeConsole@kernel32.dll stdcall';
```

It attaches the calling process to the console of the specified process and if the function succeeds, the return value is nonzero. A process or app can use the *FreeConsole()* function to detach itself from its console. If other processes share the console, the console is not destroyed, but the same process that called *FreeConsole()* cannot refer to it.
Next we have a function to get the parent process name:

```
function GetParentProcessName(): String;
```

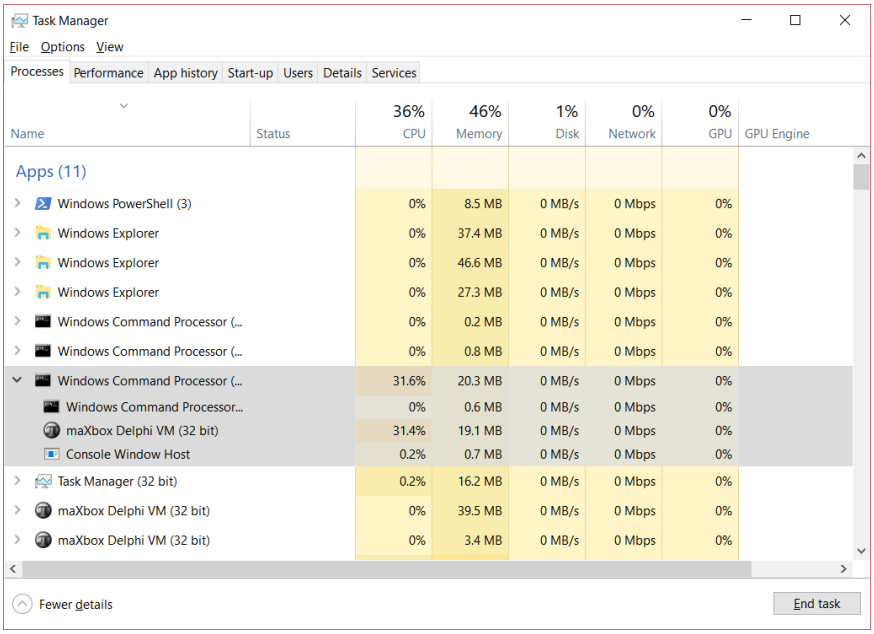
This function needs another DLL from the lib PsAPI:

```
function GetModuleFileNameEx(Handle: THandle; pid: THandle;
                             ppath: Pchar; buffsize: DWORD): DWORD;
              external 'GetModuleFileNameExA@psapi.dll stdcall';
```

This allows me to run my GUI app from a command prompt and display output to the same console where my app was launched. Otherwise, it will run the full GUI part of the app and shows the window as a win- or webform.
The *GetParentProcessName()* asks the command prompt (*powershell* or *cmd*):

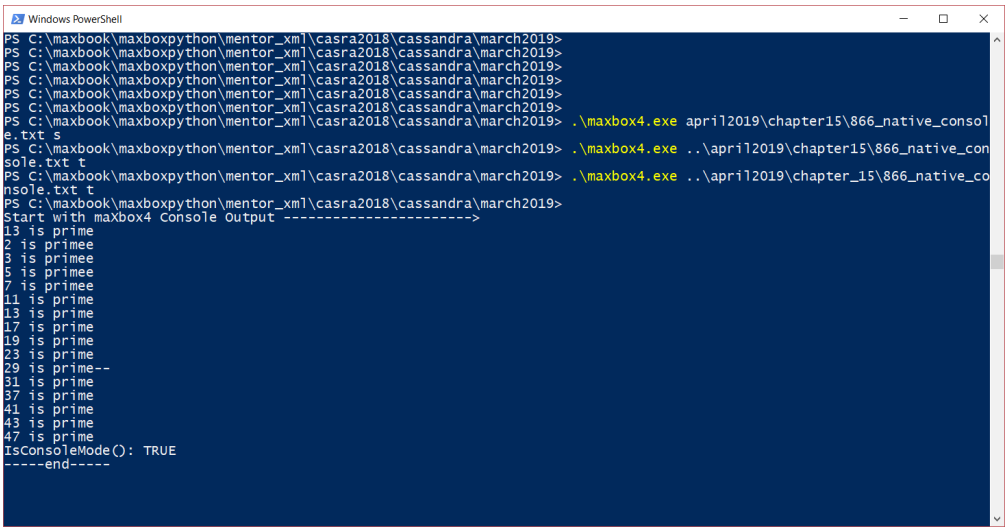
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
C:\Windows\SysWOW64\cmd.exe

There's no reliable way for a GUI subsystem application to attach to the console of its parent process. If you try to do so you end up with two active processes sharing the same console and only one is running:

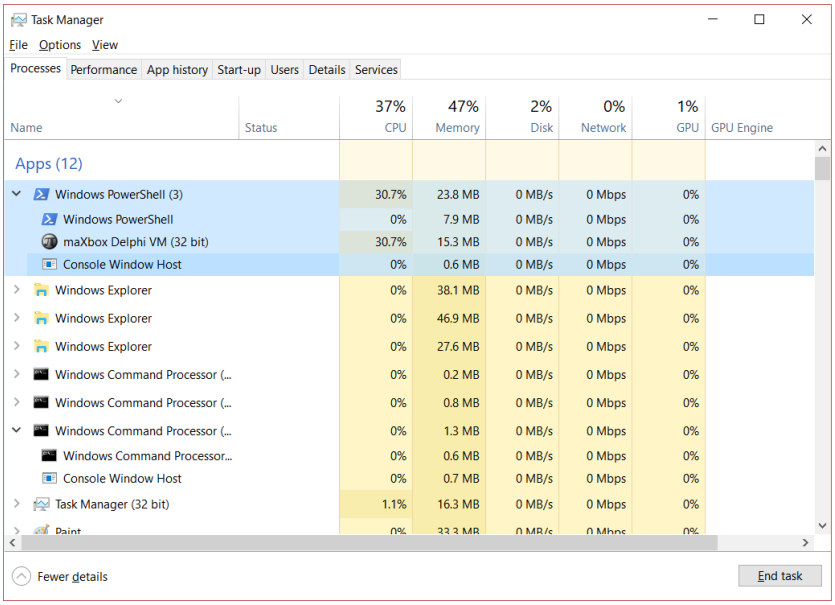


Be careful with this *GetModuleFileNameEx()*. In Win 7, it shows up in the lib *kernel32.dll*, so you might want to code to check for this and load dynamically as I do with *GetModuleFileNameExA()*.
The main part is as follow:

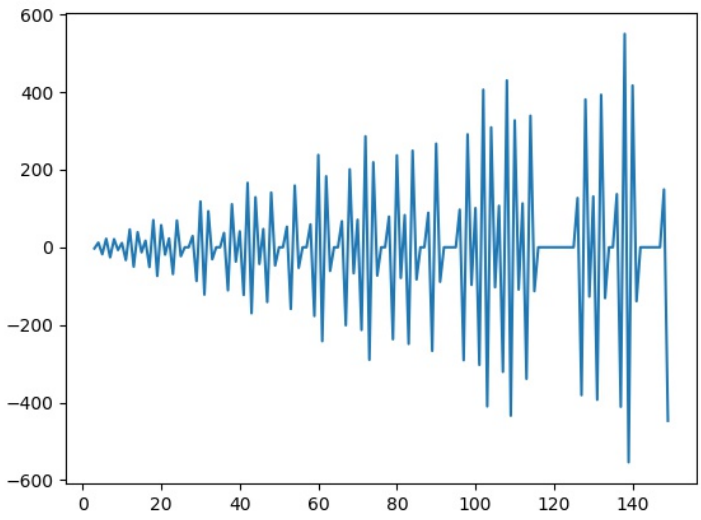
```
ParentName:= strlower(GetParentProcessName());
ParentName:= PathExtractName(ParentName);
Set_ReportMemoryLeaksOnShutdown(false)
if (ParentName='cmd.exe') or (ParentName='powershell.exe') then begin
   AttachConsole(-1);
   NativeWriteln('Start with maXbox4 Console Output--->');
   for it:= 1 to 50 do if IsPrime(it) then
                              NativeWriteln(IntToStr(it)+' is prime');
   NativeWriteln('-----end-----');
FreeConsole();
```



This is OK if you are just wanting to display output into the command line. But operations like redirecting output into a file for example are not working e.g.: start /wait Checker.exe > out.txt would still output into console and not into file out.txt. Different solution exists for the PowerShell:



If you are lost into the source code then you could easily add parameters to your app to write output to a file instead of the console: *-o out.txt*, since it's your tool doing the writing, you can write wherever you want for example to start out of the shell and get output to the shell and in the end plot an image to another file output as a png-graphic like below:



We believe the best option is to create two separate executables or scripts. One for the GUI subsystem, and one for the shell subsystem. This is the approach also taken by:

Java: java.exe, javaw.exe.
Python: python.exe, pythonw.exe.
Visual Studio: devenv.com, devenv.exe.

A console is closed when the last process attached to it terminates or calls *FreeConsole*. After a process calls *FreeConsole*, it can call the *AllocConsole*

function to create a new console or *AttachConsole* to attach to another console.
Call the script from the shell with
>>> .\maxbox4.exe ..\examples\866_native_console.txt

The script can be found:

http://www.softwareschule.ch/examples/866_native_console.txt

Author: Max Kleiner

Ref: http://www.softwareschule.ch/box.htm
https://scikit-learn.org/stable/modules/

Doc: https://maxbox4.wordpress.com

```
function GetParentProcessName(): String;
var
  HandleSnapShot: THandle;
  EntryParentProc: TProcessEntry32;
  CurrentProcessId: THandle;
  HParentProc: THandle;
  ParentProcessId: THandle;
  ParentProcessFound: Boolean;
  ParentProcPath: String;
begin
  ParentProcessFound:= False;
  HandleSnapShot:= CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
  if HandleSnapShot<>INVALID_HANDLE_VALUE then
  begin
    EntryParentProc.dwSize:= SizeOf(EntryParentProc);
    if Process32First(HandleSnapShot,EntryParentProc) then
    begin
      CurrentProcessId:= GetCurrentProcessId();
      repeat
        if EntryParentProc.th32ProcessID=CurrentProcessId then begin
          ParentProcessId:= EntryParentProc.th32ParentProcessID;
          HParentProc:= OpenProcess(PROCESS_QUERY_INFORMATION or
                PROCESS_VM_READ,False,ParentProcessId);
          if HparentProc<> 0 then begin
            ParentProcessFound:= True;
            SetLength(ParentProcPath,BufferSize);
            GetModuleFileNameEx(HParentProc,0,PChar(ParentProcPath),BufferSize);
            ParentProcPath:= PChar(ParentProcPath);
            CloseHandle(HParentProc);
          end;
          Break;
        end;
      until not Process32Next(HandleSnapShot,EntryParentProc);
    end;
    CloseHandle(HandleSnapShot);
  end;
  if ParentProcessFound then Result:= ParentProcPath
  else Result:= '';
end;
```

```
>>> from geopy.geocoders import Nominatim
>>> geolocator = Nominatim('maxbox-app')
>>> place,(lat,lng) = geolocator.geocode("Breitenrainplatz 2 Bern")
>>> print ("%s: %.5f, %.5f" % (place, lat, lng))
```