LogicLib

From NSIS Wiki

Contents

Description

Logic Lib adds some more familiar flow control and logic to NSI Scripts. Things like if, else, while loops, for loops and similar. It is also known as the NSIS Logic Library.

It is appallingly non-documented, but certainly handy if you want to do anything more than very basic logic without having complete Spaghetti Code. LogicLib operators can nest to an indefinite depth, limited by the compiler's macro and label processing capacity.

Additionally, since variable assignment doesn't take place within LogicLib, conditional operators do not conform to "programmatic standards." For example, a single equals sign (=) is used as a comparative operator for integers, rather than the traditional double equals sign (==), which is only valid for string comparisons.

It was created by dselkirk@hotmail.com and eccles@users.sf.net. Shipped with NSIS is version 2.6, which added IfNot support to the otherwise unchanged code from 2004.

A thread exists for discussion of this on the winamp forums at http://forums.winamp.com/showthread.php?s=&postid=1116241

Some Reference

These have been deduced from actually viewing the header file itself, which has a little help info at the top.

The include file for this is LogicLib.nsh. It can be included with:

```
!include 'LogicLib.nsh'
```

Statements

```
If|IfNot|Unless..{ElseIf|ElseIfNot|ElseUnless}..[Else]..EndIf|EndUnless
```

Conditionally executes a block of statements, depending on the value of an expression. IfNot and Unless are equivalent and interchangeable, as are ElseIfNot and ElseUnless.

```
AndIf|AndIfNot|AndUnless|OrIf|OrIfNot|OrUnless
```

Adds any number of extra conditions to If, IfNot, Unless, ElseIf, ElseIfNot and ElseUnless statements.

```
IfThen..|..|
```

Conditionally executes an inline statement, depending on the value of an expression.

```
IfCmd..||..|
```

Conditionally executes an inline statement, depending on a true value of the provided NSIS function.

```
Select..{Case[2|3|4|5]}..[CaseElse|Default]..EndSelect
```

Executes one of several blocks of statements, depending on the value of an expression.

`Switch..{Case|CaseElse|Default}..EndSwitch`

Jumps to one of several labels, depending on the value of an expression. Like in the majority of high-level programming languages, if you do not use ${Break} the execution will 'fall through' the proceeding ${Case} blocks.

`Do[While|Until]..{ExitDo|Continue|Break}..Loop[While|Until]`

Repeats a block of statements until stopped, or depending on the value of an expression. Example:

```
FileRead $fp $line
${DoUntil} ${Errors}
        MessageBox MB_OK $line
        FileRead $fp $line
${LoopUntil} 1 = 0
```

The expression at the start and end of the loop are not required (use ${Do} and ${Loop} without any condition) and need not match. This means you can exit the loop with diferent conditions by specifing both, omit a condition on either the begingin or the end, to obtain a classical "C-like" while(){} or do{}while() loop, or create an infinte loop (${Do} .... ${Loop}).

`While..{ExitWhile|Continue|Break}..EndWhile`

An alias for DoWhile..Loop (for backwards-compatibility)

`For[Each]..{ExitFor|Continue|Break}..Next`

Repeats a block of statements varying the value of a variable. Example:

```
Var i
${ForEach} $i 10 0 - 2
        MessageBox MB_OK $i
${Next}
```

The code above will count backwards from 10 to 0, in steps of 2. Both ends will be reached: 10, 8, 6, 4, 2, 0.

Expressions

The following "expressions" are available:

- Standard (built-in) string tests (which are case-insensitive):

`a == b; a != b`

- Additional case-insensitive string tests (using System.dll):

`a S< b; a S>= b; a S> b; a S<= b`

- Case-sensitive string tests (using System.dll):

`a S== b; a S!= b`

- Standard (built-in) signed integer tests:

`a = b; a <> b; a < b; a >= b; a > b; a <= b`

- Standard (built-in) unsigned integer tests:

`a U< b; a U>= b; a U> b; a U<= b`

- 64-bit integer tests (using System.dll):

`a L= b; a L<> b; a L< b; a L>= b; a L> b; a L<= b`

- Built-in NSIS flag tests:

`${Abort}; ${Errors}; ${RebootFlag}; ${Silent}`

- Built-in NSIS other tests:

```
${FileExists} a
```

Any conditional NSIS instruction test:

```
${Cmd} a
```

- Section flag tests:

```
${SectionIsSelected} ${secA};
${SectionIsSectionGroup} ${secA};
${SectionIsSectionGroupEnd} ${secA};
${SectionIsBold} ${secA};
${SectionIsReadOnly} ${secA};
${SectionIsExpanded} ${secA};
${SectionIsPartiallySelected} ${secA}
```

FileExists

The `${FileExists}` condition included in LogicLib will evaluate to true if what exists is actually a directory. I suggest you add the following macros to your script (right after including `LogicLib.nsh`):

```
;FileExists is already part of LogicLib, but returns true for directories as well as files
!macro _FileExists2 _a _b _t _f
        !insertmacro _LOGICLIB_TEMP
        StrCpy $_LOGICLIB_TEMP "0"
        StrCmp `${_b}` `` +4 0 ;if path is not blank, continue to next check
        IfFileExists `${_b}` `0` +3 ;if path exists, continue to next check (IfFileExists returns true if this is a directory)
        IfFileExists `${_b}\*.*` +2 0 ;if path is not a directory, continue to confirm exists
        StrCpy $_LOGICLIB_TEMP "1" ;file exists
        ;now we have a definitive value - the file exists or it does not
        StrCmp $_LOGICLIB_TEMP "1" `${_t}` `${_f}`
!macroend
!undef FileExists
!define FileExists `"" FileExists2`
!macro _DirExists _a _b _t _f
        !insertmacro _LOGICLIB_TEMP
        StrCpy $_LOGICLIB_TEMP "0"
        StrCmp `${_b}` `` +3 0 ;if path is not blank, continue to next check
        IfFileExists `${_b}\*.*` 0 +2 ;if directory exists, continue to confirm exists
        StrCpy $_LOGICLIB_TEMP "1"
        StrCmp $_LOGICLIB_TEMP "1" `${_t}` `${_f}`
!macroend
!define DirExists `"" DirExists`
```

Then you can use `${If} ${FileExists}` and `${If} ${DirExists}`.