WinFIOL User Guide

USER GUIDE

1/1553-CNA 403 3276 Uen H

Table of Contents

1. WinFIOL User Guide

Short Description

The WinFIOL application is used for terminal emulation to configure and test NEs. It supports VT100 and COMCLI, and runs legacy WinFIOL (7.4 WinFIOL on Windows) scripts.

1.1. WinFIOL Introduction

WinFIOL Overview

WinFIOL is a communication program designed for installation, testing and maintenance of customer exchanges, such as AXE. WinFIOL includes multi-channel communications with up to 100 channels at a time, logging capabilities and a script language. The GUI version includes an editor for script files.

Examples in this User Guide are fetched from Ubuntu. The GUI appearance and path descriptions may be different when running on other versions of Linux.

Differences between WinFIOL on Linux and WinFIOL Windows

WinFIOL on Linux is a porting of the core functionalities that exists in WinFIOL Windows. Not all functionality in WinFIOL Windows has been moved to the new Linux versions.

The following functionalities have been ported:

- Communication on SSH
- The WinFIOL script functionality
- Support for multiple channels (GUI only)
- WinFIOL Logging functionality

The new WinFIOL CLI is available on the scripting cluster, and it enables the usage of WinFIOL functionalities in a terminal.

Target Group

The intended target groups for this document are:

- Operators which need access to MML

1.2. Scripting in WinFIOL

Short Description

This section describes the the script module that is included in WinFIOL. It provides a simple but powerful script language with over one hundred commands. Script commands allow simple-to-use programming control during transmission of script files, which consist of Man-Machine Language (MML) commands and script commands.

Unsupported script commands

> The porting of WinFIOL to Linux deprecated some of the legacy script commands.

Using one of these in a script file generates an error message and the transmission is paused. The following table shows the script commands that are not supported in the Linux version of WinFIOL:

Command name	Command description
`@P`	Printer log
`@PRINTER`	Switch on/off printer logging
`@EXECUTE`	Execute another application
`@EXECWAIT`	Execute another application and wait for it to terminate
`@OPTION`	Change option
`@TEMPLATE`	Load template file and change traffic options
`@TRACE`	Switch on single-step transmission

@UNTRACE	Switch off single-step transmission
@NODE	Switch node
@MACRO	Run a macro
@EXTRACT	Extract data from parse tree
@PARSE	Parse data into parse tree using a PPM file
@XMODEMRCV	Receive file using X-modem
@XMODEMSEND	Send file using X-modem

Also, note that some script commands are only available in WinFIOL GUI. See a list of these in the section Script Commands.

Script basics

All script commands start with @. Any line that does not start with @ is considered a regular MML command. The script commands are case-insensitive. Type the script commands in a script file and transmit the file in the WinFIOL CLI or GUI. The execution of the script commands takes place during script file transmission. A script command is normally only executed after a prompt has been received. If the option "Ignore script commands" is switched on in traffic settings, all script commands are ignored and skipped. Detected errors are reported during transmission.

Variables

The script module allows to set variables using different script commands, for example @SET. A variable is always enclosed in braces:

```
@SET {counter} = 3
@SET {message} = "Something went wrong"
```

Variables can be used in script commands, for example @COMMENT and @IF:

```
@SET {number} = 5
@SET {text} = "The number is smaller!"
@IF {number} < 10 THEN @COMMENT {text}
```

Variables can also be embedded in MML commands, unless that option is disabled in the traffic settings file:

```
@SET {device} = 32
STDEP:DEV=LI-{device};
```

Variable names

Every variable has a unique name and the variable name is case-insensitive. If a value is assigned to an existing variable, the old value is overwritten. The maximum length of a variable name is 40 characters. The variable name must start with a letter (a .. z or A .. Z), and it must **not** contain any of the following characters:

```
~ @ # $ % ^ & * ( ) - + = [ ] ; : ' < > , . / ? | <space>
```

Variable types

There are two types of variables, numeric and text, and the type is determined when assigning the value to the variable. A numeric variable is an integer value between -2147483647 and +2147483648.The maximum length of a text variable is 80 characters.

Learn more about what script commands to use for the respective variable type in the lists of numeric commands and string commands.

Array Variables

All variables are potential arrays. The above rules apply equally to every element in an array variable. An array element is accessed using square brackets []. Array indexes start from 0 (zero), and its first element (index 0) is the same as the non-indexed variable, that is, {x} is equivalent to {x[0]} for most commands. An array element is created when it is assigned. Constant numeric values, variables or expressions can be used as an index.

```
@SET {i} = 6
@COMMENT i = {i[0]}
@SET {counter[7]} = 42
@SET {message[{i}]} = "text string"
```

Automatic variables

Every line received from the NE when executing a script file is stored in an automatic variable. It's possible to read the variable, but not change its value.

The first line received is stored in variable {_LINE0}, the second in {_LINE1} etc. The sequence is reset after every MML command is sent, or when text specified with @ONRECEIVE is received. Usually, {_LINE0} is the MML command itself. The memory pool for storing received lines is 64 kB.

All preceding spaces are removed from the received text stored in the {_LINEn} variables. For example, " NOT ACCEPTED" is stored as "NOT ACCEPTED". Some commands, such as @MERGE, @SIZE, @WRITE, access this automatic variable as an array variable by using the name {_LINES}. The automatic variable {_RCVLINEn} contains the same information as the corresponding {_LINEn} variable, except that preceding spaces are kept intact. For example, " NOT ACCEPTED" is stored as " NOT ACCEPTED".

> Note that different types of network elements may add preceding spaces, which can cause a script to not work as expected with {_RCVLINEn}.

Another automatic variable is {_ONRCVLINE}. This variable is set when text specified with @ONRECEIVE is received, and assigned to the line number of the last line sent command.

The following table shows all automatic variables that are possible to use in scripts:

Variable	Description
{_PROMPT}	Contains the last received prompt including any control characters. An AXE prompt is often preceded by the character ETX (H'03), for example, "\003<".
{_LINES}	Contains the received lines from last sent command. Can be accessed as an array, for example, {_LINES[0]} points normally to the MML command sent. Preceding spaces has been removed.
{_LINEn}	Holds the last received line n, for example, {_LINE5} is equal to line 6 starting from zero (0). {_LINE5} is equal to {_LINES[5]}. {_LINE0} points normally to the MML command sent. {_LINE1} points normally to the first line of the received response. Preceding spaces has been removed.
{_NEXTLINE}	Holds the next received line starting from the current index. If {_LINE5} has previously been used, {_NEXTLINE} is equal to using {_LINE6}. Preceding spaces has been removed.
{_RCVLINES}	Contains the received lines from last sent command. Can be accessed as an array, for example, {_RCVLINES[0]} points normally to the MML command sent. Preceding spaces has been left intact.
{_RCVLINEn}	Holds the last received line n, for example, {_RCVLINE5} is equal to line 6 starting from zero (0). {_RCVLINE5} is equal to {_RCVLINE[5]}. {_RCVLINE0} points normally to the MML command sent. {_RCVLINE1} points normally to the first line of the received response. Preceding spaces has been left intact.
{_NEXTRCVLINE}	Contains the next received line starting from the current index. If {_RCVLINE5} has previously been used, {_NEXTRCVLINE} is equal to using {_RCVLINE6}. Preceding spaces has been left intact.
{_CHANNEL}	Contains the IP address, hostname or the channel name of the active channel.
{_CHNDIR}	Contains the default directory used for channel files. Default is $HOME/winfiol/channelfiles
{_CMDDIR}	Contains the default directory used for script files. Default is $HOME/winfiol/commandfiles
{_LOGDIR}	Contains the default directory used for log files. Default is $HOME/winfiol/logfiles
{_NAME}	Contains the IP address, hostname or the channel name of the active channel. If an alias is used for the channel, in WinFIOL GUI, it contains the alias.

{_NODE}	Contains the current connected node of the active channel. If the channel is connected to another APG40 node than the active node, it is reflected in the information that is retrieved from {_NODE}. The variable can have the values "A", "B" or "ACTIVE". The values "A" and "B" are applicable for APG40 only. Non-APG40 channels always receives "ACTIVE".
{_VERSION}	Contains the version information for WinFIOL. The format is <major version><minor version><service pack>. For example, 100 for WinFIOL 1.0, 101 for WinFIOL 1.1
{_CURRLINE}	Contains the current line when executing a @FOREACH command
{_CURRIDX}	Contains the current line index of the array variable used by the @FOREACH command
{_ONRCVLINE}	Contains the number of the last sent line in the script file, when executing a @ONRECEIVE command. The value of this variable does not change until the next time a specified text is received.

Lifetime of variables

Variables exist during the transmission of a number of lines or a file, even if the transmission is paused. When the transmission is completed or stopped, the variables are lost. However, if @PRESERVE is specified at some point during the transmission, all variables are available for the next transmission. The preservation is cancelled with @CLEAR. Automatic variables cannot be preserved.

Interrogating variables

Preserved variables can be accessed from the buffered input area in the channel window when a transmission is completed. For that, type any script command in the input area, for example @COMMENT {variable}, and the value of the variable is printed in the output area. It is also possible to change the value of preserved variables similarly, using script commands.

When a transmission is just paused, the above applies for all defined variables.

Pattern using regular expressions

A string pattern is used to match strings, or lines in array variables, and is specified using regular expressions. The Perl compatible regular expression syntax is supported in WinFIOL. This is a short summary of some of the possibilities that exist with regular expressions.

The following table shows the meta characters that have special meaning:

Character	Description
.	Matches any single character
^	Represents the beginning of the line
$	Represents the end of the line
\	Turn off the special meaning for a character
\.	Represents a regular dot
	Characters that are used to match repetitions:
*	Match zero or more expressions
+	Match one or more expressions
?	Match zero or one expressions
|	**Match either one sub-expression or another:**
a|b	Match either "a" or "b"
abc|def	Match either "abc" or "def"
[..]	**Use sets to match a list of possible variables:**
[abc]	Match either of "a", "b", or "c"
[^abc]	Match any character except "a", "b", or "c"

The following table shows some examples:

^HELLO.*	Matches a line that begins with the word "HELLO"
.*HELLO.*	Matches a line that contains the word "HELLO".

`.*HELLO$`	Matches a line that ends with the word "HELLO".
`^HELLO$`	Matches a line that contains only the word "HELLO".
`[a-z]`	Matches all characters between "a" and "z".
`foo(bar|foo)`	Matches strings "foobar" or "foofoo".
`foo.*r`	Matches strings like "foobar", "foobr" and "fooxyzr".

See also:
`@COUNT`, `@FIND`, `@GREP`

Escape characters

The text string for some script commands supports escape characters if the string is given in quotation marks. Escape characters are used to insert characters that cannot be added in the regular way, or that may conflict with the reserved characters. The character backslash (\) is used to output otherwise unprintable character.

The following table shows the notation for the respective escape character:

Escape sequence	Description
`\\`	Inserts a single backslash (\) character
`\t`	Inserts a tab character
`\r`	Inserts a carriage return
`\n`	Inserts a line feed
`\"`	Inserts a quotation mark (")
`\{`	Inserts a left bracket ({)
`\}`	Inserts a right bracket (})
`\nnn`	Insert the hexadecimal number nnn
%%	Inserts a single percent sign (%)
%	Shows as % in WinFIOL Linux

Examples:

```
"Ordinary string"
"String\twith\ttab"
"Escaped path: C:\\path\\file.txt"
"This is not a variable \{no_var\}"
"This is \00d\00a two lines"
```

See also:
`@AFTER`, `@APPEND`, `@BEFORE`, `@FORMAT`, `@ITEM`, `@TRIMCHAR`, `@COUNT`, `@CUT`, `@FIND`, `@GREP`, `@PASTE`, `@SENDHEX`, `@SETPROMPT`

Script commands

This section contains syntax, descriptions and usage examples for all script commands that are currently supported in WinFIOL.

> ⚠ Some script commands are only available in the GUI.

WinFIOL CLI generates an error message and the script transmission is not started, if one of these, or one of the deprecated script commands, are used in a script file.

The table below shows the commands that are only available in the GUI:

Command name	Command description
@FORM	Show form window
@ASK	Ask user to assign a value to the variable

@PARAMASK	Variable from parent file, ask if non-existent
@CONFIRM	Ask for confirmation before continuing transmission
@W	Wait for operator response
@CHANNEL	Switch to new channel
@PORTNO	Switch to new channel
@OPENCHAN	Open and switch to new channel
@CLOSECHAN	Close specified channel
@SELECTCHAN	Display browse dialog to select a channel property file
@CLONE	Clone active channel. It is recommended to use @SAVECHAN
@SAVECHAN	Save active channel
@SELECTFILE	Display browse dialog to select files
@SCRIPTLOG	Create, append, or close a script log file
@PAUSE	Pause the transmission

Variable names are case-insensitive

The notation used in script command is shown in the table below:

Notation	Description
<text>	text is mandatory
[text]	text is optional
x|y or-sign	define either x or y

In the following table you can find all available script commands with a short description. Click on the command name to see the full description and examples:

Command name	Command description
!@ or @! or @@	Comment (Portability)
@A <+|->	Auto confirm
@ABS {variable}	Returns the absolute numerical value
@AFTER {dest} <source> <substr>	Return string after substring
@APPEND <filename> <text>	Append text to a file
@ASK {variable} [question]	Ask user to assign a value to the variable
@BEEP	Beep
@BEFORE {dest} <source> <substr>	Return string before substring
@BREAK	Break
@C <+label>|<-label>	Conditional jump to label
@CALC {result} = <expression>	Calculate a numerical expression
@CHANNEL <alias>	Switch channel (alias of the opened channel)
@CLEAR	Clear all previously preserved variables
@CLONE {result}	Clone active channel. It is recommended to use @SAVECHAN
@CLOSE	Close log file

`@CLOSECHAN <alias>`	Close channel (alias of the opened channel)
`@COMMENT <message>`	Show message in the output window
`@COMPACT {variable}`	Remove empty lines
`@CONCAT {dest} <sub-string1> [<sub-string2> …]`	Concatenate sub-strings
`@CONFIRM [text]`	Confirm
`@CONNECT`	Connect
`@COPY {source} {dest} <m> <n>`	Copy a sub-string
`@COUNT {dest} {variable} <pattern>`	Count matching lines
`@CUT {dest} {source} COL|POS <n> [SEP]`	Cut, or extract, a column or range of characters
`@DEC {variable} [n]`	Decrement variable by n
`@DECIMAL {dest} {variable}|<number>`	Convert number to decimal
`@END`	Stop the transmission
`@ENDLOOP`	End of loop
`@ENDWHILE`	End of while loop
`@ERASE <filename>`	Erase/Delete file
`@EXIT`	Exit include file
`@FIND {dest} <var> <pattern>`	Find matching lines, return line index
`@FOREACH {variable} <script cmd>`	For each line, apply script command
`@FORM <sub-command> [parameters]`	Form window
`@FORMAT {dest} <format> …`	Format a string
`@GETDATE {variable} [format]`	Store current date in variable
`@G <+|->`	Send comments
`@GETDAY {variable} [days]`	Store current day of week in variable
`@GETFILE <url> <filename>`	Get file from URL to local machine
`@GETTIME {variable} [format]`	Store current time in variable
`@GOSUB <label>| {variable}`	Go to subroutine
`@GOTO <label>| {variable}`	Go to label
`@GREP {dest} <source> <pattern>`	Find matching lines, return matching lines
`@HALT`	Stop the transmission
`@HEX {dest} {variable}|<number>`	Convert number to hexadecimal
`@I <filename> [n]`	Include file to transmit
`@IF {str} IN {arr} THEN <statement>`	Conditional statement
`@IF {str} MATCHES <pattern> THEN <statement>`	Conditional statement
`@IF <condition> THEN <statement>`	Conditional statement
`@IFDEF {variable} THEN <statement>`	Conditional statement if variable exists
`@IFERROR THEN <statement>`	Conditional statement on error
`@IFNDEF {variable} THEN <statement>`	Conditional statement if variable does not exist
`@INC {variable} [n]`	Increment variable by n
`@INCLUDE <filename>|<url>`	Include file to transmit
`@INSERT {string} <position> <sub-string>`	Insert a substring
`@ITEM {dest} <source> <delim> <n>`	Return the n:th substring separated by delimiters

@K <HH:MM:SS>	Wait until HH:MM:SS
@L <[+]filename|+|->	Log file
@LABEL <label>	Label destination
@LENGTH <string> {length}	Get string length
@LOG ON <file>|APPEND <file>|OFF	Create log file/close log file
@LOOP <n>	Loop <n> times
@LOWCASE {variable}	Convert to lower case
@M <n>	Wait n minutes
@MERGE {variable} = {source} [[+|-]{var_n}]	Copy, append and/or remove data from an array variable
@ONERROR [statement]	Error handling
@ONRECEIVE [text [statement]]	Check all received text
@OPENCHAN <channel name> [node]	Open and switch to channel
@PARAM {variable}	Variable from parent file
@PARAMASK {variable} <question>	Variable from parent file, ask if non-existent
@PASTE {dest} {source} [SEP <s>]	Paste, or append, a column to a variable
@PAUSE [reason]	Pause the transmission
@PORTNO <alias>	Switch channel (alias of the opened channel)
@PRESERVE	Preserve all variables
@PUTFILE <filename> <URL>	Transfer local file to remote URL
@QUIET	Do not show confirmation of script command
@R <+|->	Pause on error
@READ <filename>|{variable} {dest}	Read data from a file, one line per array element, into variable {dest}
@RELEASE	Release
@REPLACE {string} <from-string> <to-string>	Replace a substring
@RESTOREPROMPT	Restore default prompts
@RETURN	Return from subroutine
@RITEM {dest} <source> <delimiters> <n>	Return the n:th substring from the right separated by delimiters
@ROUND {variable}	Returns the integer part of a numerical value
@SAVECHAN {result} {variable}|"filename"	Save active channel. A file name may be provided, either as variable or in a text string
@SCAN {variable} <text> {pos}	Scan local position of text in a variable
@SCRIPTLOG <ON filename>|<APPEND filename>|<OFF>	Create/Append/Close script log file
@SELECTCHAN {dest} [<title>]	Display browse dialog to select a channel property file
@SELECTFILE {dest} <title> [<initialdir>] [<filter>]	Display browse dialog to select files
@SENDFILE <file>	Send file
@SENDHEX <hh>	Send hexadecimal value immediately
@SET {variable} = <expression>	Assign a value to a variable
@SETERROR [error]	Set error condition
@SETPROMPT <prompt>	Set temporary prompt for the channel

`@SIZE {source} {elementcount}`	Calculate the number of array elements in {source} and set variable {elementcount} to that value
`@SORT {variable}`	Sort lines
`@STOP`	Stop the transmission
`@T <n>`	Wait n seconds
`@TRIM {variable}`	Trim line from white space
`@TRIMCHAR {variable} <char>`	Delete characters from string
`@UNIQUE {variable}`	Remove duplicate lines
`@UNSET {variable}`	Undefine (remove) variable
`@UPCASE {variable}`	Convert to upper case
`@VERBOSE`	Show confirmation of script command
`@WAITFOR /<text>[,text][...]/`	Wait for specific text
`@WAITPROMPT <ON|OFF>`	Disables or enables waiting for prompt
`@WHILE <condition>`	Loop until <condition> is false
`@WRITE {source} <filename>|{variable}`	Write data, one element per line, from variable {source} to a file
`@/<text>[,text][...]/`	Check for text in received data

@A <+|->

A line starting with @A+ switches on and @A- switches off the option auto confirm. When auto confirm is switched on, WinFIOL automatically sends a semicolon (;) if a command needs to be confirmed. Note that auto confirm is not available when a line is sent manually, that is, from the input window.

This value can also be set in the traffic settings file.

@ABS {variable}

This script command returns the absolute value of the given numerical value.

Examples:

```
@SET {num} = -17
@COMMENT before: {num}
## before: -17
@ABS {num}
@COMMENT after: {num}
## after: 17
```

```
@CALC {num} = -14.12
@COMMENT before: {num}
## before: -14.12
@ABS {num}
@COMMENT after: {num}
## after: 14.12
```

See also:
`@CALC`, `@ROUND`

@AFTER {dest} <source> <sub-string>

This script command scans the source string for the given sub-string. If the sub-string is found, it returns the rest of the string, which comes after the substring, in variable `{dest}`. If the sub-string is not found, `{dest}` is set an empty string. The sub-string supports escape characters. The source and sub-strings may contain variables.

Example:

```
@SET {source} = "AD-47"
@AFTER {dest} {source} "-"
@COMMENT dest holds: {dest}
## dest holds: 47
@AFTER {dest} "This is a string" "is a"
@COMMENT dest holds: '{dest}'
## dest holds: ' string'
@SET {source} = "This is a string"
@AFTER {dest} {source} "Hello"
@COMMENT dest is an empty string '{dest}'
## dest is an empty string ''
```

See also:
`@BEFORE`, `@FORMAT`, `@ITEM`

@APPEND <filename> <text>|{variable} [+ <text>|{variable}] …

This script command appends one or multiple string values or variables to the specified file. Each element in an array variable is added to the file on a separate line. Strings or variables are added by using a plus-sign (+). If a single valued value, string or variable, is stated with a multiple valued array variable, it is converted to an array variable with the same amount of elements before appended to the file. The last value is used to fill the created array variable.

The `<filename>` and `<text>` can be given as a quoted string or a variable. `<text>` may also be given as a combination of the two.

Examples:

```
@SET {fname} = "/tmp/append.txt"
@SET {single} = "prefix"
@MERGE {arr} = "Line1" + "Line2" + "Line3"
@ERASE {fname}
@APPEND {fname} {single} + "-" + {arr}
@READ {fname} {result}
@FOREACH {result} comment {_currline}
@@ The result is:
## prefix-Line1
## prefix-Line2
## prefix-Line3
```

```
! Append a comment to a file !
@APPEND "/tmp/TEST.LOG" "Starting test case"
```

```
! Using variables !
@SET {FILE} = "/tmp/USER.LOG"
@APPEND {FILE} "Running test"
```

```
! Using variables !
@SET {TEXT} = "This is a comment"
@SET {FILE} = {_LOGDIR} + "/USER.LOG"
@APPEND {FILE} {TEXT}
```

```
 ! Writing an array !
@READ "TESTCASE1.CMD" {TEXT}
@APPEND "COMPLETE.CMD" {TEXT}
@READ "TESTCASE2.CMD" {TEXT}
@APPEND "COMPLETE.CMD" {TEXT}
```

```
! Appending single and multiple valued variables !
@SET {fname} = "/tmp/append.txt"
@SET {single} = "prefix"
@MERGE {arr} = "Line1" + "Line2" + "Line3"@erase {fname}
@APPEND {fname} {single} + "-" + {arr}
@READ {fname} {result}
@FOREACH {result} comment {_currline}
@@ The result is:
## prefix-Line1
## prefix-Line2
## prefix-Line3
```

See also:
@READ, @WRITE, @MERGE

@ASK {variable} [question]

> @ASK is only supported in the GUI version of WinFIOL

This script command shows a dialog box, in which the user can type a value (text or number) for the specified variable. The variable is created if it doesn't exist yet. When the dialog box is displayed, the transmission is temporarily paused. The question is optional and may contain variables.

It is recommended to use the @FORM command to initialize multiple variables.

Example:

```
@COMMENT Connect private meter
@ASK {lidev} "Give number of LI device"
@ASK {seprm} "Give number of SEPRM device"
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
UPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
```

See also:
@SET, @PRESERVE, @PARAMASK

@BEEP

This script command makes a short "beep" sound. It is also possible to use @B.

Example:

```
Connect private meter
@BEEP
@ASK {lidev} "Give number of LI device"
@ASK {seprm} "Give number of SEPRM device"
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
SUPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
```

@BEFORE {dest} <source> <sub-string>

This script command scans the source string for the given sub-string. If the sub-string is found, it returns the string prior to the sub-string in variable {dest}. If the sub-string is not found, {dest} is set to <source>. The sub-string supports escape characters. The source and sub-strings may contain variables.

Example:

```
@BEFORE {dest} "This is a string" "is"
@COMMENT dest holds: '{dest}'
## dest holds: 'Th'
@SET {source} = "This is a string"
@BEFORE {dest} {source} "Hello"
@COMMENT dest contains: {dest}
## dest contains: This is a string
@SET {source} = "AD-47"
@BEFORE {dest} {source} "-"
@COMMENT dest holds: {dest}
## dest holds: AD
```

See also:
@AFTER

@BREAK

This script command immediately sends a break sequence to the communications port.
Example:

```
@WAITPROMPT OFF
PCORP:BLOCK=ALL;
@BREAK
```

See also:
@CONNECT, @RELEASE

@C+|- <label>

This script command makes a jump to the subroutine specified by <label>, if the condition is met. The table below describes the different options for @C:

Option	Condition	Example	Description
+	On error	@C+ errorLabel	Jumps to subroutine "errorLabel" if WinFIOL detects an error, else WinFIOL continues on the next line in the script file.
-	No error	@C- labelNoError	If no error is detected, WinFIOL jumps to the label "labelNoError", else (error) it continues on the next line in the script file.

If an error message is received from the NE, for example after a faulty command, WinFIOL sets an error condition. After a new every line sent, the error condition is reset.

> ⚠ For compatibility reasons, @C without any parameters closes the current log file. However, it is recommended to use @CLOSE.

See also:
@LABEL, @IFERROR

@CALC {result} = <expression>

This script command calculates a numerical expression using the correct operator precedence, and returns a result with up to two decimals.

If the result consists of a integer part only, it is stored without any decimal part. The result can then be used by script commands that accept numeric values, for example @SET.

If the result consists of a decimal part, it is stored with an extra two digits representing the decimal part. For example: 12.34 or -785.34. This format with decimal part can only be used by the numeric commands. Other script commands does not see it as a valid numeric value, that is, an integer. Use @ROUND or @ABS to convert this format to an integer only format, which can be used as a regular numeric value.

The following table shows what functions that is supported by the @CALC command:

Function	Description

`Pow(x,y)`	Returns the power of x
`Exp(x)`	Returns the exponential of x
`Div(x,y)`	Divides the two numbers and returns the result
`Mod(x,y)`	Returns the modulo of the two numbers
`Min(x,y)`	Returns the smallest of the two numbers
`Max(x,y)`	Returns the largest of the two numbers

The command `@SET` does not follow the rule of operator precedence.

Examples:

```
! Assign a number !
@CALC {RES} = 1.23
```

```
! Some expressions !
@SET {var1} = 100
@SET {var2} = 23
@CALC {res} = (1+2+3+4)*5/3+500
@comment {res}
## 516.67
```

```
@CALC {res} = {var1} / {var2}
@COMMENT {res}
## 4.35
@CALC {res} = pow(2,8)
@COMMENT {res}
## 256
```

```
@CALC {res} = 100 + exp(3)
@COMMENT {res}
## 120.09
@CALC {res} = min(100,200)
@COMMENT {res}
## 100
@CALC {res} = max(100,200)
@COMMENT {res}
## 200
```

See also:
`@ABS`, `@ROUND`

@CHANNEL <alias>, @PORTNO < alias >

> `@CHANNEL` and `@PORTNO` are only supported in the GUI version of WinFIOL

This script command enables the transmission to continue in another channel. The <alias> is set in the channel by right-click and type a name. It supports 80 characters and the following conditions must be met:

1. <alias> may not be the current channel
2. <alias> must be open (both window and communications port)
3. channel <alias> must not be transmitting

Example:
Assume that this is channel 1, transmitting to AT-1. Channel 2 is connected to AT-2. Both CP-sides are initially parallel.

```
FLSLI:SPG=0;
IMLCT:SPG=0;
MCDVC:IO=AT-2,SEP=YES;
END;
FCSEI;
@CHANNEL 2
@RELEASE
@/SYSTEM RESTARTED/
@CONNECT
SYATI;
@CHANNEL 1
Same example using PORTNO
FLSLI:SPG=0;
IMLCT:SPG=0;
MCDVC:IO=AT-2,SEP=YES;
END;
FCSEI;
@PORTNO 2
@RELEASE
@/SYSTEM RESTARTED/
@CONNECT
SYATI;
@PORTNO 1
```

See also:
@OPENCHAN, @CLOSECHAN

@CLEAR

This script command clears all preserved variables. If the command is executed from an include file, that is, a script file run from another script file, using the @INCLUDE command, any variables from the parent file are not cleared. For a more detailed description, see @PRESERVE.
Example:

```
@PRESERVE
@SET {test} = "test"
@CLEAR
```

See also:
@PRESERVE

@CLONE {result}

> @CLONE and @SAVECHAN is currently only supported in the GUI version of WinFIOL

This script command clones the active channel and returns the file name of the new channel. The channel is opened using @OPENCHAN.

This script command is deprecated, but will still function to support legacy script files. It is recommended to use @SAVECHAN instead.

```
@@ Clone active channel
@CLONE {newchn}
@@ Open the new channel
@OPENCHAN {newchn}
```

See also:
@OPENCHAN, @CLOSECHAN, @CHANNEL, @SAVECHAN

@CLOSE

This script command closes a log file. This script command has the same function as the script command @LOG OFF. It is possible to use @C and @L- for short.

Example:

```
@LOG ON pcorp.log
PCORP:BLOCK=ALL;
@CLOSE
```

See also:
@LOG

@CLOSECHAN <alias>

> @CLOSECHAN is only supported in the GUI version of WinFIOL

This script command closes an open channel. The <alias> set in the channel by right-click and type a name. It supports 80 characters and the following conditions must be met:

1. <alias> may not be the last open channel
2. channel <alias> must not be transmitting

Example:

Work on the current channel is finished. A new channel is opened and the previous channel is closed.

```
@@ Save channel number for closing
@SET {prev_chan} = {_CHANNEL}
@OPENCHAN target_name
@@ Close old channel
@CLOSECHAN {prev_chan}
```

See also:
@OPENCHAN, @CHANNEL

@COMMENT <text>

This script command displays text in the output window. The text is automatically preceded by two hash marks (##). If the text specified after @COMMENT is enclosed in double quotes, it is shown in the output window without the quotes. If the text is not enclosed in double quotes, the text may contain variables.

Example:

```
@GETDATE {date}
@COMMENT The current date is: {date}
@COPY {date} {year} 1 2
@COMMENT "The current year is stored in variable {year}: " {year}

## The current date is: 160906
## The current year is stored in variable {year}: 16
```

@COMPACT {variable}

This script command removes empty lines from the specified variable.

Examples:

```
@SET {var[0]} = "This is a line"
@SET {var[1]} = "This is also a line"
@SET {var[2]} = ""
@SET {var[3]} = "A fourth line"
@SET {var[4]} = ""
@COMPACT {var}
! Print the result !
@FOREACH {var} COMMENT {_CURRLINE}
This is a line
This is also a line
A fourth line
```

```
LASIP:BLOCK=AOT;
@FOREACH {_lines} comment {_curridx}: {_currline}
@COMMENT -----------------------
! _LINES cannot be changed so we need to copy it to another
variable !
@MERGE {caclp} = {_lines}
@COMPACT {caclp}
@FOREACH {caclp} comment {_curridx}: {_currline}
! Prints: !
## 0: <LASIP:BLOCK=AOT;
## 1: SOFTWARE UNIT IDENTITY
## 2:
## 3: BLOCK BN BS FCODE ID
## 4: AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG
R2A02
## 5:
## 6: END
## 7:
## -----------------------
## 0: <LASIP:BLOCK=AOT;
## 1: SOFTWARE UNIT IDENTITY
## 2: BLOCK BN BS FCODE ID
## 3: AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG
R2A02
## 4: END
```

See also:

@COUNT, @FIND, @GREP

@CONCAT {dest} <sub-string1> [<sub-string2> ...]

This script command concatenates the specified sub-strings into variable {dest}. The sub-strings may be variables or quoted strings.
Example:

```
@CONCAT {dest} "This" " is " "a string"
@COMMENT dest holds: {dest}
## dest holds: This is a string
@SET {source} = "you there"
@CONCAT {dest} "Hello " {source}
@COMMENT dest holds: {dest}
## dest holds: Hello you there
```

See also:
@AFTER, @BEFORE, @FORMAT, @ITEM

@CONFIRM [text]

> @CONFIRM and @W are only supported in the GUI version of WinFIOL

This script command suspends the transmission and shows a message box. The transmission continues if you press the "Continue" button in that dialog box. It is also possible to use @W for short.

Example:

```
REMEI:EMG=RSS0,MAG=EM-0,PCB=EMRP-A-POU;
@CONFIRM "Switch the power of the EMRP on"
RECEI:EMG=RSS0,EMRP=0-A;
```

@CONNECT

This script command immediately attempts to connect the terminal. It is also possible to use `@S or @O` for short.

Example (AXE, auto release and reconnect are off):

```
LABUP;
@RELEASE
@WAITFOR /END/
@CONNECT
```

See also:
`@RELEASE`

@COPY {source} {dest} <m> <n>

This script command copies a sub-string from `{source}` into `{dest}`, starting at position <m>, including <n> characters. The source variable must exist. The destination variable is created if it does not already exist. Both m and n may be numeric variables. If the source variable is empty, the destination variable will also be empty.

Example:

```
@SET {var1} = "Search target string"
@COPY {var1} {var2} 8 6
@IF {var2} = "target" THEN GOTO ready
@COMMENT Could not find target (var2 = {var2})
@LABEL ready
@COMMENT var2 contains {var2}
```

See also:
`@AFTER`, `@BEFORE`, `@ITEM`, `@SET`, `@SCAN`, `@LENGTH`

@COUNT {dest} {variable} <pattern>

This script command counts the lines in `{variable}` that match the specified <pattern>. The matching count is returned in variable `{dest}`. The pattern string supports escape characters.

Example:

```
@SET {var[0]} = "This is a line"
@SET {var[1]} = "This is also a line"
@SET {var[2]} = "This is another line"
@SET {var[3]} = "A fourth line"
@SET {var[4]} = "A fifth line"
@COUNT {cnt} {var} ".*is.*"
@COMMENT Number of lines matching "is": {cnt}
## Number of lines matching "is": 3
```

See also:
`@FIND`, `@GREP`

@CUT {dest} {source} COL <column>[,..] [SEP <separator>]

@CUT {dest} {source} POS <character positions>[,..]

This script command extracts, or cuts, a range of columns or characters from the variable `{source}`. The extracted values are returned in variable `{dest}`. If multiple column or character positions are specified, a space character separates each column or character range in variable `{dest}`.

The default column separator is the space character. Use SEP to specify a different single character as a column separator. The following table describes the syntax:

Option	Description

COL	Use a comma to separate multiple columns and a dash specify a range of columns: COL 1,2,5-7
POS	Use a comma to separate multiple character positions and a dash to specify a range of positions: POS 1-5,10,12,20-30
SEP	A separator is given as a variable or a text string enclosed in quotation marks. The separator string can consist of one or a list of single characters: "/-," Escape characters are supported.

The below example cuts the columns 2, 3, 5, 6 and 7 from the variable {source}. The column separators to find in {source} are: a space character, a hash mark and an equal sign.

```
@CUT {dest} {source} COL 2,3,5-7 SEP " #="
```

Examples:

```
IOIOP:IO1=ALL;
@GREP {ioiop} {_lines} ".*-.*"
@CUT {dev} {ioiop} COL 1
@FOREACH {dev} COMMENT {_currline}
## AMTP-0
## TW-0
## AT-0
## AT-5
## AT-1
## AT-4
## NVT-120
## AF-0
## AF-1
## AF-12
```

```
IOIOP:IO1=ALL;
@GREP {ioiop} {_lines} ".*-.*"
@CUT {dev} {ioiop} COL 1 SEP " "
@FOREACH {dev} COMMENT {_currline}
## AMTP-0
## TW-0
## AT-0
## AT-5
## AT-1
## AT-4
## NVT-120
## AF-0
## AF-1
## AF-12
```

```
IOIOP:IO1=ALL;
@GREP {ioiop} {_lines} ".*-.*"
@CUT {state} {ioiop} POS 1-6,60,61,62,63,64
@FOREACH {state} COMMENT {_currline}
## AMTP-0
## AD-1   B L O C K
## AT-0   B L O C K
## AT-1   I D L E
## AT-5   I D L E
## AT-4   B L O C K
## AD-2   B L O C K
## AT-2   I D L E
## AT-3   I D L E
## AMTP-1
## AMTP-2
## AMTP-3
```

See also:
@PASTE, @FIND, @GREP

@DEC {variable} [n]

This script command decrements the numeric variable `{variable}` by *n*. The variable must exist and be numeric. If n is not specified, the variable is decreased by 1.

Example:

```
@SET {dev} = 16
@SET {loop} = 3
@LABEL again
EXDEP:DEV=BT3-{dev};
@INC {dev} 32
@DEC {loop}
@IF {loop} > 0 THEN GOTO again
```

See also:
`@SET`, `@INC`, `@LOOP`

@DECIMAL {dest} {variable}|<number>

This script command converts a given hexadecimal number to decimal. The given hexadecimal number may be a constant or a numeric variable. The hexadecimal value may be on the form 0x92A, H'92A or 92A.

Examples:

```
@SET {n} = "0x92A"
@DECIMAL {dec_n} {n}
@COMMENT dec_n = {dec_n}
## dec_n = 2346
```

```
@SET {n} = "H'92A"
@DECIMAL {dec_n} {n}
@COMMENT dec_n = {dec_n}
## dec_n = 2346
```

See also:
`@HEX`

@END, @HALT, @STOP

Each of these script commands stops the transmission immediately. Three different script commands performing the same action, is for compatibility reasons with other command handlers. If the current script file is an included script file, that is, run from another script file, using the @INCLUDE command, it is recommended to use the `@EXIT` command instead.

Example:

```
DPWSP;
@COPY {_line4} {wo} 24 2
@IF {wo} = "WO" THEN STOP
DPPAI;
@END
```

See also:
`@PAUSE`, `@EXIT`

@ENDLOOP

This script command marks the end of a loop created with the `@LOOP` command.
Example:

```
@LOOP 5
@T 10
PLLDP;
@ENDLOOP
```

See also:
@LOOP

@ENDWHILE

This script command marks the end of a loop created with the @WHILE command.
Example:

```
@SET {a} = 5
@SET {b} = 0
@WHILE {a} > {b}
@T 10
PLLDP;
@INC {b}
@ENDWHILE
```

See also:
@WHILE

@ERASE <filename>

This script command deletes the given file. The command does not ask for confirmation.
Example:

```
@ERASE "/tmp/MYLOG.TXT"
@SET {fname} = "/tmp/LOGFILE.TXT"
@ERASE {fname}
```

See also:
@APPEND, @READ, @WRITE

@EXIT

This script command ends the transmission in the current script file, and continues at the higher level script file. That is, the file that included the current script file, using the @INCLUDE command. If the current script file is not included, the transmission stops, as with the @END, @HALT and @STOP commands.

Example:

```
BLEEE:EMG=RSS0,EM=0;
@IFERROR THEN GOTO repair_emrp
@EXIT
@LABEL repair_emrp
```

See also:
@END, @INCLUDE

@FIND {dest} {variable} <pattern>

This script command finds lines matching the <pattern> in {variable} and returns the line index of the matching line in {dest}. The pattern string supports escape characters.

Examples:

```
@SET {var[0]} = "This is a line"
@SET {var[1]} = "This is also a line"
@SET {var[2]} = "This is the third line"
@SET {var[3]} = "A fourth line"
@SET {var[4]} = "Last line"
@FIND {index} {var} "This.*"
! Print the result !
@FOREACH {index} COMMENT Found "This" at index {_CURRLINE}
## Found "This" at index 0
## Found "This" at index 1
## Found "This" at index 2
```

```
LASIP:BLOCK=AOT&ADE&AD2;
@FIND {blockidx} {_lines} ".*/.*/.*"
@FOREACH {blockidx} COMMENT Block at index {_currline}
! This is shown in the output window: !
<LASIP:BLOCK=AOT&ADE&AD2;
SOFTWARE UNIT IDENTITY
BLOCK BN BS FCODE ID
AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG R2A02
ADE H'140 ACTIVE 5400/CAAZ 107 0842/M06AE R2A01
AD2 H'4D8 ACTIVE 5600/CAAZ 107 6539/M08E R2A01
END
## Block at index 4
## Block at index 5
## Block at index 6
```

See also:
`@GREP`, `@COUNT`

@FOREACH {variable} <script command>

This script command executes a script command for each of the lines stored in `{variable}`. Only script commands that does not jump or exit the control path may be used. For example, `@IF` and @GOTO are **not** allowed. The automatic variables `{_CURRLINE}` and `{_CURRIDX}` are available during the execution of `@FOREACH`. The script command may contain variables.

Examples:

```
! Print the elements in the array !
@FOREACH {variable} COMMENT {_CURRLINE}
```

```
! Execute a subrouting for each element !
@SET {var[0]} = "IOSTP;"
@SET {var[1]} = "SYCLI;"
@SET {var[2]} = "LASIP:BLOCK=ALL;"
@SET {var[3]} = "PCORP:BLOCK=ALL;"
@FOREACH {var} GOSUB EXEC_CMD
@EXIT
@LABEL EXEC_CMD
{_CURRLINE}
@RETURN
```

```
! For all network elements, execute some commands !
@READ "/tmp/cmds_to_execute.txt" {cmdList}
@READ "/tmp/list_of_ne.txt" {nelist}
@SIZE {nelist} {necount}
@SET {i} = 0
@LOOP {necount}
@OPENCHAN {nelist[{i}]} ACTIVE
@FOREACH {cmdList} GOSUB EXEC_CMD
@CLOSECHAN {_channel}
@INC {i}
@ENDLOOP
@EXIT
@LABEL EXEC_CMD
{_CURRLINE}
@RETURN
```

```
! Remove "Line " from each element in the array !
@SET {var[0]} = "Line 1"
@SET {var[1]} = "Line 2"
@SET {var[2]} = "Line 3"
@SET {var[3]} = "Line 4"
@FOREACH {var} ITEM {var[{_CURRIDX}]} {_CURRLINE} " " 1
@FOREACH {var} COMMENT {_CURRLINE}
! Prints: !
## 1
```

```
## 2
## 3
## 4
```

See also:
`@COUNT`, `@COMPACT`, `@GREP`, `@SORT`, `@UNIQUE`

@FORM <sub-command> [parameters]

> `@FORM` is only supported in the GUI version of WinFIOL

This script command is used to create and run a window for entering data. It must be used multiple times with various sub-commands, where the first sub-command must be "`CREATE`" and the last "`RUN`". In between these, other sub-commands are used to add text and input fields to the window. An OK and a Cancel button are automatically added.

The following table shows the description of the sub-commands and valid parameters:

Sub command syntax	Description
`@FORM CREATE [title]`	With the `CREATE` sub-command a new window is created. Optionally, a title can be specified. If the text is not enclosed in double quotes, the text may contain variables. The window is not visible until the `RUN` sub-command.
`@FORM TEXT <text> <x> <y> <w> <h>`	With the `TEXT` sub-command any text is added to the window. This text must be a variable or fixed text embedded in double quotes. The text or variable should be followed by four numbers or variables, representing the x-position, y-position, width and height respectively.
`@FORM EDITTEXT {variable} <x> <y> <w>`	With the `EDITTEXT` sub-command a edit field for entering text is added to the window. The first parameter is a variable, which is given the value that the user types in the edit field. The variable should be followed by three numbers or variables, representing the x-position, y-position and width respectively. The height of the edit field is chosen automatically.
`@FORM EDITNUMBER {variable} <x> <y> <w> [min] [max] [step]`	With the `EDITNUMBER` sub-command a number edit field is added to the window. The first parameter is a variable, which is given the value that the user types in the edit field. The variable should be followed by three numbers or variables, representing the x-position, y-position and width respectively. The height of the edit field is chosen automatically. The next three numbers are optional and define the minimum allowed numerical value (default -2147483648), maximum allowed numerical value (default 2147483647) and increment (default 1).
`@FORM GETCHANNEL {variable} <x> <y> <w>`	With the `GETCHANNEL` sub-command a combo box is added to the window, from which a channel other than the current one may be selected. The first parameter is a variable, which is given the value of the alias of the channel that the user selects. The variable should be followed by three numbers or variables, representing the x-position, y-position and width respectively. The height of the combo box is chosen automatically.
`@FORM RUN`	With the `RUN` sub-command the transmission is paused and the window is shown. The user enters the values and confirms using the OK button, after which the transmission continues. If the user presses Cancel, the transmission is paused.

The values for the x-position, y-position and width scale to approximately 1.5 millimetres per increment, depending on the screen resolution and monitor size.

Example:

```
@SET {value} = "a value"
@SET {data} = 6
@SET {channel} = 1
@SET {title} = "From script"
@FORM CREATE {title}
@FORM TEXT "Enter text:" 2 2 14 15
@FORM EDITTEXT {value} 20 2 25
@FORM TEXT "Enter number:" 2 8 14 15
@FORM EDITNUMBER {data} 20 8 25 0 15 1
@FORM TEXT "Enter channel:" 2 14 14 15
@FORM GETCHANNEL {channel} 20 14 25
@FORM RUN
```

See also:
`@ASK`, `@SELECTCHAN`, `@SELECTFILE`

@FORMAT {string} <format> [<value1> [<value2>] ..]

This script command formats a `{string}` variable. Each `<value>`, if any, is converted and output according to the corresponding format specification in `<format>`. A format specification, which consists of optional and required fields, has the following form:

```
%[flags] [width] [.precision] type
```

Each field of the format specification is a single character or a number signifying a particular format option. The simplest format specification contains only the percent sign and a *type* character, for example, `%s`.

The following table describes the different format options:

Option	Description
type	Required character that determines the type of the associated value. Supported characters are:
s	String
d	Number
flags	Optional character or characters that control justification of output and printing of signs and blanks prefixes. Supported flags are:
-	Left align the result within the given field width
0	If width is prefixed with 0, zeros are added until the minimum width is reached
width	Optional number that specifies the minimum number of characters output.
precision	Optional number that specifies the maximum number of characters printed for the output field, or the minimum number of digits printed for numeric values.

Unprintable characters are specified using escape characters. The script command may contain variables.

Examples:

```
! Format a single string !
@FORMAT {s} "This string contains no format specification"
@COMMENT {s}
## This string contains no format specifications
```

```
! Format strings !
@SET {me} = "me"
@FORMAT {s} "%s from \"%s\"" "Hello" {me}
@COMMENT {s}
## Hello from "me"
```

```
! Format numbers !
@SET {n1} = 123
@SET {n2} = 456
@FORMAT {s} "%d %x %X" {n1} {n2} 987
@COMMENT {s}
## 123 1c8 3DB
```

```
! Send three lines to the target system !
@FORMAT {s} "This line\00d\00ais split\00d\00ainto three lines\00d\00a"
@SENDHEX {s}
```

```
! Left and right justifications !
@SET {n1} = 123
@FORMAT {s} "%8d Right justification" {n1}
@COMMENT {s}
## 123 Right justification
@FORMAT {s} "%-8d Left justification" {n1}
```

```
@COMMENT {s}
## 123 Left justification
@FORMAT {s} "%08d Right justification with padding" {n1}
@COMMENT {s}
## 00000123 Right justification with padding
@FORMAT {s} "%-08d Left justification with padding" {n1}
@COMMENT {s}
## 123 Left justification with padding
@FORMAT {s} "%8.4d Right with precision" {n1}
@COMMENT {s}
## 0123 Right with 4 digit precision
@FORMAT {s} "%-8.4d Left with precision" {n1}
@COMMENT {s}
## 0123 Left with precision
@FORMAT {s} "%08.4d Right, padding and precision" {n1}
@COMMENT {s}
## 0123 Right, padding and precision
@FORMAT {s} "%-08.4d Left, padding and precision" {n1}
@COMMENT {s}
## 0123 Left, padding and precision
```

See also:
`@BEFORE`, `@AFTER`, `@COPY`, `@CONCAT`, `@SET`

@G <+|->

This script command turns on and off the option to send comments to then NE. A line starting with `@G+` switches off sending comments and `@G-` switches it on. While sending comments is switched on, WinFIOL also sends lines starting with an exclamation mark (!) to an APG4x NE and lines starting with /* to MD110 NE. This value is also possible to set in the Traffic settings file.

@GETDATE {variable} [format]

This script command stores the current date in `{variable}`. The `[format]` string may consist of the following characters:

`y, Y, m, M, d, D, :, ., -, /` and `\`.

If a single `Y`, `M` or `D` is specified, the number indicating the year, month or day respectively will contain 1 to 2 characters. If `YY`, `MM` or `DD` is specified, each number will always contain 2 characters, starting with a zero if necessary. If `YYYY` is specified, the year will consist of 4 characters, starting with the century (19 or 20). If the format string is not specified, it is `YYMMDD` by default.

Examples:

```
Example 1
@GETDATE {date}
@COMMENT The current date is: {date}
```

```
Example 2 (AXE)
@GETDATE {date} YYMMDD
@GETTIME {time} HHMM
@GETDAY {day} SUN MON TUE WED THU FRI SAT
CACLS:DATE={date},TIME={time},DAY={day};
```

```
Example 3 (MD110)
@GETDATE {date} YYYY-MM-DD
@GETTIME {time} HH-MM
CATII:DATE={date},TIME={time};
```

See also:
`@GETTIME`, `@GETDAY`

@GETDAY {variable} [days]

This script command stores the current day of the week in `{variable}`. Optionally, a list of `[days]` may be specified. All 7 days should be listed, starting with Sunday. All days should be separated with spaces or tabs. If the days are not specified, the variable is assigned one of the values 0 to 6, where 0 is Sunday.

Examples:

```
@GETDAY {day}
@INC {day} 1
@COMMENT The current day number is {day}
```

```
Example (AXE)
@GETDATE {date} YYMMDD
@GETTIME {time} HHMM
@GETDAY {day} SUN MON TUE WED THU FRI SAT
CACLS:DATE={date},TIME={time},DAY={day};
```

See also:
@GETTIME, @GETDATE

@GETFILE <url> <filename>

This script command fetches the contents of a file from an URL to the local machine. The URL is entered as a quoted string or as text including variables. The first part of the URL must be `http:` or `ftp:` depending on the access method. If ftp is used, an anonymous ftp transfer is performed unless a user name and password are supplied. Values for user name and password must be stated as shown below.

`@getfile ftp://<username>:<password>@<host>/<remotepath>/[filename] <localpath>/[filename]`

The data fetched from the URL is stored in a local file named `<filename>`. The `<filename>` must be given as a quoted string or as text including variables. If `<filename>` does not include a path, the default directory is used, see `{_CMDDIR}`.

If a filename is not specified in the `<url>` path, a directory listing in HTML format is retrieved. A filename must be provided in the local path to save the directory listing. See Example 4.

Examples:

```
Example 1
@@ Get file from http server
@GETFILE http://www.corporation.xyz/subfile.cmd subfile.cmd
```

```
Example 2
@@ Get file from ftp server
@SET {urlpath}= ftp://fileserver.corporation.xyz/commandfiles/
@SET {filename} = subfile.cmd
@GETFILE {urlpath}{filename} "subfile2.cmd"
```

```
Example 3
@@ Get file from ftp server using user name and password
@SET {filename} = "commandfiles/subfile.cmd"
@SET {user} = "myuser"
@SET {pwd} = "mypassword"
@GETFILE ftp://{user}:{pwd}@ftp.corp.xyz/{filename} "local.cmd"
```

```
Example 4
@@ Get a directory listing in HTML format
@SET {user} = "myuser"
@SET {pwd} = "mypassword"
@GETFILE ftp://{user}:{pwd}@ftp.corp.xyz/path/ "/tmp/tempdirlist.html"
```

See also:
@PUTFILE, @SENDFILE, @INCLUDE, @READ

@GETTIME {variable} [format]

This script command stores the current time in `{variable}` specified. The `[format]` string may consist of the following characters:

`h, H, m, M, s, S, :, ., -, / and \.`

If a single `H`, `M` or `S` is specified, the number indicating the hour, minute or second respectively will contain 1 to 2 characters. If `HH`, `MM` or `SS` is specified, each number will always contain 2 characters, starting with a zero if necessary. If the format string is not specified, it is `HHMMSS` by default.

Examples:

```
@GETTIME {time} H:MM:SS
@COMMENT The current time is: {time}
```

```
@GETDATE {date} YYMMDD
@GETTIME {time} HHMM
@GETDAY {day} SUN MON TUE WED THU FRI SAT
CACLS:DATE={date},TIME={time},DAY={day};
```

```
@GETDATE {date} YYYY-MM-DD
@GETTIME {time} HH-MM
CATII:DATE={date},TIME={time};
```

See also:
@GETDAY, @GETDATE

@GOSUB <label>| {variable}

This script command makes a jump to the subroutine specified by the <label> or {variable} label. When the subroutine returns, transmission continues on the line just after the @GOSUB script command. The label must be defined with @LABEL or @D. The script command @RETURN is used to return from a subroutine.

If the specified label is a variable, the variable must exist and the contents of the variable must be an existing label. Subroutines may be nested. Be sure that the subroutine exists in the block of data being transmitted. It is recommended to place subroutines at the end of a file, to ensure that WinFIOL finds them, when for example selecting a block with Ctrl+B and transmitting it.

Example:

```
Example (AXE)
@SET {lidev} = 3
@SET {seprm} = 19
@GOSUB CONNPRM
@END
! Subroutine: connect private meter !
! Input variables: lidev, seprm !
@LABEL CONNPRM
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
SUPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
@RETURN
```

See also:
@LABEL, @RETURN, @GOTO

@GOTO <label>| {variable}

This script command makes an immediate jump to a <label> or {variable} label. The label must be defined with @LABEL or @D. If the specified label is a variable, the variable must exist and the contents of the variable must be an existing label. It is possible to use the script command @J for short.

Example:

```
Example (AXE)
DPWSP;
@COPY {_line4} {wo} 24 2
@IF {wo} <> "WO" THEN GOTO ready
DPPAI;
@LABEL ready
```

See also:
@LABEL, @GOSUB

@GREP {dest} {variable} <pattern>

This script command finds lines matching the <pattern> in variable `{variable}` and returns the matching lines in variable `{dest}`. The destination variable is undefined if no match was found, or if the variable was undefined prior to using `@GREP`. The pattern string supports escape characters.

Examples:

```
Example 1
@UNSET {found}
@GREP {found} {_lines} ".*HELLO.*"
@IFNDEF {found} THEN GOTO NOTFOUND
@COMMENT Found HELLO
@END
@LABEL NOTFOUND
```

```
Example 2
@SET {var[0]} = "This is a line"
@SET {var[1]} = "This is also a line"
@SET {var[2]} = "This is the third line"
@SET {var[3]} = "A fourth line"
@SET {var[4]} = "Last line"
@GREP {index} {var} "This.*"
! Print the result !
@FOREACH {index} COMMENT Found "This" in line: {_CURRLINE}
## Found "This" in line: This is a line
## Found "This" in line: This is also a line
## Found "This" in line: This is the third line
```

```
Example 3
LASIP:BLOCK=AOT&ADE&AD2;
@GREP {blocks} {_lines} ".*/.*/.*"
@FOREACH {blocks} COMMENT {_currline}
! This is shown in the output window: !
<LASIP:BLOCK=AOT&ADE&AD2;
SOFTWARE UNIT IDENTITY
BLOCK BN BS FCODE ID
AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG R2A02
ADE H'140 ACTIVE 5400/CAAZ 107 0842/M06AE R2A01
AD2 H'4D8 ACTIVE 5600/CAAZ 107 6539/M08E R2A01
END
## AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG R2A02
## ADE H'140 ACTIVE 5400/CAAZ 107 0842/M06AE R2A01
## AD2 H'4D8 ACTIVE 5600/CAAZ 107 6539/M08E R2A01
```

See also:
`@FIND`, `@COUNT`

@HEX {dest} {variable}|<number>

This script command converts a given decimal number to a hexadecimal number. The given number may be a constant or a numeric variable.
Example:

```
@SET {n} = 2346
@HEX {hex_n} {n}
@COMMENT hex_n = {hex_n}
## hex_n = 92A
```

See also:
@DECIMAL

@I <filename> [n]

This script command loads another script file `<filename>` and transmits it, before continuing the transmission in the original file. The included file must exist and if it is already open in the script editor, WinFIOL sends all lines from there, instead of from the file. If a value `[n]` is specified, WinFIOL starts the transmission from line number *n*, otherwise transmission starts from the first line.

It is possible to let WinFIOL pause when the end of the included file is reached, by enabling the "Pause at end of file" option in the traffic setup file.

> To include a script file from an URL, use the command `@INCLUDE`.

See also:
@INCLUDE

@IF <condition> THEN <statement>

@IF {str} MATCHES "reg.expr" THEN <statement>

@IF {str} IN {arr[]} THEN <statement>

@IF {str} IN ["str1","str2","str3",...] THEN <statement>

This script command executes a statement, that is, some other script command, if the condition is met. The following table shows a description of the different options:

`@IF <condition> THEN <statement>`			The condition can be a comparison between the contents of a variable and text or a number, or two variables. The first operand must be a variable, the second may also be a constant. The following operators are defined:
`=`	equal	(text and numbers)	
`<>`	unequal	(text and numbers)	
`>`	greater than	(numbers only)	
`<`	less than	(numbers only)	
`=>`	equal or greater than	(numbers only)	
`<=`	equal or less than	(numbers only)	

`@IF {str} MATCHES "reg.expr" THEN <statement>`	If the regular expression pattern is found in `{str}`, the `<statement>` is executed. `{str}` must be an variable. The regular expression `"reg.expr"`, must be a quoted string.

`@IF {str} IN {arr[]} THEN <statement>` `@IF {str} IN ["str1","str2","str3",...] THEN <statement>`	If `{str}` is equal to a string in the `{arr[]}` indexed variable or a string in the `["str1","str2","str3",...]` string list, the <statement> is executed. `{str}` must be an variable. `{arr}` must be an indexed variable and referenced as `{arr[]}` if a search in the whole array is intended. The syntax `{arr[3]}` searches from index 3 to the end of the array. If a string list is used, it must use square brackets and contain quoted strings separated by comma.

A statement must follow the keyword `"THEN"`. The statement must be at the same line as the IF command, and it must be a script command, with or without the @ sign.

Examples:

```
Example 1 (AXE)
DPWSP;
@SET {state} = 0
@COPY {_line4} {wo} 8 5
@IF {wo} = "WO" THEN SET {state} = 1
@IF {wo} = "SB" THEN SET {state} = 2
@IF {wo} = "EX-A" THEN SET {state} = 3
@IF {wo} = "EX-B" THEN SET {state} = 4
@IF {state} = 1 THEN GOTO ready
...
@LABEL ready
```

```
Example 2
@SET {str} = "This is a test string"
@IF {str} MATCHES ".*tri.*" THEN @COMMENT Match found
```

```
Example 3
@SET {str[0]} = "monday"
@SET {str[1]} = "tuesday"
@SET {str[2]} = "wednesday"
@SET {str[3]} = "thursday"
@SET {str[4]} = "friday"
@SET {s} = "tuesday"
@IF {s} IN {str[]} THEN @COMMENT Match found
```

See also:
`@IFERROR`, `@IFDEF`, `@IFNDEF`

@IFDEF {variable} THEN <statement>

This script command executes `<statement>`, that is, some other script command, with or without the @ sign, if the variable `{variable}` is defined.

Example:

```
@SET {var} = "Hello"
@IFDEF {var} THEN COMMENT Variable exist
## Variable exist
@UNSET {var}
@IFDEF {var} THEN COMMENT Variable exist
@COMMENT Variable not defined
## Variable not defined
```

See also:
`@IFNDEF`, `@SET`, `@UNSET`

@IFERROR THEN <statement>

This script command executes `<statement>`, that is, some other script command, if WinFIOL detects an error. The statement must be a script command, with or without the @ sign, and must be on the same line as the `IFERROR` command.

Example:

```
Example (AXE)
BLEEE:EMG=RSS0,EM=0;
@IFERROR THEN GOSUB repair_emrp
...
@LABEL repair_emrp
```

See also:
`@IF`, `@ONERROR`

@IFNDEF {variable} THEN <statement>

This script command executes <statement>, that is, some other script command, if the variable `{variable}` is undefined. The statement must be a script command, with or without the @ sign, and must be on the same line as the `IFNDEF` command.

Example:

```
Example
@SET {var} = "Hello"
@IFNDEF {var} THEN COMMENT Variable does not exist
@UNSET {var}
@IFNDEF {var} THEN COMMENT Variable does not exist
@@COMMENT Variable is defined
```

See also:
@IFDEF, @SET, @UNSET

@INC {variable} [n]

This script command increments the numeric variable {variable} by *n*. The variable must exist and be numeric. If n is not specified, the variable is increased by 1.

Example:

```
Example (AXE)
@SET {dev} = 16
@SET {loop} = 0
@LABEL again
EXDEP:DEV=BT3-{dev};
@INC {dev} 32
@INC {loop}
@IF {loop} < 3 THEN GOTO again
```

See also:
@SET, @DEC, @LOOP

@INCLUDE <filename>|<url>

This script command loads another script file, and transmits it, before continuing transmission in the original file. The included file must exist and if it is already open in the script editor, WinFIOL sends all lines from there, instead of from the file. Transmission starts at the first line. It is possible to let WinFIOL pause when the end of the included file is reached, by enabling the "Pause at end of file" option in the traffic setup file.

The included file is specified using <filename> or <url>. If a local filename does not include a path, the directory defined by the variable {_CMDDIR} is used. The file specification may contain variables. Any existing variables are not transferred to the include file, unless the script commands @PARAM or @PARAMASK are used. The first part of the URL must be http: or ftp: depending on the access method. If ftp is used, an anonymous ftp transfer is performed unless user credentials are supplied. Values for user name and password must be stated as shown below.

```
@INCLUDE ftp://<username>:<password>@<host>/<remotepath>/[filename]
```

The data fetched from an URL is temporarily stored in the default directory in a file named _url_data.cmd.

> To include a script file from a predefined line in that file, @I should be used.

Examples:

```
Example 1
@@ Include file from default directory
@INCLUDE subfile.cmd
```

```
Example 2
@@ Include file from specific directory
@INCLUDE /tmp/myfiles/subfile.cmd
@@ NOTE: If the include path contains space characters, quotes
must be used
@INCLUDE "/tmp/my files/subfile.cmd"
```

```
Example 3
@@ Include file from http server
@INCLUDE http://www.corporation.xyz/subfile.cmd
```

```
Example 4
@@ Include file from ftp server
@INCLUDE ftp://fileserver.corporation.xyz/commandfiles/subfile.cmd
```

```
Example 5
@@ Include file from ftp server
@INCLUDE ftp://<user>:<password>@fileserver.corporation.xyz/commandfiles/subfile.cmd
```

See also:
`@GETFILE`, `@SENDFILE`, `@PARAM` or `@PARAMASK`

@INSERT {string} <position> <sub-string>

This script command inserts the `<sub-string>` at the `<position>` in the `{string}` variable. The first position in the string is 1. If the given position is greater than the length of the string, the sub-string is appended to the end.

The position and sub-string may contain variables.

Examples:

```
Example 1
@SET {str} = "source string"
@INSERT {str} 1 "This is the "
@COMMENT {str}
## This is the source string
```

```
Example 2
@SET {str} = "aa cc"
@INSERT {str} 4 "bb "
@COMMENT {str}
## aa bb cc
```

```
Example 3
@SET {str} = "the "
@INSERT {str} 600 "end"
@COMMENT {str}
## the end
```

See also:
`@REPLACE`, `@AFTER`, `@BEFORE`

@ITEM {dest} <source> <delimiters> <n>

This script command extracts the <n>:th sub-string separated by `<delimiters>` from the `<source>` variable or string, and saves it into variable `{dest}`. The first sub-string is n=0, the second n=1, and so forth. One or multiple delimiters, single characters, may be specified. Escape characters may be used when specifying a delimiter. Variable `{dest}` is an empty string if no matching sub-string is found.

Examples:

```
Example 1
@ITEM {dest} "This is the source string" " " 0
@COMMENT SubStr: {dest}
## SubStr: This
```

```
Example 2
@ITEM {dest} "This is the source string" " " 2
@COMMENT SubStr: {dest}
## SubStr: the
```

```
Example 3
@ITEM {dest} "This is the source string" "X" 0
@COMMENT SubStr: {dest}
## SubStr: This is the source string
```

```
Example 4
@ITEM {dest} "This is the source string" "X" 1
@COMMENT SubStr: '{dest}'
## SubStr: ''
```

```
Example 5
@SET {source} = "AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG R2A02"
@ITEM {prefix} {source} " /" 3
@ITEM {caa} {source} "/" 1
@ITEM {rev} {source} " /" 8
@COMMENT Prefix: {prefix}
## Prefix: 5400
@COMMENT CAA: {caa}
## CAA: CAAZ 107 0841
@COMMENT Rev: {rev}
## Rev: R2A02
```

See also:
@RITEM, @AFTER, @BEFORE, @COPY

@K <HH:MM:SS>

This script commands suspends the transmission until the specified time of day, <hh:mm:ss>, has been reached. You may omit the colons in the time specification.

See also:
@M, @T

@L <[+]filename|+|->

This script command turns on or off logging of output. If <filename> does not include a path, the directory defined by the variable {_LOGDIR} is used. Script command @LOG is possible to use similarly.

The below table shows the different options for @L:

Command syntax	Description
@L <filename>	Open a log file with the specified name. If a file with the same name exists, it is overwritten.
@L +<filename>	Append a log file with the specified name. If the log file does not exist, it is created.
@L+	Reopen and append the previously opened log file.
@L-	Close the log file. The commands @CLOSE and @C can be used similarly.

See also:
@CLOSE, @LOG

@LABEL <label>

This script command defines a <label> that exists during transmission of the script file it is defined in. A jump to a label is made using @GOTO, @GOSUB, @C, and @J. When a jump to a label is made, WinFIOL searches for the label in the file, or selected text being transmitted. You can define as many labels as you want, but each label must be defined only once. Labels are case-insensitive and they must have unique names.

Use @D for short.

Examples:

```
@SET {dev} = 0
@LABEL again
STDEP:DEV=LI-{dev};
@INC {dev} 1
@IF {dev} < 3 THEN GOTO again
```

See also:
@C, @GOTO, @GOSUB

@LENGTH <string> {length}

This script command determines the length of a string or the variable <string>. The string may be enclosed in double quotes for clarity. The length of the string or variable is stored as a numeric value in the variable {length}.

Examples

```
@GETDATE {date} YYYY-M-D
@LENGTH {date} {length}
@LENGTH "This is the string" {length}
```

See also:
@SET, @SCAN, @COPY

@LOG ON <filename>|APPEND <filename>|<OFF>

This script command turns on or off logging of output. If <filename> does not include a path, the directory defined by the variable {_LOGDIR} is used.

The following table shows a description of the different options:

@LOG ON <filename>	When using ON to open a log file, a new file is created. If a file with the same name already exists, the existing file is truncated without a warning. <filename> must be a valid file name.
@LOG APPEND <filename>	Append data to en already existing file. The file is created if it does not exist.
@LOG OFF	Close a log file

The log file specification may contain variables and environment variables.

Script command @L works similarly and facilitates reopening of the previous log file.

Examples:

```
Example 1 (AXE)
@LOG ON "/tmp/pcorp.log"
PCORP:BLOCK=ALL;
@LOG OFF
```

```
Example 2
@LOG APPEND "/tmp/username.log"
IOSTP;
@LOG OFF
```

```
Example 3
@GETDATE {date} YYYYMMDD
@LOG APPEND {date}.log
LASIP:BLOCK=ALL;
@LOG OFF
```

See also:
@CLOSE, @L

@LOOP <n>

This script command repeats the commands between @LOOP and @ENDLOOP <n> number of times. If the argument <n> is zero, or less then zero, all commands until the @ENDLOOP command are skipped. <n> may be a numeric variable.

Loops can be nested up to 10 levels deep.

Example:

```
Example (AXE)
@LOOP 5
@T 10
PLLDP;
@ENDLOOP
```

Other types of loops are possible to implement using the commands @WHILE, @ENDWHILE, @LABEL, @GOTO, @SET, @INC and @IF.

See also:
@ENDLOOP, @WHILE, @ENDWHILE

@LOWCASE {variable}

This script command converts the variable <string> to lowercase.

Example:

```
Example
@SET {var} = "Hello Over There"
@LOWCASE {var}
! Print the result !
@COMMENT {var}
## hello over there
```

See also:
@UPCASE

@M <n>

This script command suspends the transmission in <n> minutes. The maximum value for n is 32767.

See also:
@K, @T

@MERGE {variable} = {source} [<oper> {variable_n} ...]

This script command merges variable {source} and other optional variables {variable_n} or strings into {variable}. The operand <oper> is one of +, -, *, / and |. The operations may be mixed and are performed in left to right order. '

The following table shows the six different operations that are supported towards array variables:

<oper>		Usage	Result in {var}
=	Copy	{M} = {M1}	M contains all elements in M1
+	Append	{M} = {M1} + {M2}	M contains all elements in M1 and all elements in M2
-	Remove equal lines	{M} = {M1} - {M2}	M contains the elements found in M1 minus the ones found in M2
*	Union (*)	{M} = {M1} * {M2}	M contains all unique elements found in both M1 and M2
/	Intersection	{M} = {M1} / {M2}	M contains elements found in both M1 and M2.
|	Difference	{M} = {M1} | {M2}	M contains the elements that are unique to M1

The remove operation (-) removes all elements currently present in {variable} that are exactly matched by an element in the variable following the subtraction operator. The source variables must exist. The automatic variable {_LINES} can be used as source variable. The destination variable is created if it does not exist. If a source variable is empty, the destination variable will also be empty.

Examples:

```
Example 1
@@ Copy received text to variable "po"
@MERGE {po} = {_LINES}
```

```
Example 2
@@ Use strings as well as variables
@SET {x} = "x"
@MERGE {a} = "a" + "a" + "a" + "a"
@MERGE {b} = "b" + "b" + "b" + "b"
@MERGE {x1} = {x} + "y" + {x} + "y"
@PASTE {dest} {a} {b} {x1} sep ":"
@COMMENT --- Dest ---
@FOREACH {dest} COMMENT {_currline}
! The result printed from above script is: !
## a:b:x
## a:b:y
## a:b:x
## a:b:y
```

```
Example 3
@@ Append received text to variable "data"
@MERGE {data} = {data} + {_LINES}
@WRITE {data} "filename.txt"
@SIZE {data} {linecount}
@COMMENT Wrote {linecount} lines. Second line was {data[1]}
```

```
Example 4
@@ Remove lines exactly matching "TIME"
@SET {banned[2]} = "TIME"
CACLP;
@MERGE {cleaned} = {_LINES} - {banned[2]}
```

```
Example 5
@@ Delete exchange headers from a printout
@GREP {headers} {_lines} ".*TIME.*PAGE.*"
@MERGE {cleaned} = {_lines} - {headers}
```

```
Example 6
@@ Match against multiple alternatives. See also @IF..IN
@ASK {answer} Continue?
@IF {answer} = "" THEN GOTO END
@SET {yes[0]} = "y"
@SET {yes[1]} = "Y"
@SET {yes[2]} = "yes"
@SET {yes[3]} = "Yes"
@SET {yes[4]} = "YES"
@MERGE{empty} = {answer} - {yes}
@IF {empty} <> "" THEN GOTO END
@COMMENT Continuing...
```

```
Example 7
@SET {M1[0]} = "1"
@SET {M1[1]} = "2"
@SET {M1[2]} = "3"
@SET {M1[3]} = "4"
@SET {M2[0]} = "3"
@SET {M2[1]} = "4"
@SET {M2[2]} = "5"
@SET {M2[3]} = "6"
@FOREACH {M1} COMMENT M1: {_currline}
@FOREACH {M2} COMMENT M2: {_currline}
@COMMENT M1 + M2
@MERGE {M} = {M1} + {M2}
@FOREACH {M} COMMENT {_currline}
@COMMENT M1 - M2
@MERGE {M} = {M1} - {M2}
@FOREACH {M} COMMENT {_currline}
@COMMENT M1 * M2
@MERGE {M} = {M1} * {M2}
@FOREACH {M} COMMENT {_currline}
@COMMENT M1 / M2
@MERGE {M} = {M1} / {M2}
@FOREACH {M} COMMENT {_currline}
@COMMENT M2 / M1
@MERGE {M} = {M2} / {M1}
@FOREACH {M} COMMENT {_currline}
@COMMENT M1 | M2
@MERGE {M} = {M1} | {M2}
@FOREACH {M} COMMENT {_currline}
@COMMENT M2 | M1
@MERGE {M} = {M2} | {M1}
@FOREACH {M} COMMENT {_currline}
! The result printed from above script is: !
## M1: 1
## M1: 2
## M1: 3
## M1: 4
```

```
## M2: 3
## M2: 4
## M2: 5
## M2: 6
## M1 + M2
## 1
## 2
## 3
## 4
## 3
## 4
## 5
## 6
## M1 - M2
## 1
## 2
## M1 * M2
## 1
## 2
## 3
## 4
## 5
## 6
## M1 / M2
## 3
## 4
## M2 / M1
## 3
## 4
## M1 | M2
## 1
## 2
## M2 | M1
## 5
## 6
```

See also:
`@COUNT`, `@GREP`, `@READ`, `@WRITE`, `@SIZE`

@ONERROR [statement]

This script command executes a `[statement]`, that is, some other script command, every time WinFIOL detects an error. The statement must be on the same line as the `@ONERROR` command, and it must be a script command, with or without the @ sign. The syntax of the statement is checked only when an error really occurs. This script command needs only to be defined once, unlike `@IFERROR`.

To clear the error handling, specify `@ONERROR` without the statement.

Example:

```
Example (AXE)
@ONERROR GOSUB repair_emrp
BLEEE:EMG=RSS0,EM=0;
BLEEE:EMG=RSS0,EM=1;
@ONERROR ! clear error handling
...
@LABEL repair_emrp
```

See also:
`@IFERROR`, `@ONRECEIVE`

@ONRECEIVE [text [statement]]

This script command checks every line in the received data for the specified `[text]` string. If a line is found containing the string, the specified statement is executed. The syntax to define a check is:

`@ONRECEIVE <string> <statement>`

Where <string> is a string, enclosed in double quotes, or a string variable. The statement must be on the same line as the `@ONRECEIVE` command, and it must be a script command, with or without the @ sign.

This script command needs to be defined only once, like @ONERROR, but may be specified multiple times using different texts, allowing for different checks simultaneously. It is possible to define various error checking routines, using for example @GOSUB to jump to a subroutine, where the text received maybe further analyzed.

> The variable {_ONRCVLINE} is automatically defined with the value of the current line number. The value of this variable does not change until the next time a specified text is received. See Automatic variables how other automatically generated variables in WinFIOL can be used in script commands.

A specific check is cleared by only specifying the text and omitting the statement. All checks is cleared at once, by omitting both the text and the statement.

Example:

```
Example (AXE)
@CLEAR ! clear any old "onreceive"
@PRESERVE ! preserve all new "onreceive"
@ONRECEIVE "SIZE ALTERATION RESULT" GOSUB saa_check
SAAII:SAE=500,BLOCK=KR2,NI=32;
SAAII:SAE=306,NI=200;
...
@ONRECEIVE "SIZE ALTERATION RESULT" ! clear error handling
@END
@LABEL saa_check
@WAITFOR /END/
@COPY {_line3} {fcode} 21 2
@IF {fcode} = "" THEN RETURN
@SET {msg}=Size alteration error FCODE {fcode} on line {_ONRCVLINE}
@COMMENT {msg}
@SETERROR {msg}
@RETURN
```

See also:
@ONERROR, @/text/

@OPENCHAN <channel-file> [node]

> @OPENCHAN is only supported in the GUI version of WinFIOL

This script command opens a channel and continues the transmission in that channel. The argument <channel-file> must be the name of the channel property file for the NE, or a variable defined with the name. If <channel-file> does not include a path, the directory defined by the variable {_CHNDIR} is used.

The current script file is opened in the new channel window and the transmission continues from there.

Example:

```
Example (AXE)
Work on the current channel is finished. A new channel is opened and the previous channel be
closed.
@@ Save channel number for closing
@SET {prev_chan} = {_CHANNEL}
@OPENCHAN channel_name
@@ Close old channel
@CLOSECHAN {prev_chan}
@@ Do actions on new channel
@@ Open and switch to another channel's node B
@OPENCHAN another_channel B
```

> *@NODE is currenty not supported in WinFIOL on Linux*
>
> *An optional node specification can be given to select one of the alternatives ACTIVE, A or B node, if the following conditions are met:*
>
> - *The NE must be an APG40*
> - *The node ip-adresses must be defined in the channel properties for the NE*

See also:
@CLOSECHAN, @CHANNEL, @SCRIPTLOG, @SAVECHAN, @CLONE

@PARAM {variable}

This script command imports a {variable} from a parent script file, that is, the file that included the current script file, using the @INCLUDE command. An error occurs if the variable does not exist in the parent file. Changing the value of the variable does not affect the variable in the parent file.

Example:

```
Example (AXE)
@PARAM {lidev}
@PARAM {seprm}
@COMMENT Connecting private meter {seprm} to device {lidev}
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
SUPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
@EXIT
```

See also:
@SET, @PARAMASK

@PARAMASK {variable} <question>

@PARAMASK is only supported in the GUI version of WinFIOL

This script command imports a {variable} from a parent script file, that is, the file that included the current script file, using the @INCLUDE command. If the variable does not exist in the parent script file, a dialog box appears, in which the user may type a value for the variable specified. Changing the value of the variable does not affect the variable in the parent script file. When the dialog box is displayed the transmission is temporarily paused.

Example:

```
Example (AXE)
@PARAMASK {lidev} Give number of LI device
@PARAMASK {seprm} Give number of SEPRM device
@COMMENT Connecting private meter {seprm} to device {lidev}
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
SUPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
@EXIT
```

See also:
@SET, @PARAM, @ASK, @INCLUDE

@PASTE {dest} {source} [..] [SEP <separator>]

This script command appends the columns found in variable {source} to the variable {dest}. It is possible to specify multiple source variables. The order of appending is left to right.

A <separator> separates the columns. The default column separator is a space character. Use SEP to specify a different column separator. The separator must be enclosed in double quotation marks, "s", or given as a variable, and may consist of one single character or a string. Escape characters may be used as separators.

Example:

```
Example
IOIOP:IO1=ALL;
@GREP {ioiop} {_lines} ".*-.*"
@CUT {dev} {ioiop} COL 1
@CUT {state} {ioiop} POS 60-64
```

```
@PASTE {dest} {state} {dev} SEP ":"
@FOREACH {dest} COMMENT {_currline}
## IDLE:AMTP-0
## IDLE:TW-0
## IDLE:AT-0
## IDLE:AT-5
## IDLE:AT-1
## IDLE:AT-4
```

See also:
@CUT, @FIND, @GREP

@PAUSE [reason]

When running script files in WinFIOL CLI it is not possible to resume a paused transmission manually, hence @PAUSE is only supported in the GUI version of WinFIOL.

This script command pauses the transmission. If a [reason] is specified, it is shown in the status line in the script area. The reason should be maximum 30 characters.

Example:

```
Example
! Plug the cable back in;
@PAUSE
```

See also:
@END, @HALT, @STOP

@PRESERVE

This script command preserves all variables already defined, and all that are defined later. Even if the transmission is stopped, the variables are preserved for the next transmission. This is useful when running a long script file where variables are set in the beginning or during run-time, and the transmission is stopped due to an error. It is then possible to continue the script from a selected position, without having lost the variables that is reused later in the script.

All variables, including preserved variables from previous transmissions, is undefined using @CLEAR script command. It is recommended to define @CLEAR on the first and last line of the script file when using @PRESERVE.

Example:

```
Example
@CLEAR
@PRESERVE
! Start a lengthy and complex transmission;
! Define variables first;
...
@LABEL end
@CLEAR
```

See also:
@CLEAR

@PUTFILE <filename> <url>

This script command transfers a local file to a remote host using an URL as destination. The URL is entered as a quoted string or as text including variables. The first part of the URL must be http: or ftp: depending on the access method. If ftp is used, an anonymous ftp transfer is performed unless a user name and password is supplied. Values for user name and password must be stated as shown below.

```
@putfile <localpath>/[filename] ftp://<username>:<password>@<host>/<remotepath>/[filename]
```

The local <filename> must be given as a quoted string or as text including variables. If <filename> does not include a path, the default directory is used, see {_CMDDIR}.

Examples:

```
Example 1
@@ Transfer file to ftp server
@PUTFILE "script.cmd" ftp://www.corporation.xyz/new_name.cmd
```

```
Example 2
@@ Transfer file using user name and password
@SET {host}= " fileserver.corp.com"
@SET {user}= "ftpuser"
@SET {password}= "secret"
@SET {localfile} = "subfile.cmd"
@SET {remotefile}= "cmdfiles/subfile.cmd"
@PUTFILE {localfile} ftp://{user}:{password}@{host}/{remotefile}
```

See also:
@GETFILE, @SENDFILE, @WRITE

@QUIET

This script command switches off the setting to display result messages in the output window. This setting is off by default and it is switched on with @VERBOSE.

Example:

```
Example (AXE)
@VERBOSE
@COPY "Search target string" {dest} 8 6
@QUIET
```

See also:
@VERBOSE

@R <+|->

This script commands sets the option "Pause on error". A line starting with @R+ switches on and @R- switches off pause on error. When pause on error is switched on, WinFIOL automatically pauses the transmission when an error message is received. Pause on errors is on by default, but is possible to change in the traffic settings file.

> ⚠ When running script files in WinFIOL CLI it is not possible to resume a paused transmission manually.

@READ <filename>|{variable} {dest}

This script command reads data from a file into an array variable. The <filename> may be given as a quoted string or a variable. If the filename does not include a path, the default directory {_CMDDIR} is used.

The data is stored with one line from the file per array element. If the text file is empty, the variable will be empty.

Example:

```
Example (AXE)
@@ Display a text file
@READ "/tmp/saved.txt" {data}
@SIZE {data} {linecount}
@SET {i} = 0
@LOOP {linecount}
@COMMENT {data[{i}]}
@INC {i}
@ENDLOOP
```

See also:
@WRITE, @SIZE, @SENDFILE

@RELEASE

This script command tries to release the terminal at the next prompt. It is also possible to use @E or @X for short.

Example:

```
Example (AXE, auto release and reconnect are off)
LABUP;
@RELEASE
@WAITFOR /END/
@CONNECT
```

See also:
@CONNECT

@REPLACE {string} <from-string> <to-string>

This script command replaces all occurrences of the <from-string> with the <to-string> in the {string} variable. The strings may contain variables and the match is case sensitive.

Examples:

```
Example 1
@SET {str} = "aa bb cc dd"
@REPLACE {str} "bb" "xx"
@COMMENT {str}
## aa xx cc dd
```

```
Example 2
@SET {str} = "aa bb aa bb"
@REPLACE {str} "aa" "kk"
@COMMENT {str}
## kk bb kk bb
```

See also:
@INSERT, @SCAN

@RESTOREPROMPT

This script command reverses a previous @SETPROMPT command, that is, it restores the original prompt set-up for an NE in WinFIOL.

Examples:

```
Example 1
! Restore default prompts !
@RESTOREPROMPT
```

```
Example 2
! Dump the AP event log which may contain dummy prompts !
echo "Dumping event log file"
cd /d C:\
cd C:\temp
! Write the event data to the file log.txt !
dumplog application
! Change the prompt on the target system !
prompt WINFIOL$G
! Make WinFIOL check for only this prompt !
@SETPROMPT "WINFIOL>"
type log.txt
! Restore the default prompts !
@RESTOREPROMPT
! Restore the prompt on the target !
prompt $P$G
echo "--- Finished ---"
```

See also:
@SETPROMPT, @WAITPROMPT

@RETURN

This script command exits a subroutine and jumps back to the line after @GOSUB, from where it was called. Subroutines may be nested.

Example:

```
Example (AXE)
@SET {lidev} = 3
@SET {seprm} = 19
@GOSUB CONNPRM
@END
! Subroutine: connect private meter !
! Input variables: lidev, seprm !
@LABEL CONNPRM
BLODI:DEV=LI-{lidev};
BLODI:DEV=SEPRM-{seprm};
SUPMI:DEV=LI-{lidev},DEV=SEPRM-{seprm};
BLODE:DEV=SEPRM-{seprm};
BLODE:DEV=LI-{lidev};
@RETURN
```

See also:
@GOSUB

@RITEM {dest} <source> <delimiters> <n>

This script command extracts the <n>:th sub-string starting from the right, separated by `<delimiters>` from the `<source>` variable or string, and saves it into variable `{dest}`. The first sub-string from the right is n=0, the second n=1, and so forth. One or multiple delimiters, single characters, may be specified. An Escape character may be used as delimiter. Variable `{dest}` is an empty string if no matching sub-string is found. The strings may contain variables.

Examples:

```
Example 1
@RITEM {dest} "This is the source string" " " 0
@COMMENT SubStr: {dest}
## SubStr: string
```

```
Example 2
@RITEM {dest} "This is the source string" " " 2
@COMMENT SubStr: {dest}
## SubStr: the
```

```
Example 3
@RITEM {dest} "This is the source string" "X" 0
@COMMENT SubStr: {dest}
## SubStr: This is the source string
```

```
Example 4
@RITEM {dest} "This is the source string" "X" 1
@COMMENT SubStr: '{dest}'
## SubStr: ''
```

```
Example 5
@SET {source} = "AOT H'146 ACTIVE 5400/CAAZ 107 0841/M06AG
R2A02"
@RITEM {prefix} {source} " /" 5
@RITEM {caa} {source} "/" 1
@RITEM {rev} {source} " /" 0
@COMMENT Prefix: {prefix}
## Prefix: 5400
@COMMENT CAA: {caa}
## CAA: CAAZ 107 0841
@COMMENT Rev: {rev}
## Rev: R2A02
```

See also:
@ITEM, @AFTER, @BEFORE, @COPY

@ROUND {variable}

This script command returns the rounded value of the given numerical value in {variable}.

Examples:

```
Example 1
@CALC {num} = 12.34
@COMMENT before: {num}
## before: 12.34
@ROUND {num}
@COMMENT after: {num}
## after: 12
```

```
Example 2
@CALC {num} = 12.54
@COMMENT before: {num}
## before: 12.54
@ROUND {num}
@COMMENT after: {num}
## after: 13
```

```
Example 3
@CALC {num} = -12.34
@COMMENT before: {num}
## before: -12.34
@ROUND {num}
@COMMENT after: {num}
## after: -12
```

```
Example 4
@CALC {num} = -12.54
@COMMENT before: {num}
## before: -12.54
@ROUND {num}
@COMMENT after: {num}
## after: -13
```

See also:
@CALC, @ABS, @SET

@SAVECHAN {result} {variable}|"filename"

This command saves the active channel. A file name is provided as a {variable} or in plain text inside double qoutes. If the file name is omitted the saved channel file gets the same name as the channel alias. The channel is opened using @OPENCHAN {result}.

@SAVECHAN extends the script command @CLONE, that is deprecated. @CLONE will still function to support legacy script files.

```
@savechan {result}
@comment Saved file: {result}
@openchan {result}

@set {filename} = "saved_channel.xml"
@savechan {result} {filename}
@comment Saved file: {result}
@openchan {result}

@savechan {result} "saved_channel.xml"
@comment Saved file: {result}
@openchan {result}
```

See also:
@CLONE, @OPENCHAN

@SCAN {string} <sub-string> {pos}

This script command scans `{string}` variable for `<sub-string>`. If the sub-string is found, the position is stored in `{pos}` as a numeric value. If the sub-string is not found, `{pos}` be zero. The scan is case sensitive.

Example:

```
@SET {var1} = Search target string
@SCAN {var1} "target" {pos}
@COMMENT Target found at position {pos}
```

See also:
`@SET`, `@COPY`, `@ITEM`

@SCRIPTLOG <ON filename>|<APPEND filename>|<OFF>

> `@SCRIPTLOG` is only supported in the GUI version of WinFIOL.

This script command turns on or off logging of output in the same manner as @LOG script command, except that this may contain logging data from several channels. This is of practical use when the script changes the transmitting channel. If `<filename>` does not include a path, the directory defined by the variable `{_LOGDIR}` is used.

The following table shows a description of the different options:

Syntax	Description
`@SCRIPTLOG ON <filename>`	When using `ON` to open a log file, a new file is created. If a file with the same name already exists, the existing file is truncated without a warning. `<filename>` must be a valid file name.
`@SCRIPTLOG APPEND <filename>`	Append data to en already existing file. The file is created if it does not exist.
`@SCRIPTLOG OFF`	Close the log file

The log file specification may contain variables and environment variables. Note that a script variable within quotation marks is not expanded.

Examples:

```
## Example 1
@SCRIPTLOG ON allip_on_two_channels.log
@CHANNEL 1
ALLIP;
@CHANNEL 2
ALLIP;
@SCRIPTLOG OFF
```

```
## Example 2
@SET {testcase}=”TC-CID-C1C1-0075”
@CONCAT {logfilename} {testcase} “.log”
@SET {slash} = “/”
@CONCAT {localdir} “/WirelineServices/BT21CTSS/Logs” {slash}
@CONCAT {full_logfilename} {localdir} {logfilename}
@SCRIPTLOG ON {full_logfilename}
```

See also:
@LOG, @CHANNEL, `@OPENCHAN`, `@PORTNO`

@SELECTCHAN {dest} <title>

> `@SELECTCHAN` is only supported in the GUI version of WinFIOL

This script command displays a browse dialog where the user may select a channel. The channel identity is returned in the variable `{dest}` and can be used in calls to `@OPENCHAN`. If no selection was made the destination variable is undefined.

The `<title>` may contain variables.

Example:

```
@SELECTCHAN {target} "Select new target"
@IFNDEF {target} THEN GOTO NOSELECTION
@OPENCHAN {target}
..
@LABEL NOSELECTION
```

See also:
@ASK, @FORM, @IFNDEF, @OPENCHAN, @SELECTFILE

@SELECTFILE {dest} <title> [<initial directory>] [<file filter>]

> @SELECTFILE is only supported in the GUI version of WinFIOL

This script command displays a browse dialog where the user may select files. The file or files are returned in the variable {dest}. If no selection is made the destination variable is undefined. The <title> may contain variables.

Specify a directory to change the initial directory, which is by default {_CMDDIR}. The value for the initial directory support environment variables. Specify custom file filters to show only relevant files. The file filter "All files (*.*)" is always present. A file filter is specified in string pairs. The first string represents the text shown in the browse dialog and the second string specifies the filter to use. All strings shall be separated by a semicolon (;). The title, initial directory and file filter may contain variables.

Examples on different file filters:

```
"Text files only;*.txt"
"Batch files (*.bat);*.bat;Script files (*.cmd);*.cmd"
"Channel files;*.chn;Script files;*.cmd;Log files;*.log"
"Text files starting with secret;secret*.txt"
```

Examples:

```
Example 1
@SELECTFILE {filelist} "Select a file"
@IFNDEF {filelist} THEN GOTO NOFILESELECTED
@SIZE {filelist} {cnt}
@COMMENT Selected {cnt} files:
@FOREACH {filelist} COMMENT "Selected file: " {_CURRLINE}
@EXIT

@LABEL NOFILESELECTED
@COMMENT No file selected
```

```
Example 2
@SELECTFILE {filelist} "Select another file" "/tmp"
@IFNDEF {filelist} THEN GOTO NOFILESELECTED
@FOREACH {filelist} COMMENT "Selected file: " {_CURRLINE}
@EXIT

@LABEL NOFILESELECTED
@COMMENT No file selected
```

```
Example 3
@SELECTFILE {filelist} "Select file" "/tmp/My Documents" "Text files;*.txt"
@IFNDEF {filelist} THEN GOTO NOFILESELECTED
@FOREACH {filelist} COMMENT "Selected file: " {_CURRLINE}
@EXIT

@LABEL NOFILESELECTED
@COMMENT No file selected
```

```
Example 4
@SELECTFILE {filelist} "Select file" {_logdir} "Log files;*.log"
@IFNDEF {filelist} THEN GOTO NOFILESELECTED
@FOREACH {filelist} COMMENT "Selected file: " {_CURRLINE}
@EXIT
```

```
@LABEL NOFILESELECTED
@COMMENT No file selected
```

See also:
@ASK, @FORM, @IFNDEF, @FOREACH, @SELECTCHAN

@SENDFILE [TEXT|BINARY] <file>

This script command transmits the content of the specified <file> to the NE. The file can be transmitted as text or as binary data by using the options TEXT or BINARY. Default is TEXT mode. The strings may contain variables.

Examples:

```
Example 1
! Send a text file !
@SENDFILE "mytextfile.txt"
@SENDFILE text "mytextfile.txt"
```

```
Example 2
! Send a binary file !
@SENDFILE BINARY "data.dat"
```

```
Example 3
! Send a text file using copy con on a Windows target host !
! @SENDHEX 26 terminates the input mode !
@SELECTFILE {filelist} "Select a file"
@IFNDEF {filelist} THEN GOTO NoFileSelected
@WAITPROMPT off
copy con datafile.dat
@SENDFILE {filelist}
@SENDHEX 26
@WAITPROMPT on
@CONNECT
@COMMENT FILE DATA TRANSFERRED OK
@END
@LABEL NoFileSelected
@COMMENT NO FILE SELECTED
```

```
Example 4
! Send a text file using copy con on a Unix target host !
! @SENDHEX 4 terminates the input mode !
@SELECTFILE {filelist} "Select a file"
@IFNDEF {filelist} then goto NoFileSelected
@WAITPROMPT off
cat > datafile.dat
@SENDFILE {filelist}
@SENDHEX 4
@WAITPROMPT on
@CONNECT
@COMMENT FILE DATA TRANSFERRED OK
@END
@LABEL NoFileSelected
@COMMENT NO FILE SELECTED
```

See also:
@SENDHEX

@SENDHEX <hh>

This script command transmits the hexadecimal value <hh>. The value is sent immediately.

Supported formats are:

- Decimal number (0-255)
- Hexadecimal number (h'00-h'ff)
- Variable {variable}

- String "value"

If the given variable is an array the whole array is sent. String characters not recognized as hexadecimal characters are sent unchanged. Hexadecimal values in strings are specified using `"\nnn"` where nnn represents the hexadecimal value. See further in section escape characters.

Examples:

```
Example 1
! Connect to ERIOS (Ctrl+E)!
@SENDHEX 5
```

```
Example 2
! Four different ways of sending data !
@SENDHEX "HELLO" 32 "THERE" h'd "\00a"
```

```
Example 3
! Send CRLF as variable !
@FORMAT {CRLF} "\00d\00a"
@SENDHEX "This string ends with CRLF" {CRLF}
```

```
Example 4
! Send an array of commands !
@FORMAT {CRLF} "\00d\00a"
@FORMAT {EOF} "\01a%s" {CRLF}
@MERGE {cmds} = "del test.txt" + {crlf}
@MERGE {cmds} = {cmds} + "copy con test.txt" + {crlf}
@MERGE {cmds} = {cmds} + "Line 1" + {crlf}
@MERGE {cmds} = {cmds} + "Line 2" + {crlf}
@MERGE {cmds} = {cmds} + "Line 3" + {crlf}
@MERGE {cmds} = {cmds} + {eof}
@MERGE {cmds} = {cmds} + "type test.txt" + {crlf}
@SENDHEX {cmds}
```

See also:@SENDFILE

@SET {variable} = <expression>

This script command assigns a value to a `{variable}` or an array element. Array elements are accessed by using square brackets `[n]`, indexes start from 0 (zero). The variable is automatically initialized as text or numeric variable, depending on the type of expression. The following table shows how to define an `<expression>`:

<expression>	Description
Just text	Regular text
"Just text"	Text in quotes
"The date is "+{date}	Text plus variable
The date is {date}	Text plus variable
12	Numeric value
1900+{year}	Calculation

Text variables is combined with regular text by using the plus sign (+). The table below shows the operators that can be used for numeric values:

Operator	Description	Syntax
+	Addition	{year} + 1900
-	Subtraction	{dev} - 128
*	Multiplication	32 * {snt}
/	Division	{dev} / 16
&	And operation	{dev} & h'F
|	Or operation	{bm} | h'8000

The variable is created if it does not exist. It is permissible to use more than one operator in the same expression. The order of evaluation is from left to right, regardless of the type of operator, the rules of operator precedence is **not** followed. Use command `@CALC` to evaluate numeric expressions.

Example:

```
@SET {dev15} = {dev0} + 15
@SET {dev1} = 32*{snt} + 1
@SET {text} = "Variable {dev1} has value "+{dev1}
@SET {block} = ALL
@@ Assign values to array elements
@SET {dual[0]} = "some text"
@SET {dual[1]} = 14
@SET {index} = 0
@SET {list1[{index}]} = {index}
```

See also:
`@CALC`

@SETERROR [error]

This script command sets an error condition and pauses the transmission if the option "pause on error", in Traffic settings file, is set. If `[error]` is defined, it is shown in the message window. The error text may contain variables. This script command does **not** trigger any statement defined with `@ONERROR`.

> ⚠ When running script files in WinFIOL CLI it is not possible to resume a paused transmission.

Example:

```
SAAII:SAE=500,BLOCK=LI,NI=256;
@COPY {_line11} {fc} 21 2
@IF {fc} <> "" THEN SETERROR Size alteration error: fc={fc}
```

See also:
`@IFERROR`, `@ONERROR`

@SETPROMPT "<prompt>" [..]

This script command is used to temporarily override the default prompts for the current channel. It is useful in circumstances where the received data can be interpreted as a prompt. Unprintable characters are specified using escape codes in the format `"\nnn"` where nnn is the hexadecimal representation of the character. For example, `"\004"` represents the EOT character. See further in the section escape characters.

Use command @RESTOREPROMPT to restore the default prompts.

Examples:

```
Example 1
! Check only for prompts "WINFIOL>" and character 3 plus ">"!
@SETPROMPT "WINFIOL>" "\003>"
```

```
Example 2
! Dump the AP event log which may contain dummy prompts !
echo "Dumping event log file"
cd /d C:\
cd C:\temp
! Write the event data to the file log.txt !
dumplog application
! Change the prompt on the target system !
prompt WINFIOL$G
! Make WinFIOL check for only this prompt !
@SETPROMPT "WINFIOL>"
type log.txt
! Restore the default prompts !
@RESTOREPROMPT
```

```
! Restore the prompt on the target !
prompt $P$G
echo "--- Finished ---"
```

See also:
@RESTOREPROMPT, @WAITPROMPT

@SIZE {source} {elementcount}

This script command determines the number of elements in an array variable, see variables. The variable {source} must exist. The automatic variable {_LINES} may be used as source variable. The number of elements in {source} is stored as a numeric value in the variable {elementcount}.

Example:

```
CACLP;
@SIZE {_LINES} {linecount}
@COMMENT Printout contains {linecount} lines
```

See also:
@READ, @MERGE

@SORT {variable}

This script command sorts the lines found in {variable}.

Example:

```
@SET {var[0]} = "A"
@SET {var[1]} = "Z"
@SET {var[2]} = "B"
@SET {var[3]} = "S"
@SET {var[4]} = "H"
@SORT {var}
! Print the result !
@FOREACH {var} COMMENT {_CURRLINE}
## A
## B
## H
## S
## Z
```

See also:
@FIND, @GREP, @COUNT

@T <n>

This script command suspends the transmission until <n> seconds have passed. The maximum value for n is 32767.

See also:
@K, @M

@TRIM {variable}

Trims leading and trailing spaces and control characters from {variable}.

Example:

```
@SET {str} = " This is a string "
@TRIM {str}
@COMMENT Trimmed: '{str}'
## Trimmed: 'This is a string'
```

See also:
@TRIMCHAR, @AFTER, @BEFORE, @COPY

@TRIMCHAR {variable} <characters>

This script command trims the given characters from `{variable}`. Escape characters may be used to specify the characters to trim.

Example:

```
@SET {str} = " This is a string "
@TRIMCHAR {str} " isa"
@COMMENT Trimmed: '{str}'
## Trimmed: 'Thtrng'
```

See also:
`@TRIM`, `@AFTER`, `@BEFORE`, `@COPY`

@UNIQUE {variable}

This script command removes duplicate lines from `{variable}`.

Example:

```
@SET {var[0]} = "A"
@SET {var[1]} = "A"
@SET {var[2]} = "B"
@SET {var[3]} = "B"
@SET {var[4]} = "C"
@UNIQUE {var}
! Print the result !
@FOREACH {var} COMMENT {_CURRLINE}
## A
## B
## C
```

See also:
`@FIND`, `@GREP`, `@COUNT`

@UNSET {variable}

This script command undefines the `{variable}`.

The commands `@IFDEF` and `@IFNDEF` is used to check the existence of variables.

Example:

```
@SET {var} = "Hello"
@COMMENT Value: {var}
@UNSET {var}
@COMMENT Value: {var}
! Results in: !
## Value: Hello
## Cannot find variable VAR
```

See also:
`@SET`, `@IFDEF`, `@IFNDEF`

@UPCASE {variable}

This script command converts the string `{variable}` to uppercase.

Example:

```
@SET {var} = "Hello Over There"
@UPCASE {var}
```

```
! Print the result !
@COMMENT {var}
## HELLO OVER THERE
```

See also:
@LOWCASE

@VERBOSE

This script command used to display result messages in the output window. To reset use script command @QUIET.

The default setting is off.

Example:

```
@VERBOSE
@SET {src} = "Search target string"
@COPY {src} {dest} 8 6
@QUIET
```

See also:
@QUIET

@WAITFOR /<text>[,text][...]/

This script command pauses the transmission and resumes it when <text> is received. Separate multiple text strings with comma.

> ⚠ The parsing for the specified text is case sensitive.

```
Example (AXE, auto release and reconnect are off)
SAAII:SAE=500,BLOCK=CCD,NI=32;
@RELEASE
@WAITFOR /END/
@COPY {_line11} {fcode} 21 2
@IF {fcode} <> "" THEN GOTO error
@CONNECT
SAAEP:SAE=500,BLOCK=CCD;
@END
...
@LABEL error
...
```

See also:
@/text/

@WAITPROMPT ON | OFF

This script command controls how WinFIOL transmits script files. A script command is normally only sent to the NE after a prompt has been received. Using this script command it is possible to disable the check for a prompt and send the command immediately. This is useful in situations where a prompt may not always be present.

Example:

```
! Create a file using copy con !
@FORMAT {eof} "\01a\00d\00a"
del test.txt
@WAITPROMPT off
copy con test.txt
First line
Second line
@SENDHEX {eof}
@WAITPROMPT on
@COMMENT File contains:
type test.txt
```

See also:
@SETPROMPT, @RESTOREPROMPT

@WHILE <condition>

This script command repeats the commands between @WHILE and @ENDWHILE as long as <condition> is TRUE. The condition may be any numeric expression that can be evaluated to TRUE or FALSE. If <condition> is zero or less then zero, all commands until the @ENDWHILE command are skipped. While loops can be nested up to 10 levels deep.

Example:

```
@SET {a} = 5
@SET {b} = 0
@WHILE {a} > {b}
@T 10
PLLDP;
@INC {b}
@ENDWHILE
```

Other types of loops are possible to implement using the commands @LOOP, @ENDLOOP, @LABEL, @GOTO, @SET, @INC and @IF.

See also:
@ENDWHILE

@WRITE {source} <filename>|{variable}

This script command writes the data contained in variable {source} to a file. The <filename> may be given as a quoted string or as a {variable}. If the filename does not include a path, the directory defined by {_CMDDIR} is used. The data is stored with one array element per line in the file. If the source variable is empty, the created text file will be empty.

Example:

```
@SET {filename} = "/tmp/data/caclp.txt"
CACLP;
@WRITE {_LINES} {filename}
@SIZE {_LINES} {linecount}
@COMMENT Wrote {linecount} lines to file {filename}
```

See also:
@READ, @SIZE, @SENDFILE

@/[text][,text][...]/

This script command checks for [text] in received data. The transmission is paused until the specified text and a prompt is received. However, if the next line is a script command, transmission is resumed even if no prompt is received. If no text is specified, WinFIOL transmits the next line immediately. This is useful after a MML command that causes a release. All text is case sensitive.

If a line starts with a AXE or MD110 command, @/ may be specified on the same line.

In the following example for AXE, WinFIOL continue the transmission only if 212 is received.

Example:

```
SAOSP;@/212/
DPWSP;
In the following example for MD110, WinFIOL continue the transmission only if EXECUTED is
received.
PCASI:UNIT=RMPS12,CI=TEST;@/EXECUTED/
PCCOS:UNIT=RMPS12,CI=TEST;@/EXECUTED/
In the following example for AXE, WinFIOL continue the transmission immediately after the EXIT
command.
MCLOC:USR=SYSTEM,PSW=INIT;
EXIT;@//
@S
DPWSP;
```

Script commands grouped by function

Variable handling commands

Command name	Command description
@CLEAR	Clear all previously preserved variables
@DEC	Decrement variable
@DECIMAL	Convert number to decimal
@HEX	Convert number to hexadecimal
@INC	Increase variable
@PARAM	Variable from parent file
@PARAMASK	Variable from parent file, ask if non-existent
@PRESERVE	Preserve all variables
@SET	Assign a variable a value
@UNSET	Undefines (deletes) the variable

Array handling commands

Command name	Command description
@COMPACT	Remove empty lines
@COUNT	Count matching lines
@CUT	Cut, or extract, a column or range of characters
@FIND	Find matching lines, return the line index
@FOREACH	For each line, apply script command
@GREP	Find matching lines, return the matching lines
@MERGE	Merge lines by appending, subtraction, union, intersection, or difference
@PASTE	Paste, or append, a column to a variable
@SIZE	Get number of array elements
@SORT	Sort lines
@UNIQUE	Remove duplicated lines

String handling commands

Command name	Command description
@AFTER	Return string after sub-string
@BEFORE	Return string before sub-string
@CONCAT	Concatenate sub-strings
@COPY	Copy a sub-string
@FORMAT	Format a string
@INSERT	Insert a sub-string in a string
@ITEM	Extract the n:th string separated by delimiters
@LENGTH	Get string length
@LOWCASE	Convert string to lower case
@REPLACE	Replace a sub-string in a string
@RITEM	Extract the n:th string from the right separated by delimiters
@SCAN	Find a sub-string in a string
@TRIM	Delete white space from a string

@TRIMCHAR	Delete specific characters from a string
@UPCASE	Convert string to upper case

Numeric commands

Command name	Command description
@ABS	Returns the absolute value of a numeric value
@CALC	Calculates a numerical expression
@ROUND	Returns the integer part of a numeric value
@DEC	Decrement variable
@INC	Increase variable

Flow handling commands

Command name	Command description
@A	Auto confirm
@C	Conditional jump to label
@END	Stop the transmission
@ENDLOOP	End of loop
@ENDWHILE	End of while loop
@EXIT	Exit script or include file
@G	Send comments
@GOSUB	Go to subroutine
@GOTO	Go to label
@HALT	Stop the transmission
@IF .. IN .. THEN	Conditional statement
@IF .. MATCHES .. THEN	Conditional statement
@IF .. THEN	Conditional statement
@IFDEF .. THEN	Jump if variable exists
@IFERROR .. THEN	Jump if error
@IFNDEF .. THEN	Jump if variable does not exist
@LABEL	Label destination
@LOOP	Loop
@R	Pause on error
@RETURN	Return from subroutine
@STOP	Stop the transmission
@WAITFOR	Wait for specific text
@WAITPROMPT	Disables or enables waiting for prompt
@WHILE	While loop

File handling commands

Command name	Command description
@APPEND	Append text to a file
@CLOSE	Close log file

@ERASE	Erase
@GETFILE	Get file from URL to local host
@INCLUDE	Include file to transmit
@L	Log file
@LOG	Create, append, or close log file
@PUTFILE	Transfer file to URL on remote host
@SCRIPTLOG	Create, append, or close a script log file
@READ	Read data from file
@WRITE	Write data to file

User interface commands

Command name	Command description
@ASK	Ask user to assign a value to the variable
@BEEP	Make sound
@COMMENT	Show message in the output window
@CONFIRM	Ask for confirmation before continuing transmission
@FORM	Show form window
@SELECTCHAN	Display browse dialog to select a channel property file
@SELECTFILE	Display browse dialog to select files

Date/Time commands

Command name	Command description
@GETDATE	Store current date in variable
@GETDAY	Store current day of week in variable
@GETTIME	Store current time in variable
@K	Wait until HH:MM:SS
@M	Wait n minutes
@T	Wait n seconds

Communication commands

Command name	Command description
@BREAK	Send Break character
@CHANNEL	Switch to new channel
@CLONE	Clone active channel. It is recommended to use @SAVECHAN
@CLOSECHAN	Close specified channel
@CONNECT	Connect terminal
@ONERROR	Execute script command on error
@ONRECEIVE	Execute script command on reception of text
@OPENCHAN	Open and switch to new channel
@PAUSE	Pause the transmission
@PORTNO	Switch to new channel
@RELEASE	Release the terminal

@RESTOREPROMPT	Restore default prompts
@SAVECHAN	Save active channel
@SENDFILE	Send file
@SENDHEX	Send hexadecimal value
@SETERROR	Set error condition
@SETPROMPT	Set temporary prompt for the channel
@STOP	Stop the transmission
@WAITFOR	Wait for specific text
@WAITPROMPT	Disables or enables waiting for prompt

Trace commands

Command name	Command description
@VERBOSE	Show confirmation of script command
@QUIET	Disables @VERBOSE

Debug script commands

A number of script commands may be typed in the buffered input window.
These are commands that can print or change the value of variables or perform certain tasks.

Command name	Command description
@ASK	Ask user to assign a value to the variable
@BEEP	Beep
@BREAK	Break
@COMMENT	Show message in the output window
@CONNECT	Connect
@COPY	Copy a sub-string
@DEC	Decrement variable by n
@GETDATE	Store current date in variable
@GETDAY	Store current day of week in variable
@GETTIME	Store current time in variable
@INC	Increment variable by n
@INCLUDE	Include file to transmit
@LENGTH	Get string length
@ONERROR	Error handling
@ONRECEIVE	Check all received text
@RELEASE	Release
@SCAN	Scan local position of text in a variable
@SENDHEX	Send hexadecimal value immediately
@SET	Assign a value to a variable

Deprecated short-version script commands

The previously named "Internal Script Commands" are no longer separated in the WinFIOL application, and because of that the distinction between the types are not necessary. Some of these commands has functionality that are unique. A description of those can be found in the list with the regular script commands. The others are kept for backwards compatibility and are mentioned together with the regular script command that has the corresponding functionality. It is recommended to use the regular commands for better readability of scripts.

1.3. WinFIOL CLI

Short Description

This section describes the usage of WinFIOL CLI in ENM and WinFIOL CLI in the Scripting VM.

WinFIOL CLI in ENM

Application Concepts

WinFIOL CLI connects to Network Elements (NE) for fault handling, testing, configuration and maintenance. The tasks are handled manually, by entering commands in the terminal, or automatically, by using WinFIOL Scripts.

Permissions in ENM

To run WinFIOL, the user must be assigned to both WinFIOL_Operator and any POSIX Role (Scripting_Operator, Amos_Operator, or Element_Manager_Operator). If you cannot access WinFIOL, contact the Administrator.

> To connect to network elements via TLS, the user must have appropriate COM roles with the correct Target Group associated to its ENM user.

Start WinFIOL CLI in ENM

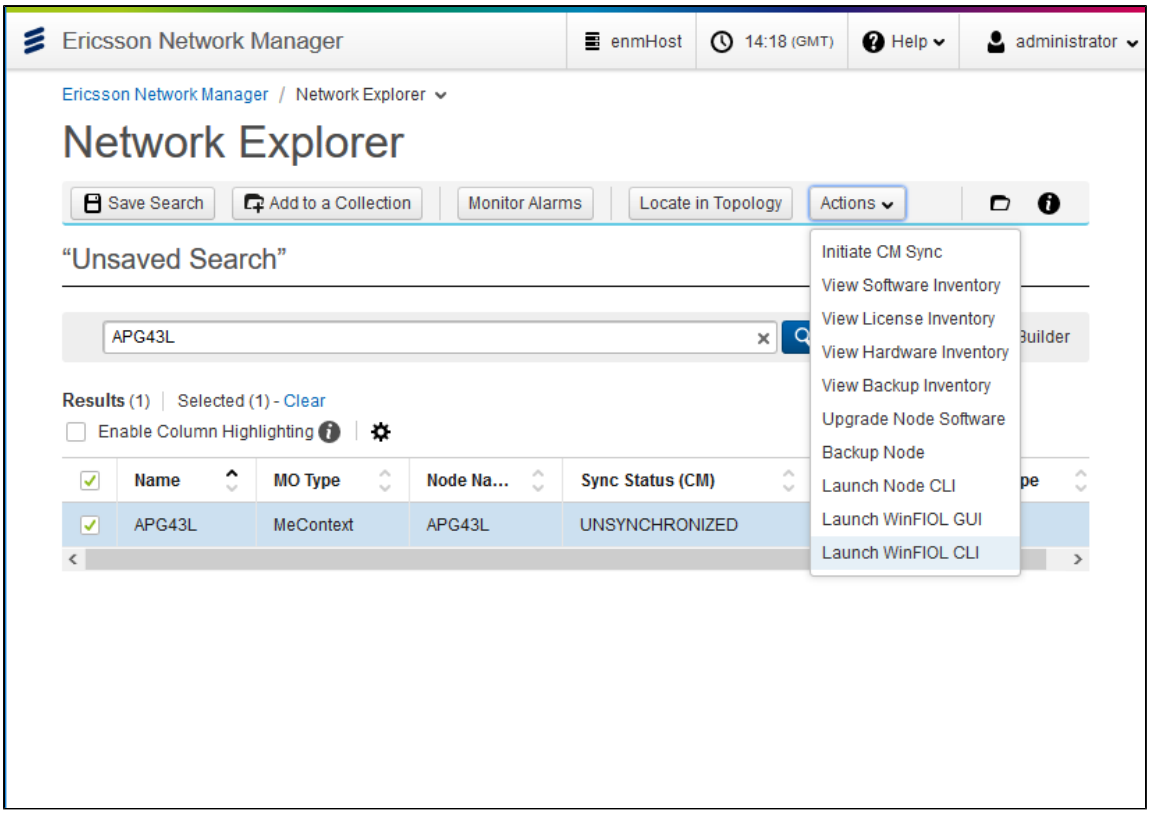
There are two ways for starting WinFIOL: directly from the Application Launcher, in the Tools box; or from the "Open With" menu in Network Explorer or Topology Browser.

When starting directly from the WinFIOL CLI application, this opens up the WinFIOL CLI prompt, where the user connects manually to an NE. See the Connecting to an NE for more information.

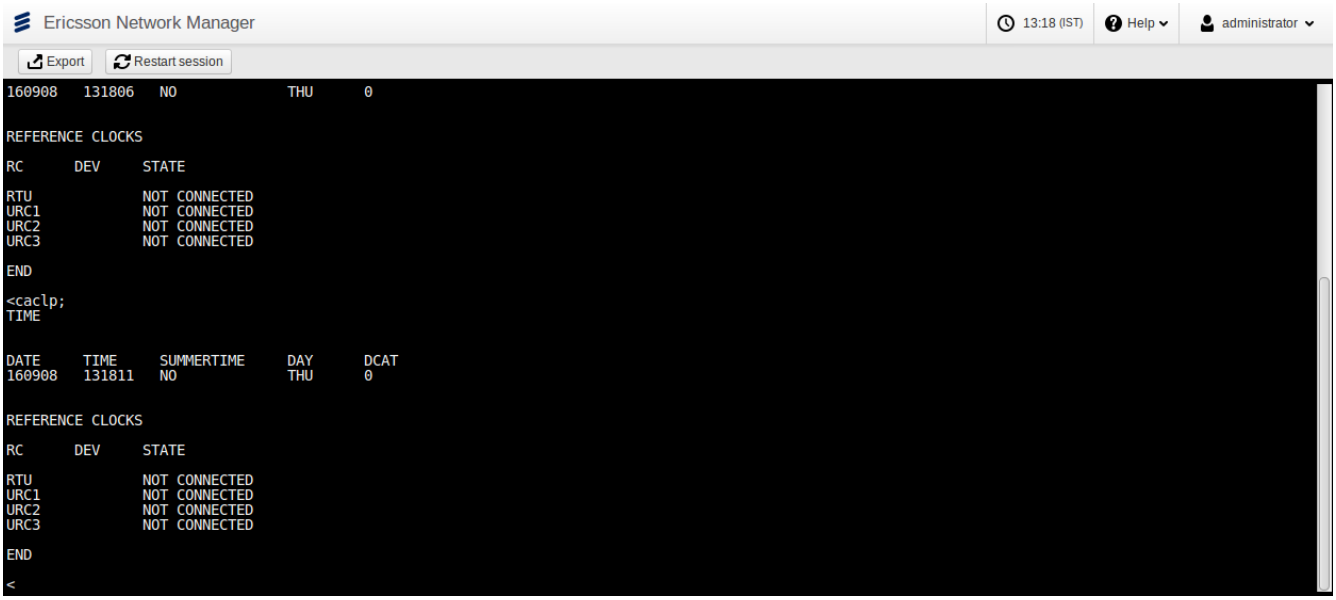
When launching with a Network Element selected, a login prompt towards the node is shown in the shell. Port 22 is always used for ssh connections.

To start WinFIOL from the menu in Network Explorer or Topology Browser, select an NE from the list, and click the "Launch WinFIOL CLI" button located on the action bar. If there are many actions available, some actions are gathered under the "Action" dropdown menu.

> Additional roles might be required to use Network Explorer and Topology Browser.



This brings up the shell terminal with the prompt of the selected NE.



User Interface

When the channel is open, all commands and key combinations are transmitted transparently to the selected NE. See Network Element documentation to learn more about the commands and key combinations supported by the NE.

WinFIOL functionality, such as script execution and logging, is accessed from the Channel Menu. When connected to an NE, press F8 to open the Channel Menu.

Type the command at the channel prompt and press ENTER to execute it. To close the menu, press the F8 again.

Command	Usage
script <scriptfile>	Opens and transmits the selected script file in the current channel. The menu is closed and output from the transmission shown in the terminal window. See more detailed information in the Running script files section.
props	Shows a selection of the connection and logging settings used for the current channel, and some of the settings related to script transmission. See more detailed information in the

	Channel Property Files and Traffic setup settings sections.
save <channel file>	Saves the current channel settings in a channel property file. See some more information regarding channel settings in the Channel Property Files section.
log enable <logfile.log>	Turns on the traffic log in the current channel. See a detailed description in the Logging section.
log disable	Turns off the traffic log in the current channel. Note that logging is automatically closed when the channel is disconnects.
log status	Shows the status, "`enabled`" or "`disabled`", of the different logging methods available in WinFIOL CLI, including the daily log, traffic log and logging of raw data.
close	Closes the channel and returns to WinFIOL CLI prompt if the channel was started from there, else both the channel **and** WinFIOL CLI are closed.
help	Shows the channel menu options

The default directories used by WinFIOL CLI in ENM, when not specifying an absolute path, are:

$HOME/winfiol/logfiles	When output logging is activated from the Channel menu or from inside of a script.
$HOME/winfiol/channelfiles	When saving Channel property files from the Channel menu.
$HOME/winfiol/commandfiles	When running script files from the Channel menu.

The directories require write permission.

WinFIOL CLI in Scripting VM

Application Concepts

WinFIOL CLI in Scripting VM can be started in interactive or non-interactive mode. See the differences below:

Interactive mode:

- WinFIOL CLI opens in the terminal window.
- It is started with or without application arguments
- Commands are entered from the keyboard.
- The output is shown in the terminal window.
- Access to the Channel Menu to turn on or turn off logging, open and close channels towards an NE, and to execute a script file.

Non-interactive

- WinFIOL CLI runs invisibly in the terminal window.
- It is created for scripting and scheduling.
- It is started with an NE and a command or a script file as arguments.
- No output is shown in the terminal window.
- Automatically exits when the script or command execution has finished or if an error occurs.

Permissions in ENM

The Scripting_Operator role is required to run WinFIOL in Scripting VM.

Application Arguments

When starting WinFIOL CLI using the application arguments, the default folder for Channel property files, log files and script files is the current directory.

The following table lists the available arguments:

Argument option	Description
-h [--help]	Displays this help.
-v [--version]	Displays version information.
--poid <poid name>	Instantly open a channel to the POID specified.
--neid <neid name>	Instantly open a channel to the NEID specified.
--host <ip-address>	Instantly open a channel to the NE using IP address.
--protocol <protocol name>	Select a protocol. Valid protocol values are ssh or tls. Must be combined with --host. If no value is informed, the selected protocol from the template channel property file will be used.

--port <port number>	The remote port number. The valid range for the port is [0, 65535]. Must be combined with --host. If no value is informed, the port from the template channel property file will be used.
--channelfile <channel property file path>	Instantly open a channel with the parameters from the selected channel property file.
--logfile <log file path>	Logs traffic into a logfile. If the file already exists, the data is appended by default. Must be combined with --host, --channelfile, --poid or --neid.
-o [--overwrite]	Overwrites an existing logfile. Must be combined with --logfile.
--scriptfile <script file>	Instantly start a WinFIOL script. Must be combined with --channelfile, --poid or --neid. WinFIOL will run in non-interactive mode and exit when the script is finished.
-1 [--command] <command>	Connect and execute one command. It must be enclosed in single quotes (') and combined with --channelfile, --poid or --neid. WinFIOL will run in non-interactive mode and exit when the command is executed.

⚠ '-1' (the short version of --command) is the number one, not the lower case L. This is a concession for backwards compatibility with CHA scripts.

Connecting to a Network Element (non-interactive mode)

The non-interactive mode is started with the arguments `--channelfile, --neid` or `--poid` in combinaton with `--command` (`-1`) or `--scriptfile`. The mode is somewhat invisible in the terminal, that is, no output from command execution or errors are shown, except the execution status "`Running...`" When the execution has finished, the status "`Finished!`" is printed in the terminal and WinFIOL CLI automatically exits.

```
user@machine:~$ wfcli --channelfile ~/winfiol/channelfiles/test_channel.xml -1 'mml ALLIP;'
Running...
Finished!
user@machine:~$
```

Connecting to a Network Element (interactive mode)

A channel is set up by starting WinFIOL CLI with the command `wfcli` and an argument like `poid, neid` or `channelfile`:

```
user@machine:~$ wfcli --poid 2044007

>mml
WO      GSM02BSC01/LTE/BSC/19-Q2- AD-2    TIME 200423 1631  PAGE    1
<exit;

>exit

## Connection to host lost
## Press any key to exit.
```

If WinFIOL CLI is started without any arguments, the `wfcli>` prompt appears. To show all available options at this prompt, type "`help`". Use the command `open` to establish a connection to an NE.

```
wfcli>help
Available commands:
open <hostname>[:<port>]                                    Open a channel to the <host> (IP or
hostname)
[-protocol <protocol> -l <logfile> [-o]]                                    If <port> not
supplied, the standard port for protocol will be used.
                                                            Port must be in range (0, 65535].
                                                            Valid <protocol> values are ssh or tls.
Default is ssh.
                                                            Traffic logs will be put into <logfile>.
                                                            '-o' is an optional flag to overwrite
log file (it is appended by default).
open <channel_file.xml> [-l <logfile> [-o]]                 Open a channel with the parameters from
the <channel_file>.
                                                            Traffic logs will be put into <logfile>.
                                                            '-o' is an optional flag to overwrite
```

```
log file (it is appended by default).
reopen                                                  Reopen the last channel. If '-o' was
used flag will be omitted and logging will be appended.
exit (or CTRL-C anytime)                                Quit WinFIOL CLI.
help                                                    Display this help.
```

```
wfcli>open 172.17.139.194 -protocol ssh
login name:ts_user1
ts_user1's password:***********
*******************************************************************************
IF YOU ARE NOT AN AUTHORIZED USER, PLEASE EXIT IMMEDIATELY.
*******************************************************************************

>mml
WO      BSC AP05 60C_RT_25_CM01   AD-482  TIME 200423 1634  PAGE    1
<caclp;
TIME

LOCAL TIME

DATE     TIME     SUMMERTIME     DAY      DCAT
200423   163458   NO             THU      0


REFERENCE CLOCKS

RC      DEV     STATE            URCTYPE  SERVICE   LOCATION
URC1            NOT CONNECTED
URC2            NOT CONNECTED
URC3            NOT CONNECTED

END
<
<exit;

>exit

## Connection to host lost
wfcli>
```

Channel Property Files

To save a channel as a Channel property file, press "F8" to access the channel menu and type save <filename>. To learn more about saving and reusing connection parameters, see the section Channel Property Files.

```
CHANNEL MENU - Options:
script <scriptfile>         Run script <script file>
props                       Show channel properties
save <channel file>         Save channel <channel file>
log enable <logfile.log>    Enable traffic logs to <logfile>
log disable                 Disable traffic logs
log status                  Show logging status
close                       Close channel
help                        Display this help
(Press F8 to go back)
channel>save test123.xml
## Channel file saved
channel>
```

```
user@machine:~$ ll ~/winfiol/channelfiles/
total 16
drwxr-xr-x 2 user user 4096 Mar  2 14:23 ./
drwxr-xr-x 5 user user 4096 Feb  3 10:25 ../
-rw-r--r-- 1 user user 1327 Mar  2 14:23 test123.xml
```

Logging methods

There are four different logging methods available in WinFIOL: Traffic log, Daily log, Script log, and Raw log. The traffic log can be activated from the Channel menu (F8) in the CLI. The command "`log enable`" defines a new traffic log file, and the command "`log disable`" closes an open traffic log file for the currently active channel. If an existing log file is defined, it is appended by default, that is, it keeps the previous data and writes new data to the end of the log file. The command "`log status`" shows if logging is enabled in the channel and the name of the file. See the section Logging for a description of the different log methods, and how they are used.

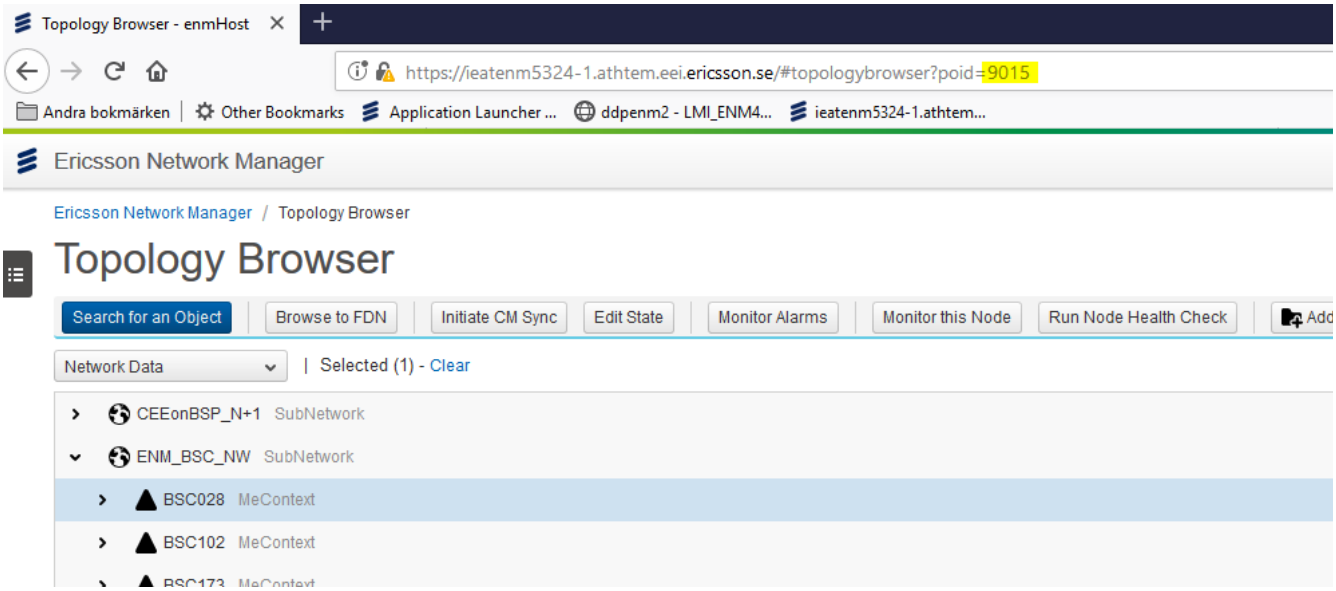
Traffic log files can also be enabled by using the non-interactive startup arguments.

Running script files

Run the script files from the Channel menu or by using the non-interactive startup arguments. For a more detailed description of the available script commands in WinFIOL, see the section WinFIOL Script Commands.

How to determine POID and NEID

The POID is found in the address bar in the browser when selecting a node in Topology Browser or Network Explorer. The NEID is the actual name of the node, in the example below BSC028.



1.4. Logging Methods

All text received from the NEs and shown in the output window can be logged in a channel-associated log file. Each channel can have its own log file, and a log file cannot be shared among two or more channels simultaneously. There are four different logging methods available in WinFIOL: Traffic log, Daily log, Script log and Raw log. One channel cannot have more than one of each log methods active simultaneously.

Traffic log

The traffic log method is activated in the channel menu, and the created log files are saved by default to `$HOME/winfiol/logfiles`, but an absolute path can also be used.

Refer to section WinFIOL CLI and the WinFIOL GUI online help to learn more about how to enable the traffic log.

Daily log

The daily log method is activated in a Channel property file. In the the property file, you specify the path to your preferred log file directory, but the file name is created automatically every time a channel is opened. The file name is in the format: *YYYYMMDD#number_channelFileName.log. A* new file is automatically created every day at midnight if the channel is open.

The predefined path in the Channel property file is `$HOME/winfiol/logfiles`.

Script log

It is possible to turn on the script log method from inside of a script file, using one of the three different script commands: `@LOG`, `@SCRIPTLOG` and `@L`. The default directory for these logfiles is `$HOME/winfiol/logfiles`, but a path can also be used in each command. Refer to the full description and syntax for the respective command in section Script Commands.

Logging of raw received data

Raw logs allow to look at the raw data from an NE, and they are activated in a Channel property file. There you may specify the path to your preferred log file directory, but the file name is created automatically every time a channel is opened. The file name is in the format: "*YYYYMMDD_HHMMSS_rawlog_chn<number>.log".*

All data that goes through the communication driver of WinFIOL are written into this log, so an Hex Editor or reader is needed to interpret escape and other unprintable ASCII chars. The direction of data is represented by the 0x18 and 0x19 chars, that are appended as prefix to each message in the log by WinFIOL. They represent sending and receiving, respectively.

The predefined path in the Channel property file is `$HOME/winfiol/logfiles/rawlogs.`

Commands and SYSLOG

All commands from TTY mode, buffered mode and script are logged to syslog.

1.5. Channel Property Files

It is possible to save a channel into a Channel property file, XML file, in the channel menu. Each Channel property file contains connection related settings that are read by WinFIOL during the opening of a channel. One file can be used by any number of channels simultaneously. The default location for opening and saving the files, if a path is not supplied, is `$HOME/winfiol/channelfiles`.

It is possible to manually edit a Channel Property file in a text editor for further customization.

Template Channel Property File

The file "template_channel_properties.xml" contains the default parameters for new channels. When opening a channel using anything other than a saved Channel property file, the remainder of the required connection data will be taken from the template. If this file is missing in $HOME/winfiol/channelfiles, it is copied there from the WinFIOL installation folder.

By editing "template_channel_properties.xml" file a user can change the default authentication method (password) used for SSH, and other default values for connections.

```
<?xml version="1.0" encoding="utf-8"?>
<Channel>
  <ChannelFileVersionNumber>1.0</ChannelFileVersionNumber>
  <ProtocolSettings>
    <SelectedProtocol>ssh</SelectedProtocol>
    <InitialMode>tty</InitialMode> <!-- tty/buffered -->
    <HostName></HostName> <!--127.0.0.1, example.com, ::1 -->
    <Protocol name="ssh">
      <Port>22</Port>
      <Authentication type="pw"> <!-- Password(pw), Public key(pk), Keyboard Interactive
Authentication(kia), pw+pk, pk+pw -->
        <Key>~/.ssh/id_rsa</Key> <!-- Use absolute path or '~/' for HOME -->
      </Authentication>
    </Protocol>
    <Protocol name="tls">
      <Port>9830</Port>
      <Authentication type="pw"> <!-- Not applicable -->
        <Key>~/Ericsson/OMSec/CAcert/pKey.key</Key> <!-- Use absolute path or '~/' for HOME -->
        <CaChainCert>~/Ericsson/OMSec/CAcert/RootCA.pem</CaChainCert>
        <ClientCert>~/Ericsson/OMSec/CAcert/NECertCA.pem</ClientCert>
      </Authentication>
    </Protocol>
  </ProtocolSettings>
  <TargetSettingsFile>etc/apg4x_target.xml</TargetSettingsFile>
  <TrafficSetupSettingsFile>etc/default_traffic_setup.xml</TrafficSetupSettingsFile>
  <Logging>
    <DailyLog>false</DailyLog>
    <DailyLogFile>~/winfiol/logfiles</DailyLogFile> <!-- Use absolute path or '~/' for HOME-->
    <RawLog>false</RawLog>
    <RawLogDir>~/winfiol/logfiles/rawlogs</RawLogDir> <!-- Use absolute path or '~/' for HOME-->
  </Logging>
  <Alex>
    <AlexUrl></AlexUrl> <!-- http://example.com/alex -->
```



```
    <LibraryName></LibraryName> <!-- li=EN/ABC123456789R1A, id=12345 -->
  </Alex>
  <AppearanceSettings>
    <MaxInputLines>100</MaxInputLines> <!-- Maximum lines in buffer mode: 10 to 200 -->
    <MaxOutputLines>5000</MaxOutputLines> <!-- Maximum lines in output window: 200 to 100000 -->
  </AppearanceSettings>
  <IacSettings>
    <UseIacOptions>false</UseIacOptions>
  </IacSettings>
  <DebugSettings>
    <ShowIAC>false</ShowIAC>
  </DebugSettings>
</Channel>
```

SSH Authentication

SSH in WinFIOL allows different authentication methods: password, public key and keyboard interactive. The password and public key methods can be used separately or in combination. In order to use public key authentication, your public key has to be available on the remote server.

Use the Channel property file to change the SSH Authentication options.

Value	Description
pw	The Password option defines password authentication for the session. This option also handles password change in a special way for the target type APG 43L (AXE). This is the default option.
pk	The Public Key option defines public key authentication for the session.
pw+pk	The Password and the Public Key option defines password and public key authentication for the session. Password authentication will be tried as the preferred authentication. If the password authentication fails, public key authentication will be used. The option also applies to SSH servers that require both authentication methods.
pk+pw	The Public Key and Password option defines public key and password authentication for the session. Public key authentication will be tried as the preferred authentication. If the public key authentication fails, password authentication will be used. The option also applies to SSH servers that require both authentication methods.
kia	The Keyboard Interactive option defines the Keyboard-Interactive authentication method for the session. It uses a challenge-response mechanism that requires the user's input to answer to server requests. If the SSH server does not have this method enabled, the authentication will fail.

TLS Authentication

TLS in WinFIOL allows authentication method: password.

Value	Description
pw	The Password option defines password authentication for the session.