www.IT-Visions.de
Diplom-Wirtschaftsinformatiker
Microsoft Certified Solution Developer
Holger Schwichtenberg

1 DOTNET Scripting Host (DSH) 1.1

Author of the original document in German: Holger@Schwichtenberg.de
Translation: Maria Schnurr
Date: 11/20/2002
DSH Version: 1.1.3

DSH

The DOTNET Scripting Host (DSH) is an Active Scripting Host for the .NET framework and simulates the basic functions of the Windows Script Host (WSH) for the Common Language Runtime.

DSH features are:

- Launching scripts at the command line and with double click
- Passing parameters to the script
- Packaging (multiple) scripts in a XML structure
- Using different languages in one script document
- Support for Visual Basic .NET, C# and JScript .NET
- Using components (.NET Framework Class Library classes and other .NET assemblies)
- Remote execution of scripts (Remote Scripting)
- Optional logging to text files and the Windows event log
- Optional saving of compiled script as an executable

Background

DSH is a freeware and is available in its version 1.1 at the time on the following Internet site: www.windows-scripting.de Holger Schwichtenberg is the author of DSH (hs@IT-Visions.de)

compilation

DSH is implemented in DSH.EXE. Copy the file onto your system. If you wish to link the file extension .DSH with DSH.EXE please enter the path to DSH.EXE in the included file INSTALLDSH.REG. Then start the file in order to perform the registry.

Installation

Freeware

Languages

All the messages of the DSH are in German in case your system language is German. In all other cases the messages will be in English.

Instead of using Script for .NET/Visual Studio for Applications (VSA) DSH directly uses the compiler integrated in the .NET Framework Class Library (FCL) also know als Code Document Object Model (CodeDOM). Working with the CodeDOM the DSH can compile the source code file and then execute the assembly thus generated. The assembly is generated in memory but not saved on the hard disk (standard mode). If requested, the assembly can be stored in the file system. DSH users do not know that the program is not interpreted but compiled instead. At the moment DSH does only support the .NET compatible programming languages Visual Basic .NET, C# and Jscript .NET.

1.1 File format

DSH uses XML as file format because thus it is possible to transfer additional information besides the actual script to the compiler, e.g. names of the languages used and the referenced assemblies.

A script document may consist of several scripts which are executed sequentially. There is no predefined file extension. Normally the extension .DSH will be used.

<table>
<tr><th>XML-element</th><th>Allowed frequency</th><th>Comment</th></tr>
<tr><td><?xml version="1.1" encoding="ISO-8859-2"?></td><td>1</td><td>Processing Instruction</td></tr>
<tr><td><scriptdoc></td><td>1</td><td>Root element framing all subordinate elements.</td></tr>
<tr><td><comment></td><td>0..N</td><td>Element of comment, ignored by DSH.</td></tr>
<tr><td><references></td><td>0/1</td><td>Combines one or more <assembly> elements.</td></tr>
<tr><td><assembly></td><td>0/1</td><td>Defines an assembly which has needs to be referenced</td></tr>
<tr><td><log></td><td>0/1</td><td>Defines which DSH messages are written down in a result protocol and which ones in a text file.
<table>
<tr><th>Attribute</th><th>Comment</th></tr>
<tr><td>Error</td><td>Optional attribute If the attribute exists and is not turned on “NO” a default protocol entry is created.</td></tr>
<tr><td>Success</td><td>Optional attribute If the attribute exists and is not turned on “NO” a protocol entry is created after finishing of a script.</td></tr>
<tr><td>Start</td><td>If the attribute exists and is not turned on “NO” a protocol entry is created upon starting a script.</td></tr>
</table></td></tr>
<tr><td><eventlog></td><td>0/1</td><td>Subordinate element of <log>.
Defines the name of the result protocol for the attribute name that has to appear in the entries. Except for the standard protocols (e.g. Application, System and Security) it is possible to use any protocol name. The DSH automatically creates the appropriate result protocol.</td></tr>
<tr><td><logfile></td><td>0/1</td><td>Subordinate element of <log>
Defines the path of a text file for the attribute name that has to appear in the protocol entries.</td></tr>
<tr><td><script></td><td>1..N</td><td>Source code of the script. It is possible to have as many elements as the user wishes. The scripts are executed in the order they appear.
<table>
<tr><th>Attribute</th><th>Comment</th></tr>
<tr><td>Name</td><td>Optional attribute
Script name</td></tr>
<tr><td>Language</td><td>Necessary attribute
Possible attribute values: "VB.NET", "CSharp" and "JS.NET".</td></tr>
<tr><td>startClass</td><td>Optional attribute
Name of the class that is the set-off mark and starts the script with its subroutine. (In the standard mode a class with the name Main is searched for, i.e. Main::Main().)</td></tr>
</table></td></tr>
</table>

Table 1: XML elements in DSH scripts

Beispiel

The following listing gives an example with three scripts: ***Example***

- The first script (CSharp) issues Hello World.
- The second script (VB.NET) excerpts the character string “Hello World” from an XML file.
- The third script (VB.NET) generates a list with the scripts parameters.

```
<?xml version="1.1" encoding="ISO-8859-2"?>
<scriptdoc process="1">

<comment>
Demo-Skript von Holger Schwichtenberg
</comment>

<references>
<assembly>system.xml.dll</assembly>
<assembly>System.DirectoryServices.dll</assembly>
<assembly>system.data.dll</assembly>
</references>

<log error="YES" success="YES" start="YES">
<eventlog1 name="Scripting"/>
<logfile1 name="e:\log.txt"/>
</log>

<!-- ************************************** -->

<script name="FirstScript" language="CS" startClass="Scripting.StartKlasse">
using System;
namespace Scripting
{
	class StartKlasse
	{
		static void Main(string[] args)
		{
			Console.WriteLine("FirstSkript: Hello World!");
		}
	}
}
</script>

<!-- ************************************** -->

<script name="SecondScript" language="vb.net">
```

```
<![CDATA[
Imports System.XML
Imports System

Module Main
    Sub Main()

        ' --- Test XML
       Dim d as new XmlDocument
        d.loadxml("<test><message>Hello World!</message></test>")
       Console.Writeline("SecondSkript:   Message   from   the   document:   "   &
d.SelectSingleNode("*//message").InnerText)

     End Sub
End Module
]]>
</script>

<!-- ************************************* -->

<script name="ThirdScript" language="vb.net">
<![CDATA[
Imports System

Module Main
    Sub Main(args as string())
    Console.WriteLine("ThirdScript: List auf script arguments:")
         ' --- Argumente ausgeben
            Dim s as string
            for each s in args
                  console.writeline("- "& s)
          next
    End Sub
End Module
]]>
</script>
</scriptdoc>
```

Listing 1.1: Example of a DSH script

1.2 Structure of a script

Main ()

A script must provide a clear entry point. Therefore it must consist of at least one class (a module in Visual Basic is special kind of class) with a static method called `Main()`. It is possible to add further attributes and methods in this class or to add further classes with attributes and methods.

Any name can be chosen for the class with the entry point but it may be defined by the XML attribute `startClass` in the `<script>` element. The name of the entry point method always must be `Main()`. Two signatures for `Main()` are possible:

startClass

```
Sub Main()
Sub Main(args as string())
```

In case the signature `Sub Main(args as string()` is chosen, `Main()` receives the command parameters from DSH that have been transferred to the script in an array of strings. Other parameters are not allowed with `Main()`. Any name can be given to this parameter.

Parameters

```
Module Main
    Sub Main(Argumente as string())
    ' …
    End Sub
End Module
```

Listing 1.2: Main structure for an DSH script

1.3 Starting a script

A DSH script is started by accessing DSH.EXE with the file name of the script.

dsh.exe

```
H:\DSH>dsh.exe testscript.dsh
```

This script will produce the following output:

```
---------------------------------
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
---------------------------------

FirstSkript: Hello World!
SecondSkript: Message from the document: Hello World!
ThirdScript: List of script arguments:
- testscript.dsh
```

1.4 Command line parameters

The DSH – like the WSH – supports three kinds of command line parameters:

Command parameters

? The name (and path) of the script that executes the accessing. The first parameter that does not start with a slash is considered to be the file name.

? The parameter fort he DSH itself. They start with a double slash (`//`).

? The parameters fort he accessed script. Parameters that do not begin with a double slash (`//`) are transferred to the accessed script. Thus the script also receives its own file name which will always be transferred to the script as the first parameter.

Parameter	**Comment**
`//ABOUT`	Provides informationen about DSH.
`//H` or `//HELP`	Shows the list of command line options

//EN	Displays all messages in English.
//DE	Displays all messages in German.
//S or //Silent	The DSH issues only outputs it has generated itself – provided no errors occur.
//V or //verbose	The DSH is very verbose and delivers additional outputs concerning the processing of the XML file and the translation.
//ERRORGUI	With this parameter all error messages are displayed in dialogue boxes.
//SUCCESSGUI	All scripts will be displayed in dialogue boxes after successful processing.
//OUT:directory	The compiled assemblies will be saved in the directory given in the parameter.
//SERVER:portnumber	Starts the DSH as a server (listener). On the given port the DSH waits for calls from another DSH instance. The DSH listener remains active until the process is finished.
//COMPUTER:name	Name or IP-Address of the computers on which the script has to be started. With no specification the script will automatically be started on the local computer.
//PORT:portnumber	Port number of the remote DSH instance. With no specification the script will automatically be started on the local computer.
//HTTP	As a protocol for Remote Scripting, HTTP is used instead of TCP. This option may be used in combination with //COMPUTER or with //SERVER. When using //HTTP it is necessary to specify //PORT.

Table 1.1: DSH 1.1 command line options

1.5 Intrinsic objects

Intrinsic Objects, FCI

There are no Intrinsic Objects in the DSH. For the scripts all classes of the .NET Framework Class Library (FCL) are available, like for a "normal" .NET program.

1.6 Assembly references

In order to use a class, the assembly that implements this class has to be specified as a reference. This can be done by indicating the name of the file that contains the assembly in the element <assembly>. Don't add a path to the assembly name.

For the placement of the referenced assembly, we have tow cases:

? Normally, the referenced assembly must be found in the same directory that the dsh.exe. Unfortunately, it is not possible to put the assembly to the GAC or the directory, where the script is.

? The FCL assembly are a special case, because they are in the GAC as well as in the .NET Framework directory. They will be found be the compiler without copying them to the directory where the dsh.exe is.

It is preferable but not necessary to integrate the namespace with Imports, Import or using.

1.7 Assembly persistence

Saving generated assembly

With the option //OUT:directory the generated assembly can be made persistent in the file system. That means that the script is filed in a compiled form. This compilation is a MSIL code that can be started immediately.

When doing this the assembly receives the name of the script within the script document. The file extension is .EXE. After //OUT: the directory where the assembly is to be stored needs to be indicated. In case another file with the same name already exists it will be overwritten without warning.

In case the script document consists of several scripts several assemblies will be produced.

1.8 Remote Scripting

Remoting

The DSH supports Remoting. It is possible to access the DSH on a computer and to instruct it to execute a local script on a remote computer. This is similar to the WSH remoting concept except that with the DSH it is not necessary to have a script on the client itself: The remoting is started with a DSH command line option.

An instance of the DSH must have been started on a certain port on the server. This is another difference to the WSH but corresponds to the remoting concept of the .NET framework.

TCP/IP

The communication between client and server takes place via TCP/IP on a port of your choice

Starting the server

The server is started with the option `//server` by indicating an unused port number. In case the port is still available the incoming calls to the port are controlled.

```
H:\DSH>dsh.exe //server:9999
```

This produced this output:

```
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
Starting DSH listener on port 9999...
Press enter to stop this process.
```

In case the port is not available the following error message appears:

```
ERROR: Normally socket addresses (protocol, net address or connection port) may be
used only once.
```

The word "normally" may seem a little out of place but this is the message prescribed by the.NET Framework Class Library.

Staring the client

The client is started simply by indicating the script name. Additional specifications to be made are the name of the computer `//computer` and the port (`//port`) on which the remote DSH “listens”.

```
dsh.exe testscript.dsh //computer:byfang //port:9999 //v
```

Following a successful access:

```
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
```

In the „talkative“ mode (`//V`) additional information is given:

```
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
Loading Script Document...testscript.dsh
Connecting to Client byfang:9999...
Calling Remote DSH...
Script Document testscript.dsh finished successfully!
```

The client receives an error message in case the access was not successful:

```
DOTNET Scripting Host (DSH)
```

```
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
Loading Script Document...testscript.dsh
Connecting to Client byfang:9999...
Calling Remote DSH...
COMPILE ERROR in Script at Line # 10: BC30451:Name 'irgendwo1' is not declared.
COMPILE ERROR in Script at Line # 15: BC30451:Name 'xyz' is not declared.
```

Output on a server

Please note that all screen outputs of the transferred script take place on the server process in the DOS window!

```
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
Starting DSH listener on port 9999...
Press enter to stop this process.
FirstSkript: Hello World!
SecondSkript: Message from the document: Hello World!
ThirdSkript: List auf script arguments:
- testscript.dsh
- /abc
- testoption
```

A server started in the verbose mode `//v` gives very detailed information:

```
DOTNET Scripting Host (DSH)
Version: 1.1
(C) Holger Schwichtenberg 2002
http://www.windows-scripting.com
Starting DSH listener on port 9999...
Press enter to stop this process.
Receiving call...
Parsing document...
Parsing References...
Adding References: system.xml.dll
Adding References: System.DirectoryServices.dll
Adding References: system.data.dll
Parsing Script Document...
usw.
```

1.9 Limitations

Current limitations of the DSH:

Missing featu

- One script calling up another is not possible.
- It is not possible to let scripts reference other script documents. References can only be made to assemblies.