Delphi TExecFile Component 7/95 - Release 1.0

The TExecFile is a non-visual component which you place onto your form to enable easily initiated execution of other Windows or DOS applications from within your own original Delphi application.
Although it seems that running another application from within your own is no great task, I have noticed many people asking how to do it. Having seen some code posted in one of the forum sections which showed how to 'execute and wait', I decided that it would be a good idea to incorporate this capability using a slight variation of that code into a simple yet useful component. I think you will also find that use of this component beats fumbling around with code or special functions each time you wish to run an outside application. It may also be useful for chaining your Delphi application or executing multiple modules belonging to the same application. It is also simple enough so that it does not overlap or delve into any of the functions available through the use of DDE.

A glance at the properties and events of the TExecFile, once installed, will more than likely explain it's use. Nevertheless, I have included detailed explanations of all it's facets for reference purposes in the following documentation.

To install the TExecFile, copy the EXECFILE.DCU and EXECFILE.DCR into your \DELPHI\LIB directory and install it as you would any other component. It will appear in the Samples section of your component palette.

There are seven component properties:

FileName (string)
Name of file you wish to execute (.EXE, .PIF, etc.). *This will also be subject to Windows file extension associations.*
Examples: MYFILE.EXE
C:\BATCHES\BATCHRUN.PIF
A:\LIST.TXT
Can be set at design time and/or run time.

FilePath (string)
Default directory for file, aka working directory.
Example: C:\MYDIR\
Can be set at design time and/or run time.

Name (string)
Name of your TExecFile component.
Example: ExecFile1

Parameters (string)
Any parameters you wish to pass to the executable file named in the FileName property.
Example: -p C:\PDOXAPPS
Can be set at design time and/or run time.

Tag
Not used.

Wait (boolean)
If set to *True,* execution of your procedure will pause until the executed file is closed. If set to *False*, file is executed, and procedure continues without pausing. See 'Component Behavior' below for details.
Can be set at design time and/or run time.

WindowStyle (constant)
Window style you wish the executed file to possess.
Choices for this property are:
wsNormal -- display Window in normal position
wsMinimize -- display Window as an icon

wsMaximize -- display Window as full screen or maximized
wsHide -- run executable as hidden in background
(note: Be careful that if this is used, the application you are running does not require user interaction in order to execute and terminate itself.)
wsMinNoActivate -- display Window as an icon, however, current Window remains active.
wsShowNoActivate -- display Window normally, however, current Window remains active.

Can be set at design time and/or run time.

There is one component event:

OnFail Code that you place here is triggered when the execution of the specified file fails, for reasons such as file not found, invalid path, disk error, file is not executable, etc.

There are four component functions:

Execute (boolean) Commence file execution based on the current property settings. This function returns a boolean value (*False* if execution failed, *True* if it was successful. See 'Component Behavior' below for details.)

IsWaiting (boolean) Returns a boolean value of *True* if TExecFile is currently waiting on an executed file, *False* if it is not.

StopWaiting (boolean) Returns *True* to acknowledge abort of waiting on an executed file. Used in conjunction with the TExecFile.IsWaiting function and/or the TExecFile.Wait property. See 'Component Behavior' below for details.

ErrorCode (longInt) Returns a longInt representing a specific type of error which occurred during an attempted file execution. If this is >32 then execution was successful. The following is a list of possible return values of the ErrorCode function:

Value	*Description*
0	System was out of memory, executable file was corrupt, or relocations were invalid.
2	File was not found.
3	Path was not found.
5	Attempt was made to dynamically link to a task, or there was a sharing or network-protection error.
6	Library required separate data segments for each task.
8	There was insufficient memory to start the application.
10	Windows version was incorrect.
11	Executable file was invalid. Either it was not a Windows application or there was an error in the .EXE image.
12	Application was designed for a different operating system.
13	Application was designed for MS-DOS 4.0.
14	Type of executable file was unknown.
15	Attempt was made to load a real-mode application (developed for an earlier version of Windows).
16	Attempt was made to load a second instance of an executable file containing multiple data segments that were not marked read-only.

19 Attempt was made to load a compressed executable file. The file must be decompressed before it can be loaded.
20 Dynamic-link library (DLL) file was invalid. One of the DLLs required to run this application was corrupt.
21 Application requires Windows 32-bit extensions.
31 No association for specified file.

Examples of use:

To use the TExecFile, simply place the component onto your form, and access it based on the following examples:

To set properties at runtime (assuming default name is used):

```
ExecFile1.FileName := 'MYFILE.PIF';      {Set file to be executed as MYFILE.PIF}
ExecFile1.FilePath := 'C:\MYDIR\';       {Set path of MYFILE.PIF as C:\MYDIR\}
ExecFile1.WindowStyle := wsNormal;       {Set Window display to normal}
ExecFile1.Wait := True;                  {Set wait for file to be closed before continuing}
```

To execute the file specified by the property settings:

```
ExecFile1.Execute;                       {Execute file based on current property settings}
```

Component Behavior:

When executing a file with the TExecFile.Wait property set to *True*, the TExecFile will more or less 'keep an eye' on the application which is running (also called an 'instance'). In fact, once the TExecFile.Execute function is called, it, along with the procedure within your application containing it, will wait for the instance to be closed before continuing. In addition to waiting, it does not disable user interaction with Windows, your application, or other applications in the current Windows session.

The TExecFile.Execute function will return a boolean value (*True* or *False*) depending on whether or not the file specified in it's properties was successfully executed. If execution fails for any reason (file not found, disk error, file was not a valid executable, etc.) the TExecFile.Execute function will return *False*. When executing a file with the TExecFile.Wait property set to *True*, the TExecFile.Execute function will return *True* ONLY when the called instance is closed. The following example shows this:

```
procedure TForm1.SpeedButton3Click(Sender: TObject);
begin

ExecFile1.Wait := True;                                  {Set TExecFile to wait}

If not ExecFile1.IsWaiting then begin                    {Proceed only if TExecFile is not
                                                          currently waiting on a called app}
If ExecFile1.Execute then                                {Begin file execution and wait here.}
 Label1.Caption := 'Application successfully executed.'  {This code executes upon closing the
                                                          called application}
else
 Label1.Caption := 'Execution Failed';                   {This code executes if the called
                                                          application could not be executed,
                                                          This could also be placed in the
                                                          OnFail procedure of ExecFile1.}
```

```
end
else
 Label1.Caption := 'Application is already running.';        {This code executes if the TExecFile
                                                             is already waiting on an application}
end;
```

Note that at the waiting point above, if focus is returned to your application prior to closing the called instance, your application will function normally, while the procedure containing the TExecFile.Execute function is still waiting for the instance to close. If your application is closed prior to the instance, the instance will remain open until it is otherwise closed by the user or another application, while your application will terminate normally.

In addition to returning the boolean *False* when execution fails, code that you place in the TExecFile.OnFail event will execute. This may be a useful area to place code informing the user that the file could not be executed, or to otherwise handle the failure. You could also use this event to perform your own testing in order to determine the reason for failure and communicate this to other areas of your application, or to the user. The TExecFile provides a means by which to determine the cause of failure, using the TExecFile.ErrorCode function. A call to TExecFile.ErrorCode will return a longInt specific to the reason for failure. Possible ErrorCodes are listed in the 'Component Functions' section of this documentation. An example of the use of TExecFile.ErrorCode:

```
procedure TForm1.ExecFile1Fail(Sender: TObject);
begin
If ExecFile1.ErrorCode = 2 then
Label1.Caption := 'File Not Found';
If ExecFile1.ErrorCode = 3 then
Label1.Caption := 'Path Not Found';
end;
```

Also, if your application opens more than one instance using the same TExecFile component with the TExecFile.Wait property set to *True*, the first instance remains open, however the TExecFile will no longer 'keep an eye' on the previously executed instance (which will remain open until otherwise closed by the user or another application). In other words, TExecFile will only 'keep an eye' and wait on its most recently executed instance.

Multiple TExecFile components may be used, if desired, in order to monitor multiple instances by using the TExecFile.Execute function within seperate procedures. However, in doing so, the procedure containing the TExecFile.Execute will resume execution only when the called instances have been closed in the reverse order of that which they were opened (similar to a 'stack').

An example of using 3 seperate buttons on a form to execute and wait on 3 different applications simultaneously is shown below in Diagram 1.

Diagram 1:

MYAPP

Button1	----->	ExecFile1	----->	Running APP1.EXE
Button2	----->	ExecFile2	----->	Running APP2.EXE
Button3	----->	ExecFile3	----->	Running APP3.EXE

Example:

The user, in this order, first clicks Button1, and APP1 is executed...then clicks Button2 and APP2

is executed...then clicks Button3 and APP3 is executed. MYAPP is now waiting for 3 applications to close. In this case, MYAPP will not acknowledge the closing of APP2 by resuming procedure execution for Button2 until APP3 is closed, since APP3 was the most recently executed file. Similarly, MYAPP would not acknowledge the closing of APP1 by resuming procedure execution for Button1 until BOTH APP2 and APP3 are closed. The closing of APP3, since it was the most recently executed file, would be acknowledged immediately by MYAPP, and procedure for Button3 would resume at the time it is closed. If APP1 and APP2 are closed, followed by APP3, procedures for Button3, Button2, and Button1 would resume at the point in time that APP3 is closed (in that order).

This can be explained easily:

1. The procedure in Button1 was halted by ExecFile1, which is in a loop, getting and dispatching Windows messages (mouse clicks, keystrokes, etc.), and waiting for a signal from APP1 that it has closed.

2. The user invokes the procedure in Button2, which branches from the loop of ExecFile1, and activates ExecFile2, which begins getting and dispatching Windows messages, and waiting for a signal from APP2 that it has closed. In the meantime, ExecFile1 cannot receive the signal from APP1 if it closes, since it's loop has temporarily stopped because of branching to ExecFile2. ExecFile1 would not receive the signal from APP1 in this case, until returning from the branch to ExecFile2.

3. The user invokes the procedure in Button3, which branches from the loop of ExecFile2, and begins getting and dispatching Windows messages, and waiting for a signal from APP3 that it has closed. In the meantime, ExecFile2 AND ExecFile1 cannot receive the signal from their APP's if they close, since thier loops have temporarily stopped because of consecutive branching to ExecFile3.

4. If, at this point, APP1 and APP2 are closed, the unique handle assigned to them indicates that they are closed, and indication of this is pending return to their corresponding TExecFile components from ExecFile3. ExecFile1 and ExecFile2 will not be aware of this until APP3 is closed, and control is returned first to ExecFile2, which immediately recognizes that APP2 has closed, and then returned to ExecFile1, which recognizes that it's app (APP1) has closed as well.

The TExecFile.StopWaiting function can be used in cases that you wish a procedure within your application to tell a TExecFile to stop waiting on it's executed application, and resume execution of the procedure which called it. The TExecFile.StopWaiting function does not affect the execution of the application which was called. It will remain open until closed by the user or another application. The TExecFile's Execute function will return *True* if it's StopWaiting function is called just as it would if the application it had called was closed.

When running DOS applications using the TExecFile component, a Windows .PIF (Program Information File) should be utilized. This will allow the WindowStyle property to govern the method of display of your DOS application, as well as assist in avoiding some of the other problems which may be associated with not using a .PIF.

This component is free, and a reasonable amount of time has gone into it's development and testing. If you have questions or find a malfunction, or have any general comments concerning the TExecFile, feel free to drop me a message here or via Internet: bolt@gcomm.com

Kevin Savko
75024,2760