

Application Note



5800 Series- Executing External Programs

Reference: AN503-02



The Aeroflex Integrated Development Environment (AIDE) provides a comprehensive programming platform capable of controlling a vast array of devices in addition to the 5800 Automatic Test Equipment (ATE) hardware.

These notes describe how External Programs may be Executed and Monitored from the AIDE in a .NET environment.

Functionality

The AIDE has full access to the .NET Framework. Within that framework the Process Class provides all the required functionality to start, stop, control and monitor external applications. See the .NET Framework SDK Documentation for a full description.

With the Process component, the user can obtain a list of the processes that are running or start a new process. A Process component is used to access system processes. After a Process component has been initialized, it can be used to obtain information about the running process. Such information includes the set of threads, the loaded modules (.dll and .exe files) and performance information such as the amount of memory the process is using.

Program Example 1 – Starting another Program from within the AIDE

Create a new Program called, for example, AIDEExec.

Detailed information on the Framework SDK can be obtained from the installed documentation by selecting:-

```
Start\All Programs\Microsoft .NET Framework SDK vX.Y\Documentation
```

Configuration Data

Expand the Configuration Data section of the program and add the following .NET Assembly References from the .NET Global Assembly Cache (GAC) using the browse facilities provided:-

```
.NET Assembly Reference AssemblyFile:microsoft.dll
.NET Assembly Reference AssemblyFile:System
```

Add the following .NET Namespace References:-

```
.NET Namespace Reference System
.NET Namespace Reference System.Diagnostics
```

Global Symbols

Create a new Global Variable of type Process called process1 and assign it an InitialValue = null. The available data types can be found and selected by clicking the button in the Type property window and pressing the More Types... button and scrolling the subsequent display contents until the required data type can be seen and selected.

```
Variable [ Process] process1 InitialValue = null
```

Main Method

Create a new Method called Main and reference it in the Program OnStart Property.

We can now create Evaluate statements that can be used to control execution of external programs.

Simple Program Execution

To Execute an external test program, insert a new Evaluate Method Code statement as follows:-

```
Evaluate process1 = new Process()
```

This statement Creates a new Process object to manage the executable.

We now require to specify the program (e.g. Notepad.exe) to be executed and any parameters to be supplied by inserting statements of the form:-

```
Evaluate process1.StartInfo.FileName = "notepad.exe"
Evaluate process1.Start()
Evaluate process1.Close()
```

When executed, the above code starts up Windows Notepad in its default state (editing an 'untitled' document), which then runs totally independently of the AIDE. The process1.Close() statement disposes of the AIDE process1 object but DOES NOT affect the execution of the spawned process itself.

More Complex Program Execution

To Execute an external test program that requires parameters and WAIT until that program terminates we modify the code as follows:-

```
Evaluate process1.StartInfo.FileName = "notepad.exe"
Evaluate process1.StartInfo.WorkingDirectory = @"c:\tmp" Comment "the '@' directive forces correct interpretation of single '\' characters in a directory PATH"
Evaluate process1.StartInfo.Arguments = "MyFile.txt"
Evaluate process1.Start()
Evaluate process1.WaitForExit()
Evaluate process1.Close()
```

When executed, the above code starts up Windows Notepad, setting the working directory to "C:\tmp" and editing file "MyFile.txt". The AIDE then WAITS until the spawned process terminates before proceeding with further execution.

Parallel Program Execution

In general, to Execute one or more external test programs that require parameters, continue with execution and then WAIT until those programs terminate we add as many Process variables and AIDE Methods as necessary and modify the code as follows:-

```
Evaluate process1.StartInfo.FileName = "app1.exe"
Evaluate process1.StartInfo.WorkingDirectory = "app1 working directory"
Evaluate process1.StartInfo.Arguments = "app1 arguments"
Evaluate process1.Start()

...

Evaluate process2.StartInfo.FileName = "app2.exe"
Evaluate process2.StartInfo.WorkingDirectory = "app2 working directory"
Evaluate process2.StartInfo.Arguments = "app2 arguments"
Evaluate process2.Start()

...

Evaluate AIDEMethodA()
Evaluate AIDEMethodB()
```

```
Evaluate AIDEMethodC()
...
Evaluate process1.WaitForExit()
Evaluate process1.Close()
Evaluate process2.WaitForExit()
Evaluate process2.Close()
```

When executed, the above code starts up app1.exe with appropriate parameters, starts up app2.exe with appropriate parameters and continues execution, including running AIDEMethodA, AIDEMethodB and AIDEMethodC. The AIDE then WAITS until firstly, process1 terminates, then secondly process2 terminates before proceeding with further execution.

Testing for both processes terminating at once can be accomplished with the following statement.

```
While (!process1.HasExited && !process2.HasExited)
```

The user should refer to the .NET SDK Documentation for a full description of the Properties and Methods made available with the Process object.

Program Example 2 – Using Common Methods

The commands employed to handle external processes tend to be largely the same and so can be generalised into a number of common AIDE Methods.

Common Methods

Create a new Method called StartProcess defined as returning an object of type Process and taking two Method Parameters of type String as follows.

```
Method StartProcess [ Process ] ([ String ]
executable , [ String ] args )
Local Symbols
Method Parameters
Method Parameter [ String ] executable
Comment = "the EXE file name"
Method Parameter [ String ] args
Comment = "the arguments"
```

Variables

```
Variable [ Process ] process
Comment = "the Process object to be created"
```

Add the following .NET Namespace reference at the start of the program.

```
.NET Namespace Reference System.IO
```

Method code to handle the creation of the Process object is added to the StartProcess as follows.

```
Evaluate process = new Process()
Evaluate process.StartInfo.FileName = executable
Evaluate process.StartInfo.WorkingDirectory =
Path.GetDirectoryName(executable)
Evaluate process.StartInfo.Arguments = args
Evaluate process.StartInfo.UseShellExecute =
false
```

```
Evaluate process.StartInfo.CreateNoWindow = true
Evaluate process.StartInfo.RedirectStandard
Error=true
Evaluate process.Start()
Return process
```

A single command can now be employed to start an external process, namely.

```
Evaluate process1=StartProcess("notepad.exe",@"
c:\tmp\MyText.txt")
```

Create a new Method called WaitForProcess defined as returning nothing and taking a single Method Parameter of type Process as follows.

```
Method WaitForProcess [ ] ([ Process ] process )
Local Symbols
Method Parameters
Method Parameter [ Process ]
process
Comment = "the process
to be monitored"
```

Method code to handle the monitoring of the Process until it terminates is added as follows.

```
Evaluate process.WaitForExit()
Evaluate process.Close()
```

A single command can now be employed to monitor for an external process terminating, namely.

```
Evaluate WaitForProcess(process1)
```

Parallel Program Execution

The previous generalised Parallel Processing command sequence can now be simplified as follows.

```
Evaluate process1 = StartProcess ( "app1.exe" ,
"app1 command line parameters")
Evaluate process2 = StartProcess ( "app2.exe" ,
"app2 command line parameters")
```

...

```
Evaluate AIDEMethodA()
Evaluate AIDEMethodB()
Evaluate AIDEMethodC()
```

...

```
Evaluate WaitForProcess(process1)
Evaluate WaitForProcess(process2)
```

...

Obtaining External Program Results

Where an external executable returns a result, an alternative Method may be created.

Create a new Method called WaitForProcessResult defined as returning a String and taking a single Method Parameter of type Process as follows.

```
Method WaitForProcessResult [ String ] ([
Process ] process )
Local Symbols
```

Method Parameters

```
Method Parameter [ Process ] process  
Comment = "the process to be monitored"
```

Variables

```
Variable [ String ] resultString Comment = "the  
Process return value (if any)"
```

Method code to handle the monitoring of the Process object and obtain the return data is added as follows.

```
Evaluate process.WaitForExit()  
Evaluate resultString = process.StandardError.  
ReadToEnd()  
Evaluate process.Close()  
Return resultString
```

A single command can now be employed to wait for an external process to terminate and obtain the returned data namely.

```
Evaluate resultData = WaitForProcessResult  
(process1)
```

Where resultData is a previously defined String Variable.

When an external executable returns a numeric Exit Code, another alternative Method may be created.

Create a new Method called WaitForProcessExitCode defined as returning an Integer and taking a single Method Parameter of type Process as follows.

```
Method WaitForProcessExitCode [ Int ] ( [  
Process ] process )  
Local Symbols  
Method Parameters  
Method Parameter [ Process ] process  
Comment = "the process to be monitored"  
Variables  
Variable [ Int ] exitCode Comment = "the  
Process exit code"
```

Method code to handle the monitoring of the Process object and obtain the return data is added as follows.

```
Evaluate process.WaitForExit()  
Evaluate exitCode = process.ExitCode  
Evaluate process.Close()  
Return exitCode
```

A single command can now be employed to wait for an external process to terminate and obtain the returned data namely.

```
Evaluate exitData = WaitForProcessExitCode  
(process1)
```

Where exitData is a previously defined Int Variable.

For successfully completed processes the return value is usually ZERO. Negative values usually indicate an ERROR. The user must check with the description or specification of the external application to determine the meaning of any return codes.

Forced Termination

In desperation the user might wish to DESTROY an external process. This is achieved by executing the following statement.

```
Evaluate process.Kill()
```

A more appropriate means of requesting a spawned Windows application to terminate is provided by the following method.

```
Evaluate process.CloseMainWindow()
```

Create a new Method called ForceProcessExitCode defined as returning an Integer and taking a single Method Parameter of type Process as follows.

```
Method ForceProcessExitCode [ Int ] ( [ Process ]  
process )
```

Local Symbols

Method Parameters

```
Method Parameter [ Process ]  
process Comment = "the process to  
be monitored"
```

Variables

```
Variable [ Int ] exitcode Comment  
= "the Process exit code"
```

Method code to handle the monitoring of the Process object and obtain the return data is added as follows.

```
Evaluate process.CloseMainWindow()  
Evaluate process.WaitForExit()  
Evaluate exitCode = process.ExitCode  
Evaluate process.Close()  
Return exitCode
```

A single command can now be employed to tell an external process to terminate and obtain the returned data namely.

```
Evaluate exitData = ForceProcessExitCode  
(process1)
```

The user is again referred to the .NET SDK Documentation for a full description of the Properties and Methods made available with the Process object.

For the very latest specifications visit www.aeroflex.com

For the very latest specifications visit www.aeroflex.com

CHINA Beijing

Tel: [+86] (10) 6539 1166
Fax: [+86] (10) 6539 1778

CHINA Shanghai

Tel: [+86] (21) 5109 5128
Fax: [+86] (21) 5150 6112

CHINA Shenzhen

Tel: [+86] (755) 3301 9358
Tel: [+86] (755) 3301 9356

FINLAND

Tel: [+358] (9) 2709 5541
Fax: [+358] (9) 804 2441

FRANCE

Tel: [+33] 1 60 79 96 00
Fax: [+33] 1 60 77 69 22

GERMANY

Tel: [+49] 89 99641 0
Fax: [+49] 89 99641 160

HONG KONG

Tel: [+852] 2832 7988
Fax: [+852] 2834 5364

INDIA

Tel: [+91] 80 [4] 115 4501
Fax: [+91] 80 [4] 115 4502

JAPAN

Tel: [+81] (3) 3500 5591
Fax: [+81] (3) 3500 5592

KOREA

Tel: [+82] (2) 3424 2719
Fax: [+82] (2) 3424 8620

SCANDINAVIA

Tel: [+45] 9614 0045
Fax: [+45] 9614 0047

SINGAPORE

Tel: [+65] 6873 0991
Fax: [+65] 6873 0992

UK Stevenage

Tel: [+44] (0) 1438 742200
Fax: [+44] (0) 1438 727601
Freephone: 0800 282388

USA

Tel: [+1] (316) 522 4981
Fax: [+1] (316) 522 1360
Toll Free: 800 835 2352

As we are always seeking to improve our products, the information in this document gives only a general indication of the product capacity, performance and suitability, none of which shall form part of any contract. We reserve the right to make design changes without notice. All trademarks are acknowledged. Parent company Aeroflex, Inc. ©Aeroflex 2011.

www.aeroflex.com
info-test@aeroflex.com



Our passion for performance is defined by three attributes represented by these three icons: solution-minded, performance-driven and customer-focused.