Capture Console Output To Memo in RealTime

The below code captures console outputs and send it in real time to callback function, so the application can handle it such as displaying its progress by adding to memo, so end user may not think the application is hanging while program performing the command.

```
procedure TMainFrm.CaptureConsoleOutput(const ACommand: String; CallBack: TArg<PAnsiChar>);
const
    READ_BUFFER_SIZE = 2400;
var
    Security: TSecurityAttributes;
    piEncoder: TProcessInformation;
    readableEndOfPipe, writeableEndOfPipe: THandle;
    start: TStartUpInfo;
    Buffer: PAnsiChar;
    BytesRead: DWORD;
    AppRunning: DWORD;

    tmpSL:TStringList;
    tmpS:string;
begin
    tmpS := GetEnvironmentVariable('TEMP') + '\' + 'CKJUNPA.BAT';
    tmpSL := TStringList.Create;
    tmpSL.Add(ACommand);

    // tmpSL.SaveToFile(tmpS, TEncoding.UTF8); - bat file seems correct, but does not work
    // tmpSL.SaveToFile(tmpS, TEncoding.ANSI); - multibyte character is incorrectly written on batch file
    tmpSL.SaveToFile(tmpS, TEncoding.ANSI);

    Security.nLength := SizeOf(TSecurityAttributes);
    Security.bInheritHandle := True;
    Security.lpSecurityDescriptor := nil;

    if CreatePipe({var}readableEndOfPipe, {var}writeableEndOfPipe, @Security, 0) then
    begin
        Buffer := AllocMem(READ_BUFFER_SIZE+1);
        FillChar(Start, Sizeof(Start), #0);
        start.cb := SizeOf(start);

        // Set up members of the STARTUPINFO structure.
        // This structure specifies the STDIN and STDOUT handles for redirection.
        // - Redirect the output and error to the writeable end of our pipe.
        // - We must still supply a valid StdInput handle (because we used STARTF_USESTDHANDLES to swear that
all three handles will be valid)
        start.dwFlags := start.dwFlags or STARTF_USESTDHANDLES;
        start.hStdInput := GetStdHandle(STD_INPUT_HANDLE); //we're not redirecting stdInput; but we still have
to give it a valid handle
        start.hStdOutput := writeableEndOfPipe; //we give the writeable end of the pipe to the child process;
we read from the readable end
        start.hStdError := writeableEndOfPipe;

        //We can also choose to say that the wShowWindow member contains a value.
        //In our case we want to force the console window to be hidden.
        start.dwFlags := start.dwFlags + STARTF_USESHOWWINDOW;
        start.wShowWindow := SW_HIDE;

        // Don't forget to set up members of the PROCESS_INFORMATION structure.
        piEncoder := Default(TProcessInformation);

        //WARNING: The unicode version of CreateProcess (CreateProcessW) can modify the command-line "DosApp"
string.
        //Therefore "DosApp" cannot be a pointer to read-only memory, or an ACCESS_VIOLATION will occur.
        //We can ensure it's not read-only with the RTL function: UniqueString

        if CreateProcess(nil, PChar(tmpS), nil, nil, True, NORMAL_PRIORITY_CLASS, nil, nil, start, {var}
piEncoder) then
        begin
            //Wait for the application to terminate, as it writes it's output to the pipe.
            //WARNING: If the console app outputs more than 2400 bytes (ReadBuffer),
            //it will block on writing to the pipe and *never* close.
            repeat
```

```
                Apprunning := WaitForSingleObject(piEncoder.hProcess, 100);
                Application.ProcessMessages;

                //Read the contents of the pipe out of the readable end
                //WARNING: if the console app never writes anything to the StdOutput, then ReadFile will block
and never return

                repeat
                  BytesRead := 0;
                  ReadFile(readableEndOfPipe, Buffer[0], READ_BUFFER_SIZE, {var}BytesRead, nil);

                  if BytesRead>0 then
                  begin

                    Buffer[BytesRead]:= #0;
                    OemToAnsi(Buffer,Buffer);

                    // added for debugging ------------------------------------------
                    KDEBUG( string(Buffer));
                    // end of debugging ---------------------------------------------
                    CallBack(Buffer);
                  end;

                until BytesRead<READ_BUFFER_SIZE;

            until (Apprunning <> WAIT_TIMEOUT);
        end;
        FreeMem(Buffer);
        CloseHandle(piEncoder.hProcess);
        CloseHandle(piEncoder.hThread);
        CloseHandle(readableEndOfPipe);
        CloseHandle(writeableEndOfPipe);
    end;
    tmpSL.Destroy;
end;
```

Example)

```
var
  strDir, strFFMPEG: string;
begin
  strDir := GetCurrentDir;
  strFFMPEG := '"' + strDir + '\ffmpeg.exe"';
  strFFMPEG := strFFMPEG + ' -y -i C:\test.mkv -vcodec h264 -acodec aac -f mp4 C:\test-2.mp4';
  Memo1.Lines.Add(strFFMpeg);
  CaptureConsoleOutput(strFFMPEG,
            procedure(const Line: PAnsiChar)
            begin
                //Panel1.Caption := String(Line);
                Memo1.Lines.Add( String(Line));
            end
 );
end;
```