ERICSSON

OPS-User Guide

User Guide

1/1553-CNA 403 3474 Uen J

Contents

1 About This Document

This chapter contains the following:

- Purpose
- Target groups
- Prerequisites
- Typographic conventions

Purpose

This document describes how to use Operations Procedure Support (OPS). OPS is an editor, debugger, and execution environment for command files using the OPS Script language.

Target Groups

This document is intended for operators and application developers who use the OPS Script language and the Command File Developer to create OPS Scripts for exchange management.

Prerequisites

It is assumed that you are familiar with the following:

- Workstations
- Window-based computer interfaces
- Maintenance of NEs connected to the ENM
- ENM
- Tasks that are performed by the command files

Typographic Conventions

The OPS script language described in this document uses metalanguage. The following notation is used: [xxx\] The parameter between the square brackets is optional. The square brackets are not typed in the command file statement. {xxx}. The parameter between the braces is optional, but can be repeated as often as required by the statement. The braces are not typed in the command file statement.

2 Launch OPS Native User Interface (NUI)

1. Go to ENM Launcher and launch Shell Terminal on Scripting.
2. SSH to OPS VM. (ssh -X <user>@<opsvm>)
3. Go to path /opt/ops/ops_client/bin/ops_nui.

   The execution of the command without any parameters produces only help information.

   OPS NUI can be used to execute command files.

The parameters controlling the way OPS is started are described in OPS- System Administrator Guide, 1/1543-CNA 403 3474.

The maximum number of parallel running sessions is limited to 50 MAX GUI sessions.

The Network Element drop-down in the OPS GUI displays all the supported nodes. The time taken to display all the nodes depends on the number of Network Elements configured in ENM.

Note: If the Network Element drop-down takes more than a minute to display all the nodes, possibly one or more nodes are incorrectly added in ENM. Recheck the connectivity information of the Network Elements.

The correct ports to be used for configuring the Network Elements are described in ENM NIG Connection Matrix, 1/102 72-AOM 901 151-1.

2.1 OPS Script Language

The OPS script language is a programming language that allows intelligent command files to be created. Command files manage the external system types supported by the ENM system. The language allows variables, expressions, conditional jumps, subroutines, and arithmetic operations.

Command Files that include elements of the OPS script language are referred to as OPS Scripts.

2.2 Program Source Structure

OPS scripts are written as text files. An OPS script can contain Man Machine Language (MML) commands, OPS script commands and functions. Each command (with the exception of comments) starts on a new line.

The first non-blank character on a line determines if a command is an MML command or an OPS script command.

OPS script commands start with the (@) character, the (/) character, or the (!) character. Otherwise the line (if not empty) is treated as an MML command.

An OPS script command can span line boundaries, provided that the last non-blank character on each incomplete line is a comma (,). This does not apply in a quoted string:

@COMMENT("The external system name is ", es_name); @@ This is ok

@COMMENT("Fred, my boss is managing this project"); @@ Syntax Error

Example 1 OPS Script Command Spanning Line Boundary

MML command can span line boundaries provided that the last non-blank character on each incomplete line is either an () character or a (,) character.

Characters in command files are case insensitive, except characters in quoted strings. At execution, MML commands are converted to upper case, if needed, before being sent to the external system.

Execution begins from a single file at a particular line. Commands are executed one after another. Another command file can be included in the execution and executed as part of the original command file through the @INCLUDE command. When the execution of an included file is completed, the execution resumes in the original file at the line following @INCLUDE. If @INCLUDE is the last line in the original file, the execution of the original file is also considered as completed.

An included file can include another file. No cyclic inclusion is allowed.

2.2.1 Parsing and Execution

After variable substitution, the command file interpreter analyzes and executes an OPS script command or function. An MML command is only transmitted to the external system. The command file interpreter has three normal states: Running The execution is proceeding. Input Pending A user input is pending as a result of commands such as @INPUT or @PROMPT. Stopped The execution has been stopped.

Note: Run-time information is retained in this state.

3 Lexical Elements

This section describes the following:

1.OPS script commands and functions

2. MML commands

3. Remarks

4. Command and function names

5. Variables

6. Variables substitution

7. Simulated arrays

8. String constants

9. Character constants

10. Numeric constants

11. Labels

12. Expressions

13. Operators

3.1 OPS Script Commands and Functions

OPS script commands and functions are used to build OPS scripts that perform certain tasks within management of external systems. The command files can be used to send MML commands to the external systems.

An OPS script that starts on a new line must start with the @ character (the only exception is the !$$$$! command), for example, @CONNECT. OPS script functions are used within OPS commands. These are not preceded by @ characters, but they end with parentheses. An example of an OPS script function is GETDEST().

The commands and functions can be grouped into the following categories, based on what they do:

— Error handling

— Flow control

— Child script handling

— External system communication

— File and printer logging

— Time event

— User input and output

— File system

— String and number processing

— Progress reporting

— Miscellaneous

— FIOL commands

The FIOL commands have different purposes in a command file. These are grouped because they make up a separate language. The FIOL commands are in the OPS script language to guarantee backward compatibility. The FIOL commands can be used in the same command files as the OPS script commands and functions. In this section, the commands and functions in each of the groups are listed, together with short descriptions.

Error Handling

The following are error handling commands and functions:

@CHECK Enables or disables error checking

@IFERROR THEN Tests if the error flag has been set

@SETERRORFLAG Sets the error flag

Flow Control

The following are flow control commands and functions:

!$$$$! Marks the end of an included file

@BREAK Interrupts an ongoing @FOR-TO loop

@ELSE Executed if the condition made by @IF-THEN is not fulfilled

@ENDIF Indicates the end of the last alternative in the @IF-THEN and @IFERROR THEN commands

@GOTO Performs a jump to a label in the current command file

@GOSUB Calls a subroutine within the current command file

@GOSUB command that called the subroutine @FOR-TO Marks the beginning of a for-loop structure @NEXT Marks the end of a for-loop structure

@IF-THEN Enables conditional execution

@INCLUDE Includes another command file, in which the execution continues

@LABEL Specifies a location to which an @GOTO or an @GOSUB command refers

LABELEXIST() Checks whether the indicated label exists in the current OPS script

@ONDISCONNECT Enables continued script execution at the indicated label if the NE connection is disrupted

@ONTIMEOUT Indicates where to continue script execution after time-out while waiting for an input

@RETURN Returns from a subroutine to the command after the

@RESET All run-time data except the subroutine stack is reset to default values.

@RESTART Performs a jump to the beginning of the current command file

@STOP Stops the execution of a command file

@SWITCH-CASE-DEFAULT Enables conditional execution

External System Communication

The following are external system communication commands and functions:

@CONNECT Establishes a connection to an external system

@DISCONNECT Breaks the connection to an external system

@FUNBUSY Specifies the total number of attempts that will be made to execute an MML command when FUNCTION BUSY or FUNCTION BUSY - SESSION LOCKED is encountered

@LOGOF Makes it possible to display delayed responses when @ORDERED("OFF") has been executed

@LOGON Suspends the effect of @LOGOF

@MARK Shifts the mark in the buffer

@ORDERED Used to control the receiving of delayed responses

@REPORT Used to control the receiving of sequences of delayed responses from external systems of the type System 12

@SPONTREP Sets up a subscription to spontaneous reports from an external system.

@SPONTREPOFF Terminates spontaneous report subscriptions that have been set up with the @SPONTREP command

@SEND Used to send arbitrary characters to an external system

@SETREPLY Makes it possible to override the normal sets of accepted and not accepted responses

@WAITFOR Waits for one or two specified text strings in a response

@WAITREPLY Waits for a response

GETDEST() Returns the string used when setting up the connection

GETIOTYPE() Returns the IO node type of an external system

GETMODE() Returns the type of the currently connected external system ("AXE")

REPLY() Returns all text on a specified response line

REPLYLEN() Determines the number of lines that have been displayed since the last MML command or @MARK command was executed

SCAN() Scans the response buffer, since the latest MMLcommand or @MARK command, for a specified string

GETPROTOCOLTYPE() Returns the protocol type used for communication with the external system.

GETMARK() Retrieves the line number being pointed at by the invisible buffer mark within the Command File Output Area

GETEXCHHDR Returns an empty string as the exchange header of the connected exchange is not available for OPS.

File And Printer Logging

The following are file and printer logging commands and functions:

@LOG Specifies a file on which communication is logged.

@CLOSE Closes a log file that is currently open.

GETLOG() Returns the name of a log file, if a log file is open.

Time Event

The following are time event commands and functions:

ADVANCE() Allows a specified date and time to be increased by a specified amount

REGRESS() Allows a specified date and time specified by Exp1 to be decreased by a specified amount

User Input and Output

The following are user input and output commands:

@CLRSCR Clears the Input Output window and the Comment window

@COMMENT Displays a specified text string in the Command File Output area or the Comment window

@DRAW Allows elaborate output to the Input Output window

@FORM Makes it possible to request user input in the Input Output window

@INKEY Opens a pop-up window, which asks the user to enter a single-character value

@INPUT Opens a pop-up window, which asks the user to enter a text string

@MENU Used together with the @DRAW command to enable menu selection

@PROMPT Displays a pop-up window with the choices "Continue" , "Stop" , and "Stop and Quit"

@WINDEND Closes the Comment window @WINDOW Opens and clears the Comment window

File System

The following are file system commands and functions:

@CHDIR Changes the current directory to a specified directory

@ERASE Erases a specified file

@MKDIR Creates a new directory

@OPENREAD Opens the specified file for reading

@READ Reads a line from the file that has been opened by the

@OPENREAD command

@RENAME Renames an existing file, or moves and renames the file to another directory

@VIEWFILE Displays a specified file

@WRITE Writes to a specified file the values of certain specified parameters as a line

@RMDIR Deletes the specified directory

CENTRALDIR() Returns the value of the OPS server environment variable OPS_CENTRALDIR and has the path to the central directory, which is defined in the parameter database

DISKFREE() Returns the amount of free disk space

GETDIR() Retrieves the path of the current directory

GETFILELENGTH() Returns the number of lines in the specified file

SAVEDIR() Returns the value of the OPS_SAVEDIR OPS server environment variable and has a default and recommended value /cha/response under every user's home directory

PRIVATEDIR() Returns the value of the OPS server environment variable OPS_PRIVATEDIR indicating the path and has a default and recommended value /cha/cmdfile under every user's home directory

SELFILE() Displays a file browser and prompts the user to enter a file name

String and Number Processing

The following are string and number processing functions:

CONCAT() Concatenates (joins in one string) specified parameters

COPY() Copies a specified part of a text string

DECIMAL() Returns the decimal value of a specified hexadecimal value

HEX() Returns the hexadecimal value of a specified decimal value

LENGTH() Determines the length of a specified text string

LOWCASE() Returns a specified text string in lower case

PAD() Produces a padded text string with a selected adjustment and a fixed length for use in files or in windows to get a nice layout

POS() Returns the start position of one specified string within another specified string

STYLE() Controls the text style (plain, bold, underline, italic) in pop-up and input/output windows (invoked by the @COMMENT, @DRAW, @INPUT, and @INKEY commands)

TRIM() Removes leading and trailing spaces, tab characters, and new line characters from a specified string

UPCASE() Returns a specified text string in upper case

Progress Reporting

The following are commands and functions for progress reporting to other processes invoking the execution of OPS scripts:

@SETPHASE Sets a value to an internal system variable indicating the current status of a script execution

@SETPROGRESS Sets the progress counter to the value indicated by Exp

@SETTOTALCOUNT Sets the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress

@STEPPROGRESS Steps the script execution progress counter a number of steps indicated by Exp

GETPHASE() Retrieves the current progress status set by the @SETPHASE command

GETPROGRESS() Retrieves the value of the script execution progress counter set by the commands @SETPROGRESS and @STEPPROGRESS

GETTOTALCOUNT() Retrieves the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress

Miscellaneous

The following commands and functions have miscellaneous purposes:

@DEC Decreases the value of a specified variable by 1

@DELAY Pauses the command file execution for a specified number of minutes

@DELAYSEC Pauses the command file execution for a specified number of seconds

@EXECUTE Executes a UNIX command and then returns to the command file

@FLUSHVAR Removes a specified variable

@INC Increases the value of a specified variable by 1

@MAIL Sends a mail with the body provided in the command to the specified email id (e.g. user@lmera.ericsson.se)

@SET Sets a variable value

GETDATE() Returns the system date, time, and day of the week

GETSESSIONID() Returns the current OPS session number

VAREXIST() Allows a command file to determine whether a certain variable exists

VERSION() Allows a command file to determine which version of the command file interpreter that is currently used

FIOL Commands

The following are FIOL commands:

/*...*/ Displays all characters between /* and */ in the Command File Output area or theComment window (see also @COMMENT)

@A Turns automatic confirmation of MML commands on or off

@C Closes the log file (see also @CLOSE)

@E Makes it possible to display delayed responses(see also @LOGOF)

@G Retained for compatibility reasons only, and does not have any effect

@I Includes another command file and executes this file (see also @INCLUDE)

@L Specifies a file on which communication is logged (see also @LOG)

@/W1,W2,.../ Checks that the next immediate response from the external system contains the specified character sequences

@R If the command @R+ has been executed before the @/W1,W2.../ command, the execution continues even if the text string specified by @/W1,W2.../ does not match the printout

@S Suspends the effect of a previously executed @E or @LOGOF command (see also @LOGON)

@T Pauses the command file execution for a specified number of seconds (see also @DELAYSEC)

@W Displays a pop-up window with the choices Continue , Stop , and Stop and Quit (see also @PROMPT)

@Z Turns the automatic response check on

3.1.1 Error Handling

This section describes how errors checks can be performed.

As part of the run time information, there is a global error flag. At the beginning of execution, the flag is not set. For MML commands and many OPS script commands, the error flag is set if the command fails. A command file can check if an error has occurred for a specific command by using @IFERROR THEN in conjunction with @CHECK("OFF").

The following example shows how these commands can be used, when the @CHECK("OFF") command is executed and the error checking is turned off. This means that the execution continues even if a command fails. The @IFERROR THEN command tests if the error flag is set.

Example 1 Use of Commands @IFERROR THEN and @CHECK ("OFF")

```
!No connection yet @CHECK("OFF") ALACP;
@IFERROR THEN
@COMMENT("Can not send ALACP") @ELSE
@COMMENT ("ALACP has been sent") @ENDIF
@CHECK("ON")
```

The effect on the error flag by individual commands is documented in the corresponding command descriptions.

3.1.2 Flow Control

The following are flow control commands and functions:

!$$$$! Marks the end of an included file

@BREAK Interrupts an ongoing @FOR-TO loop

@ELSE Executed if the condition made by @IF-THEN is not fulfilled

@ENDIF Indicates the end of the last alternative in the @IF-THEN and @IFERROR THEN commands

@GOTO Performs a jump to a label in the current command file

@GOSUB Calls a subroutine within the current command file

@GOSUB command that called the subroutine @FOR-TO Marks the beginning of a for-loop structure @NEXT Marks the end of a for-loop structure

@IF-THEN Enables conditional execution

@INCLUDE Includes another command file, in which the execution continues

@LABEL Specifies a location to which an @GOTO or an @GOSUB command refers

LABELEXIST() Checks whether the indicated label exists in the current OPS script

@ONDISCONNECT Enables continued script execution at the indicated label if the NE connection is disrupted

@ONTIMEOUT Indicates where to continue script execution after time-out while waiting for an input

@RETURN Returns from a subroutine to the command after the

@RESET All run-time data except the subroutine stack is reset to default values.

@RESTART Performs a jump to the beginning of the current command file

@STOP Stops the execution of a command file

@SWITCH-CASE-DEFAULT Enables conditional execution

3.1.3 External System Communication

The following are external system communication commands and functions:

@CONNECT Establishes a connection to an external system

@DISCONNECT Breaks the connection to an external system

@FUNBUSY Specifies the total number of attempts that will be made to execute an MML command when FUNCTION BUSY or FUNCTION BUSY - SESSION LOCKED is encountered

@LOGOF Makes it possible to display delayed responses when @ORDERED("OFF") has been executed

@LOGON Suspends the effect of @LOGOF

@MARK Shifts the mark in the buffer

@ORDERED Used to control the receiving of delayed responses

@REPORT Used to control the receiving of sequences of delayed responses from external systems of the type System 12

@SPONTREP Sets up a subscription to spontaneous reports from an external system.

@SPONTREPOFF Terminates spontaneous report subscriptions that have been set up with the @SPONTREP command

@SEND Used to send arbitrary characters to an external system

@SETREPLY Makes it possible to override the normal sets of accepted and not accepted responses

@WAITFOR Waits for one or two specified text strings in a response

@WAITREPLY Waits for a response

GETDEST() Returns the string used when setting up the connection

GETIOTYPE() Returns the IO node type of an external system

GETMODE() Returns the type of the currently connected external system ("AXE")

REPLY() Returns all text on a specified response line

REPLYLEN() Determines the number of lines that have been displayed since the last MML command or @MARK command was executed

SCAN() Scans the response buffer, since the latest MMLcommand or @MARK command, for a specified string

GETPROTOCOLTYPE() Returns the protocol type used for communication with the external system.

GETMARK() Retrieves the line number being pointed at by the invisible buffer mark within the Command File Output Area

GETEXCHHDR Returns an empty string as the exchange header of the connected exchange is not available for OPS.

3.1.4 File And Printer Logging

The following are file and printer logging commands and functions:

@LOG Specifies a file on which communication is logged.

@CLOSE Closes a log file that is currently open.

GETLOG() Returns the name of a log file, if a log file is open.

3.1.5 Time Event

The following are time event commands and functions:

ADVANCE() Allows a specified date and time to be increased by a specified amount

REGRESS() Allows a specified date and time specified by Exp1 to be decreased by a specified amount

3.1.6 User Input and Output

The following are user input and output commands:

@CLRSCR Clears the Input Output window and the Comment window

@COMMENT Displays a specified text string in the Command File Output area or the Comment window

@DRAW Allows elaborate output to the Input Output window

@FORM Makes it possible to request user input in the Input Output window

@INKEY Opens a pop-up window, which asks the user to enter a single-character value

@INPUT Opens a pop-up window, which asks the user to enter a text string

@MENU Used together with the @DRAW command to enable menu selection

@PROMPT Displays a pop-up window with the choices "Continue" , "Stop" , and "Stop and Quit"

@WINDEND Closes the Comment window @WINDOW Opens and clears the Comment window

3.1.7 File System

The following are file system commands and functions:

@CHDIR Changes the current directory to a specified directory

@ERASE Erases a specified file

@MKDIR Creates a new directory

@OPENREAD Opens the specified file for reading

@READ Reads a line from the file that has been opened by the

@OPENREAD command

@RENAME Renames an existing file, or moves and renames the file to another directory

@VIEWFILE Displays a specified file

@WRITE Writes to a specified file the values of certain specified parameters as a line

@RMDIR Deletes the specified directory

CENTRALDIR() Returns the value of the OPS server environment variable OPS_CENTRALDIR and has the path to the central directory, which is defined in the parameter database

DISKFREE() Returns the amount of free disk space

GETDIR() Retrieves the path of the current directory

GETFILELENGTH() Returns the number of lines in the specified file

SAVEDIR() Returns the value of the OPS_SAVEDIR OPS server environment variable and has a default and recommended value /cha/response under every user's home directory

PRIVATEDIR() Returns the value of the OPS server environment variable OPS_PRIVATEDIR indicating the path and has a default and recommended value /cha/cmdfile under every user's home directory

SELFILE() Displays a file browser and prompts the user to enter a file name

3.1.8 String and Numbering Processing

The following are string and number processing functions:

- CONCAT() Concatenates (joins in one string) specified parameters
- COPY() Copies a specified part of a text string
- DECIMAL() Returns the decimal value of a specified hexadecimal value
- HEX() Returns the hexadecimal value of a specified decimal value
- LENGTH() Determines the length of a specified text string
- LOWCASE() Returns a specified text string in lower case
- PAD() Produces a padded text string with a selected adjustment and a fixed length for use in files or in windows to get a nice layout
- POS() Returns the start position of one specified string within another specified string
- STYLE() Controls the text style (plain, bold, underline, italic) in pop-up and input/output windows (invoked by the @COMMENT, @DRAW, @INPUT, and @INKEY commands)
- TRIM() Removes leading and trailing spaces, tab characters, and new line characters from a specified string
- UPCASE() Returns a specified text string in upper case

3.1.9 Progress Reporting

The following are commands and functions for progress reporting to other processes invoking the execution of OPS scripts:

— @SETPHASE Sets a value to an internal system variable indicating the current status of a script execution

— @SETPROGRESS Sets the progress counter to the value indicated by Exp

— @SETTOTALCOUNT Sets the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress

— @STEPPROGRESS Steps the script execution progress counter a number of steps indicated by Exp

— GETPHASE() Retrieves the current progress status set by the @SETPHASE command

— GETPROGRESS() Retrieves the value of the script execution progress counter set by the commands @SETPROGRESS and @STEPPROGRESS

— GETTOTALCOUNT() Retrieves the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress

3.1.10 Miscellaneous

The following commands and functions have miscellaneous purposes:

— @DEC Decreases the value of a specified variable by 1

— @DELAY Pauses the command file execution for a specified number of minutes

— @DELAYSEC Pauses the command file execution for a specified number of seconds

— @EXECUTE Executes a UNIX command and then returns to the command file

— @FLUSHVAR Removes a specified variable

— @INC Increases the value of a specified variable by 1

— @MAIL Sends a mail with the body provided in the command to the specified email id (e.g. user@lmera.ericsson.se)

— @SET Sets a variable value

— GETDATE() Returns the system date, time, and day of the week

- GETSESSIONID() Returns the current OPS session number
- VAREXIST() Allows a command file to determine whether a certain variable exists
- VERSION() Allows a command file to determine which version of the command file interpreter that is currently used

3.1.11 FIOL Commands

The following are FIOL commands:

/*...*/ Displays all characters between /* and */ in the Command File Output area or theComment window (see also @COMMENT)

- @A Turns automatic confirmation of MML commands on or off
- @C Closes the log file (see also @CLOSE)
- @E Makes it possible to display delayed responses(see also @LOGOF)
- @G Retained for compatibility reasons only, and does not have any effect
- @I Includes another command file and executes this file (see also @INCLUDE)
- @L Specifies a file on which communication is logged (see also @LOG)
- @/W1,W2,.../ Checks that the next immediate response from the external system contains the specified character sequences
- @R If the command @R+ has been executed before the @/W1,W2.../ command, the execution continues even if the text string specified by @/ W1,W2.../ does not match the printout
- @S Suspends the effect of a previously executed @E or @LOGOF command (see also @LOGON)
- @T Pauses the command file execution for a specified number of seconds (see also @DELAYSEC)
- @W Displays a pop-up window with the choices Continue , Stop , and Stop and Quit (see also @PROMPT)
- @Z Turns the automatic response check on

3.1.12 MML Commands

If the first non-blank character in an OPS script is not (@), ?? or ??, and the line is not empty, the line is considered as an MML command. Exception: see command line spanning in Labels Syntax on page 147.

MML commands only analyzed in variable substitution. The MML commands are sent to and interpreted by the external system.

3.1.13 Remarks

Remarks can be included in command files through one of the @@ notation or the ! notation. Remarks are displayed on the screen.

The @@ notation and the ! notation are used to indicate that the rest of the line is a remark. (The only exception is the !$$$$! command.) The remark can start anywhere on the line.

The ! notation can also be used on the same line as an MML command. The remark can be written between two exclamation marks and appended after the semicolon, as the following example shows:

CACLP; !Displays the current date and time!

The remark can also be written between the MML command and the semicolon. Then only one exclamation mark in front of the remark is required:

CACLP !Displays the current date and time;

The remark will be logged together with the MML command.

The following example shows how remarks can be included in a command file:

```
@@ Open the Input Output window
! Clear the input output window
@INKEY() @@ Wait for a key to be pressed
@WINDEND !Close the window
```

3.1.14 Command and Function Names

The OPS script language provides a large number of commands and functions. The names of the commands and functions are not reserved words. If a command name or a function name is not expected, a command name or a function name can be used as a variable name. For example, the command @SET DELAY = 100 is valid even though there is a command called @DELAY. The syntax check currently indicates an error (in red). All function calls require a pair of brackets even if no parameter is supplied. These brackets distinguish the function name from a variable name.

3.1.15 Variables

A variable name is a variable whose value is set at execution by commands such as @INPUT, @SET, and @READ

A variable is defined by the following syntax:

Var ::= Var_char

Var Var_char
Var_char ::= any printable character except
< > = + - * / ( ) , ' " { } @
The contents of a variable can be accessed if a value has been assigned to the variable. The function VAREXIST can be used to test whether a variable is set.
The value of a variable persists until one of the following situations occurs:

1. A new value is assigned to the variable.
2. The variable is removed by the @FLUSHVAR command.

There is no logical limit to the number of variables allowed. However, the number can be limited by the amount of memory available at execution time. Variable names are not case sensitive.

All variable values are of string type with length up to 1000 characters. When a numeric value is assigned to a variable, implicit conversion from numeric value to string is done. A variable name must not conflict with any of the following constraints:

— There is no limit for the variable name length.

— A variable name must not start with a digit, the ($) character, or the (#) character.

3.1.16 Variable Substitution

Variable substitution can be used to replace a variable name or a part of a variable name.

All command file variables hold string type values. When numeric values are assigned to variables, these are converted into strings. The variables are substituted into their current values before the source code is parsed.

Variable substitution is denoted by {}. More than one variable substitution is allowed on the same command file line, and the variable substitutions can be nested.

Use of Variable Substitution

The following example illustrates how variable substitution is used:

```
@SET SNB = 9876543
SUSCP:SNB={SNB},LIST; !Get the category!
```

Value of Substituted Variable is Unknown until @COMMENT Command is Executed

Variable substitution must be used at run-time because variables can take different values at run-time, and the substitution value is the latest value assigned to a variable.

In the following example, the value of VAR to be substituted is unknown until the @COMMENT command is executed.

Differences between Variable Substitution and Variable Evaluation

There is a difference between variable substitution and variable evaluation as shown.

```
@SET VAR = "123+456"
@ "123+456" is printed @COMMENT({VAR}) @@ "579" is printed
```

In the first @COMMENT in the example, no variable substitution occurs. The contents of the string is therefore not parsed, and the string "123+456" is printed.

The second @COMMENT uses variable substitution. After variable substitution, the source line has become @COMMENT(123+456). The arithmetic of the expression is then visible to the parsing mechanism, and 579 is printed.

Despite the fact that variable substitution is denoted by {}, there are some situations where no substitution occurs:

1. If the variable to be substituted is not defined at execution time, no substitution is performed.
2. If the braces do not match no substitution is performed.
3. Substitution in @LABEL command is not allowed. The jump in the following example will not be successful.

No Successful Jump

```
@SET MYLABEL = "ABC"
@GOTO ABC @LABEL {MYLABEL}
```

Variable substitution can be used in other commands referring to program labels, like the following example:

Variable Substitution to Refer to Program Labels

```
@SET MYLABEL = "ABC"
@GOTO {MYLABEL} @LABEL ABC
```

3.1.17 Simulated Arrays

Variables can be used to simulate arrays. In the following example, a file is opened, and the ten first lines of the files are read into a simulated array:

@FOR X = 0 TO 9 @SET LINE{X} = "" @NEXT X @OPENREAD(FILE) @FOR X = 0 TO 9 @READ(LINE{X}) @NEXT X

3.1.18 String Constants

A string constant is defined by the following syntax:

String_constant ::= " Char_sequence "

The following example displays the use of different quotation characters:

' Char_sequence '
Char_sequence ::=
Any member of the source character set except the new-line character, or the enclosing quotation mark, if any.
A string constant must not conflict with any of the following constraints:

- The length of a string constant must not exceed 1000 characters.
- The same type of quotation character, (") or ('), that encloses the value of the constant must not appear within the value of the constant (see the example below).

@SET VAR2 = "The "VAR2" value" @@ Syntax Error @SET VAR2 = "The 'VAR2' value" @@ This is ok

@SET VAR2 = 'The "VAR2" value' @@ This is also ok

3.1.19 Character Constants

A single-character constant is defined by the following syntax:

Char_constant ::= #Decimal_number

$Hex_number

Decimal_number ::= Decimal_digit

Decimal_number Decimal_digit Decimal_digit ::= one of 0 1 2 3 4 5 6 7 8 9
Hex_number ::= Hex_digit

Hex_number Hex_digit

Hex_digit ::= one of 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

The number must be a legal value within the range of the character set used by the execution environment.

The first form, #Decimal_number, defines a character within the character set of the execution environment by a decimal value.

The second form, $Hex_number, defines a character by a hexadecimal value.#10 or $A defines new line if the character set used is ASCII.

3.1.20 Numeric Constants

A numeric constant is defined by the following syntax:

Numeric_constant ::= Decimal_number Decimal_number ::= Decimal_digit

Decimal_number Decimal_digit

Decimal_digit ::= one of 0 1 2 3 4 5 6 7 8 9

A numeric constant must not conflict with any of the following constraints:

— The decimal number range handled by a command file is from -2,147,483,648 to +2,147,483,647.

— Only integers are handled.

3.1.21 Labels

A label is defined by the following syntax:

Label ::= Label_char

Label Label_char

Label_char ::= Any printable character except ") " or " , "

Labels are defined by the @LABEL command and are used in conjunction with the @GOTO and @GOSUB commands. A label defines the position to which a jump from a @GOTO or a @GOSUB command is performed.

There is no limit for the label name length. The same label must not appear twice in the same source file. Variable substitution is not allowed in labels in the @LABEL command.

Jumping and Command Blocks

There is no restriction where the following commands can be used in the code:

— @GOTO

— @GOSUB

— @LABEL

Jumping into and out of command blocks can confuse the command interpreter. It is recommended to avoid placing these script elements inside the following command blocks:

— @IF-THEN (if longer than one lines

— @IFERROR-THEN (if longer than one line),

— @ELSE

— @FOR-TO

— @SWITCH

If the first two (@IF-THEN, @IFERROR-THEN) are only one-line commands, using @GOTO or @GOSUB does not affect the interpreter. The interpreter can also handle jumps inside the same block. If you must jump out of a block, do not place the label inside another block.

Consider the following examples if the running of your script stops near a @GOTO (or @GOSUB) statement.

The following example shows a code fragment that results in error. This is because there is a jump out of the @IF-THEN block and the label is inside the @FOR-TO block.

A Wrong Use of @GOTO Example

@set y = 3

@for x = 1 to 7 @if y = x then

...

@goto CHECK_DONE @endif

...

@label CHECK_DONE @next x

Using a one-liner @IF statement, @GOTO and @LABEL are in the same block (in this case: in a @FOR-TO block).

The next two examples give alternatives of how the error can be avoided.

Correct Use of @GOTO Example

@set y = 3

@for x = 1 to 7

@if y = x then goto CHECK_DONE

...

@next x

Another Correct Use of @GOTO Example

@if <condition1> then

@if <condition2> then

...

@goto LABEL1 @endif

...

@endif

@label LABEL1

This solution is also correct because, although it jumps out of two blocks, the label is not inside a block.

3.1.22 Expressions

An expression is defined by the following syntax:

Exp ::= Prefixed_exp

| Exp Arith_op Exp

Prefixed_exp ::= + Simple_exp

| - Simple_exp

|Simple_exp

Simple_exp ::= Var

| OPS_script_function

| Constant

| (Exp)

Arith_op ::= one of + - * /

3.1.23 Operators

An operator is defined by the following syntax:

Rel_op ::= one of < >= = <> <= > Arith_op ::= one of + - * /

Logical_op ::= one of AND OR

Rel_op and Logical_op can only be used in the condition expression of the

@IF-THEN and @IFERROR THEN commands.

Each operator, except unary + and -, acts on two operands. The operators men the following:

< Less than

<= Less than or equal to

= equal to

<> Not equal

>= Greater than or equal to

> Greater than

+ Sum

- Difference

*Product

/ Quotient

-(unary) Negation

+ (unary) No effect

AND Logical AND

OR Logical OR

The associative rules for interpreting the operators are listed in the following table:

Operators	Associative Rules
+ (unary), - (unary)	right to left
+ - * /	left to right
< >= = <> >= >	not applicable
AND OR	left to right

The operators are listed in decreasing precedence from top to bottom. When a numeric operation that includes a text string is carried out, the text string is converted to an integer value. The following example displays that variable y defines the sum of a string variable and an integer.

@SET X = 123 @@123 converted to string

@SET Y = X + 1 @@X converted back to integer

3.1.24 Default Values

The following table shows the values of the parameters on the start-up of a command file. During the execution of the file, these values can change.

Table 1 Default Values

Parameter	Default
Current directory	/home/<user>/cha/cmdfile
Ordered	On
Error flag	Cleared
Variables	None defined
Stack (for return from @GOSUB and @INCLUDE)	Empty
Printer logging	Off
File logging	Off
Check	On
Function busy retry	Configurable parameters
Setreply OK, FAIL strings	Empty
Input Output window	Cleared and closed
Comment window	Cleared and closed
@WINDOW	Set to not active

3.1.25 Error Handling

The following are error handling commands and functions:

— @CHECK Enables or disables error checking

— @IFERROR THEN Tests if the error flag has been set

— @SETERRORFLAG Sets the error flag

4 OPS Script Commands and Functions

The commands and functions are classified into one of the following categories:

— FIOL commands

— OPS script functions

— OPS script commands

OPS script commands, functions, and FIOL commands can be used in the same command file. This chapter describes each command use and OPS scripts function.

4.1 OPS Script Command Descriptions

This section describes the commands of the OPS script language using the metalanguage described in Typographic Conventions in About This Document on page 1 . For each command, synopsis and parameters are described. A description of the purpose of each command is also presented. It is also stated what effect the command has on the error flag.

When appropriate, examples have also been included in the command descriptions.

4.1.1 Resume Execution !$$$$

Synopsis

!$$$$!

Parameters

None

Description

When the !$$$$! command is encountered in a subfile opened through an @INCLUDE command, the execution will resume at the line following @INCLUDE in the original file.

If !$$$$! is encountered in a file that has not been opened through an @INCLUDE command, the execution is stopped. All run-time information, such as variable values, is retained.

Error Handling

The command has no effect on the error flag.

The command file in Example 15 is a subfile that is called from the original file by the @INCLUDE command. When the subroutine has been performed and the

!$$$$! command is encountered, the execution resumes at the line after the

@INCLUDE command in the original file.

Example 2 Resume Execution !$$$$!

```
@@ file_1.cmd @COMMENT("File 1")
.
.
@INCLUDE("file_2.cmd")@COMMENT("Back in file 1")
.
.
@@ file_2.cmd @COMMENT("File 2") @CLRSCR
@DRAW(BOX,19,4,55,6,CUP,20,5,"not Implemented ") @DRAW(CUP,24,8,
"Hit Any Key to Continue...") @INKEY()
@WINDEND
!$$$$!
```

Related Commands

@INCLUDE

4.1.2 @BREAK

Synopsis

@BREAK

Parameters

None

Description

The @BREAK command is used to interrupt an ongoing @FOR loop. The script execution continues right after the @NEXT command.

Error Handling

This command has no effect on the error flag.

@FOR ...

...

@IF ... THEN BREAK

... @NEXT

Example 16 @BREAK

Related Commands

@FOR-TO

4.1.3 @CHDIR

Synopsis

@CHDIR(Exp)

Parameters

Exp The path name of the new current directory

Description

This command changes the current directory to the directory specified by Exp.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

The tilde character (), which specifies a user's home directory, can be used in the parameter. For example, "/cha/cmdfile " is the command file directory of the current user, and "~nmsadm/cha/cmdfile " is the command file directory of the system administrator.

Error Handling

The command fails, and the error flag is set, in the following situations:

1.The specified path is not a valid path.

2. The specified directory does not exist.

3. You do not have permission to access the directory.

Example 3 @CHDIR

```
@CHDIR("ROUTINES")
...
or
...
@SET DIRNEW = "ROUTINES" @CHECK("OFF") @CHDIR(DIRNEW) @CHECK("ON →
")
@IFERROR THEN GOTO XERROR
```

4.1.4 @CHECK

Synopsis

@CHECK(Exp)

Parameters

Exp Evaluated to either "ON" or "OFF"

Description

When the error checking is turned on and a command fails, the execution is stopped and an error message is displayed.

When the error checking is turned off, the execution continues if a command fails.

The default mode is ON.

If the @IFERROR THEN command is used to check if a certain command has been properly executed, the @CHECK("OFF") command must first be

executed. @CHECK("OFF") suppresses the error message that would normally be generated if the command fails. Instead, the error flag will be set.

Error Handling

@CHECK("OFF") clears the error flag while @CHECK("ON") has no effect on the error flag.

In , an @IFERROR THEN command is used to check if the INFIP command has been properly executed. The @CHECK("OFF") command must first be executed

to prevent the error message that would be generated if the command INFIP fails.

Example 4 @CHECK

```
@COMMENT("STEP 11 ^ Checking file FD0A1 is defined..") INMCT:SPG →
-0;
@CHECK("OFF") INFIP:FILE-FD0A1; @CHECK("ON")
@IFERROR THEN GOTO STEP11A@@if not found
```

Related Commands

@IFERROR THEN

@SETREPLY

4.1.5 @CLOSE

Parameters

Exp The number of the log file to be closed. Evaluated to a number >= 1. For compatibility reasons an omitted parameter is interpreted as "1".

Description

This command closes a log file that is currently open. If the indicated log file is not open, the command has no effect.

Error Handling

The command has no effect on the error flag.

Related Commands

@LOG

@C

@L

4.1.6 @CLRSCR

Synopsis

@CLRSCR

Parameters

None

Description

This command clears the Input Output window and the Comment window. It also resets the current position in the Input Output window to (1,1).

Error Handling

The command has no effect on the error flag.

The @CLRSCR command can be used before a command that uses the Input Output window or the Comment window. This ensures that the Input Output window, or the Comment window, is empty when a new text is displayed in it.

Example 5 @CLRSCR

```
@WINDOW @CLRSCR
@COMMENT(" ^ Load 07RCB")
```

4.1.7 @COMMENT

Synopsis

```
@COMMENT[ ( [ Exp {, Exp } ] ) ]
```

Parameters

```
Exp Any expression
```

Description

If the

```
 @WINDOW
```

command is not in effect, the

```
@COMMENT
```

command displays a text string specified by one or more concatenated Exp parameters in the Command File Output area of the user interface.

If the

```
@WINDOW
```

command is in effect, the output text is displayed in the Comment window, which is opened by the

```
@WINDOW
```

command. If no parameter is supplied, a new line will be printed.

The following example shows how the

```
@COMMENT
```

command is used to display the current value of a variable on the screen.

Example 6

```
@SET VAR = 123
@IF x > 10 THEN SET VAR = 234
@COMMENT("The value of VAR is ", VAR)
```

4.1.8 @CONNECT

Synopsis

```
@CONNECT(Connect_Exp)
 where
_Connect_exp_ ::=
Exp \[\[, Exp1\]\[, Exp2\]\]
```

Parameters

Exp The name of the external system to be connected

Exp1 This parameter indicates which side is to be connected, either active side (`Executive`, as default) or passive side (`Standby`).

Description

This command is used to establish a connection to the specific external system through IOG or APG. Any existing connection will be disconnected.

When the NE name represents a board in an MSC Blade Cluster, OPS GUI prefixes the names of the individual blades with the name of the blade cluster and two underscore characters ( ). For example, the CP board of the following example, would display as `MSC2 CLUSTER1-CP1` in OPS GUI.

A connection is established to a CP in an MSC-Server Blade Cluster and an MML command is sent. The NE name and the CP name should be given as separate parameters without the connecting double underscore characters.

```
@CONNECT("GSM01BSC01") CACLP; @DISCONNECT
@CONNECT("NE=GSM01BSC01, NODE=B")
@DISCONNECT
```

Connections to MSC-BC CP groups are not supported by OPS.

In the next example, a connection to the active side ("NODE=A" as default) of an external system is established and an MML command is sent.

Then, the external system is disconnected and a connection to the passive side of another external system is initiated. OPS activity finishes with disconnection from the NE.

```
@CONNECT("NE=GSM01BSC01,
CPNAME=CLUSTER1-CP1") CACLP;
@DISCONNECT
```

Error Handling

The error flag is set if the connection attempt fails.

Related Commands

— `@DISCONNECT`

— `GETDEST`

4.1.9 @DEC

Synopsis

@DEC(Var) Parameters

Var A defined variable that has a numeric value

Description

This command decreases the value of the specified variable by 1. If the resulting value is less than -2,147,483,648 (the lower limit for decimal numbers), Var is set to 2,147,483,647 (the upper limit for decimal numbers).

Error Handling

The command has no effect on the error flag.

Related Commands

```
@INC
```

4.1.10 @DELAY

Synopsis

```
@DELAY(Exp )
```

Parameters

`Exp`

The time during which the command file execution is paused expressed in number of minutes (non-negative numeric value)

Description

This command pauses the command file execution for Exp number of minutes. During this period, the countdown in seconds is displayed to the far left field in the main window footer.

Error Handling

The command has no effect on the error flag.

In the following example, the

```
@DELAY
```

command is used to display the current time in the Input Output window.

```
@LABEL TIMENOTE @@Loop that displays the current time @CLRSCR
@SET DATE&TIME = GETDATE ()
@SET TIME = COPY(DATE&TIME, 7, 4) @DRAW(CUP,69,1,TIME)
@DELAY(1) @GOTO TIMENOTE
```

Related Commands

```
@DELAYSEC
```

4.1.11 @DELAYSEC

Synopsis

```
@DELAYSEC(Exp )
```

Parameters

`Exp` The time when the command file execution is paused expressed in seconds (a non-negative numeric value)

Description

This command pauses the command file execution for Exp number of seconds. If Exp is larger than 5, the countdown in seconds is displayed to the far left field in the main window footer.

Error Handling

The command has no effect on the error flag. In the following example, the

```
@DELAYSEC
```

command creates a stop of five seconds between the display of the information text "S12 Meter Reading Program" and the request to enter a customer service number.

```
@CLRSCR
```

```
@DRAW(CUP,20,6,"S12 Meter Reading Program") @DELAYSEC(5)
```

```
@CLRSCR
```

```
@INPUT(DN1, "Enter Customers Service Number")
```

Related Commands

```
@DELAY
```

4.1.12 @DISCONNECT

Synopsis

```
@DISCONNECT
```

Parameters

None

Description

This command disconnects any established connection. If no connection is established, the command has no effect.

Error Handling

The command has no effect on the error flag.

In the following example, the

```
@DISCONNECT
```

command disconnects from an external system to enable connection to another external system.

```
@CONNECT("AXE10") CACLP
```

```
@DISCONNECT
```

```
@CONNECT("AXE15")
```

Related Commands

```
@CONNECT
```

4.1.13 @DRAW

Synopsis

```
 @DRAW(Draw_exp {, Draw_exp })
```

Parameters

Each Draw_exp parameter can take on any of the following forms:

`BOX`

Draws a box in the output window;

`RightX, LowerY` RightX,LowerY are the coordinates of the lower right corner

`LeftX, UpperY` LeftX , UpperY are the coordinates of the upper left corner

`CUP, X, Y` Moves the current output position in the Input Output window to the coordinates (X,Y)

`Exp` Arbitrary text to be displayed

Description

This command allows elaborate output to the Input Output window. If the Input Output window has not been opened, this command automatically opens it.

The size of the Input Output window is 80 columns by 7 to 25 rows determined by the maximum value used for coordinate Y. If a text line exceeds 80 characters, only the first 80 characters will be displayed.

When drawing boxes using the

```
@DRAW
```

command, any contents inside the box will be erased.

Error Handling

The command has no effect on the error flag. See the example of the command.

Note: Swapping the order of the BOX and CUP expressions causes the box to be empty. This is because when drawing a box the contents inside the box is erased.

Related Commands

```
@FORM
```

```
@MENU
```

4.1.14 @ELSE

Synopsis

```
@ELSE
```

Parameters

OPS_script_command Any OPS script command except

```
@LABEL
```

,

```
@ENDIF
```

,

```
@ELSE
```

,

```
@FOR
```

or

```
@NEXT
```

Description

This command must be preceded by an

```
@IF-THEN
```

command in the file. The specified command is executed if the condition made by the

```
@IF-THEN
```

command is not fulfilled. If no other OPS script command is specified on the same line as the

```
@ELSE
```

command the subsequent commands between the

```
@ELSE
```

and the

```
@ENDIF
```

commands will be executed.

Error Handling

The command has no effect on the error flag. In the following example, the

```
*@IF-THEN*
```

command is used to display the current day of the week, after the GETDATE function has been used to retrieve information from the system. For an example of the use of the

```
*@ELSE*
```

command without<ac:structured-macro ac:name="anchor" ac:schema-version="1" ac:macro-id="134514ce-0bd7-44f1-9c43-31bcd52b4482"><ac:parameter ac:name="">_bookmark109</ac:parameter></ac:structured-macro> another OPS script command on the same line, see the example of the

```
@COMMENT("Is it Monday?")
```

```
@SET X = GETDATE(Y)
```

```
@IF Y = 1 THEN
```

```
@COMMENT("Yes, it is Monday.")
```

```
@ELSE
```

```
@COMMENT("No, it isn't Monday.")
```

```
@ENDIF
```

```
@COMMENT("Thanks")
```

Related Command

```
@IF-THEN
```

```
@IFERROR THEN
```

4.1.15 @ENDIF

Synopsis

```
@ENDIF
```

Parameters

None

Description

This command is used to indicate the end of the last alternative in the

```
@IF-THEN
```

and

```
@IFERROR THEN
```

commands.

Error Handling

The command has no effect on the error flag.

See the

```
@IF-THEN
```

command example.

Related Commands

```
@IF-THEN
```

```
@IFERROR THEN
```

4.1.16 @ERASE

Synopsis

```
@ERASE(Exp )
```

Parameters

Exp
The path name of the file to be deleted

Description

This command erases the file specified by Exp.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

You must have permission to write to and execute files in the directory containing the file to be erased.

Error Handling

If the specified file cannot be removed, the error flag will be set. However, if the specified file does not exist, the command is considered as successful.

In the following example, the contents of an existing log file are erased, before the log file is opened.

Example 7 @ERASE

```
@ERASE
```

("logfile.txt") !Old data is removed

```
@LOG
```

("logfile.txt") !New log is opened

4.1.17 @EXECUTE

Synopsis

```
@EXECUTE(Exp )
```

Parameters

`Exp`

An expression containing an external command, that is, a UNIX command.

Description

This command executes an external command in the current directory and then returns to the command file.

By default, the result of the execution is displayed in the Command File Output Area. If >/dev/null is appended to a command, the execution result is not displayed.

The commands that require user-interaction cannot be used as parameters. For example, to view the content of the file, the cat command must be used instead of more.

Error Handling

The error flag is set if the external command cannot be executed.

Examples of UNIX commands executed by the

```
@EXECUTE
```

command are shown in the following example:

```
@EXECUTE
```

("ls | sort -u") @EXECUTE("xterm -e vi")

```
@EXECUTE
```

("tar cvf backup.tar ^cmd >/dev/null")

Note: When the command `dos2unix` is executed from OPS, it is recommended to use the `-n` flag to avoid any permission errors during execution. The `-n` flag must be used if change in owner and group of the file has no impact on the use case.

For example,

```
@EXECUTE("dos2unix -n /var/tmp/a.txt /var/tmp/b.txt")
```

4.1.18 @FLUSHVAR

Synopsis

```
@FLUSHVAR(Var ) Parameter
```

Parameters

`Var`

The name of the variable to be removed

Description

This command is used to remove a variable.

Error Handling

The command has no effect on the error flag.

```
@INKEY(ANSWER
```

, "Type Y or 11 :")

```
@IF ANSWER
```

= "Y" THEN GOSUB DOIT

```
@FLUSHVAR(ANSWER
```

)

4.1.19 @FOR-TO

Synopsis

@FOR Var = Exp1 TO Exp2

@NEXT Var

Parameters

Var A loop counter

Exp1 The initial value of the loop counter

Exp2 The final value of the loop counter

Description

This command indicates the beginning of a @FOR -loop structure. The loop counter is defined by the initial value Exp1 and the final higher value Exp2..

The end of the @FOR -loop is indicated by the @NEXT command. For each @FOR-TO command, there must be one corresponding @NEXT command in a later part of the file.

Nested @FOR -loops must not iterate on the same variable.

Error Handling

The command has no effect on the error flag. If value of Exp2 is less than value of Exp1 the @FOR -loop is skipped and the script execution continues on the line after the @NEXT command.

The function of the @FOR -loop could also be implemented by @LABEL and @GOTO, which helps explaining the effect of the @FOR-TO command. The command files in Example 31 and Example 32 are conceptually equivalent.

@FOR v = exp1 TO exp2

...

@NEXT v

@SET v = exp1 @SET final = exp2 @LABEL startloop

@IF v > final THEN GOTO endloop

...

@SET v = v + 1 @GOTO startloop @LABEL endloop

Example 30 @FOR-TO

In Example 32, a @FOR -loop is used to remove the variables VAR1 to VAR20..
@FOR X = 1 TO 20

@FLUSHVAR(VAR{X}) @NEXT X

Example 31 @FOR

Related Commands

@BREAK

@NEXT

4.1.20 @FORM

Synopsis

@FORM(Form_exp {, Form_exp })

where

Form_exp ::=

Exp1, Exp2, Exp3, Var | FIELD(Exp1, Exp2, Exp3, Var)|

COMBO(Exp1, Exp2, Var, Exp4 |Expn {,Expn })| CMDBUTTON(Exp1, Exp2, Exp5, Exp6)|

CHECKBOX(Exp1, Exp2, Exp5, Var)|

RADIOBUTTON(Exp1, Exp2, Exp5 {, Exp1, Exp2, Exp5 }, Var)

Parameters

Each Form_exp expression defines one of six different input features where user entry is allowed. All seven features have the following parameters in common:

Exp1 The column number for display of the input featuret or text in the Input Output window

Exp2 The line number for display of the input feature or text in the Input Output window

The first feature is a plain input field. It also requires the following parameters:

Exp3 The length of the input field

Var A variable that stores the input entered by the user

The second feature, with the indicator FIELD, is also a plain input field: It requires the same parameters and it is functionally identical with the first alternative.

The following four features make use of different options for user entry:

COMBO specifies a "closed drop-down combo box" feature (where the list of value options is taken from a file or from the command specification itself, but with a typing possibility). The first line of the file (or the first item in the specification) is displayed as the default value. The default value can be left empty to encourage user input. This is accomplished by letting the first Expn

be an empty string (" "), or, if a file is used as parameter, letting the first line of the file be an empty line. The Exp1 and Exp2 are the same as above and

Exp4 A path name of a file containing the values to be displayed in the combo box when expanded

Expn Explicit values to be displayed in the combo box when expanded

Var A variable that stores the chosen value CMDBUTTON specifies a command button feature. It works like the Apply

button, recognizing the values of the input fields, only that the execution continues at a specified label Exp6 in the OPS script, corresponding to the text of Exp5. The Exp1 and Exp2 are the same as above and

Exp5 The text to be displayed on the button

Exp6 The label in the script to which a jump is performed when pressing the command button

CHECKBOX specifies a checkbox feature. The checkbox variable can only take two values, one or zero for true or false, depending on whether the box is selected or not. If the variable already has an assigned value (=1) the checkbox is shown "checked". The Exp1, Exp2 are the same as above, and

Exp5 A text string describing the function to be enabled if the checkbox is selected

Var A variable that says True (1) or False (0)

RADIOBUTTON specifies a radio button feature which has the radio button variable. The Exp1 and Exp2 are the same as above, and

Exp5 A text string describing the function associated with the radio button clicked by the user

Var A variable that takes the value Exp5

Several RADIOBUTTON expressions are possible within the @FORM command, each with a set of radio buttons and a variable to store the selected radio button.

Description

The Description command allows you to request user input in the Input Output window. If the Input Output window has not been opened, this command automatically opens it.

When the command is executed, the input features defined by the command parameters are displayed in the Input Output window. Each input feature definition is made up of four parameters. An arbitrary number of input features can be defined.

If Var is defined when the command is executed, the value of Var will be displayed as default value on the input line. If the user confirms the input by clicking the Apply or some command button without specifying text for a variable in the @FORM command, an empty string is assigned to the variable.

The @DRAW command can be used to create boxes and texts that make up labels for the first three of the input features described above.

Error Handling

The command has no effect on the error flag.

In Example 32, two input fields, a pull down menu (COMBO box), a command button and a set of radio buttons are created by the @FORM command. The

@DRAW command is used to create a text for each input field and for the COMBO box. The data entered is assigned to the variables VAR1-VAR4.

@CLRSCR @DRAW(BOX,4,2,30,4,

CUP,6,3,STYLE(BOLD,"COMMAND FILE ENTRY"),

CUP,6,6,"Specify external system", CUP,6,9,"Specify command file",
CUP,6,12,"Specify time for execution", CUP,6,13,"(YYMMDDHHMM)")

@FORM(COMBO(35,6,VAR1,"cha/cmdfile/nefile"),
35,9,30,VAR2,35,12,10,VAR3,
RADIOBUTTON(6,17,"Stepwise",17,17,"Automatic",VAR4),

CMDBUTTON(6,19,"Execute now","Now"))

Example 32 @FORM

When the commands are executed, the form appears in the Input Output window as shown in Figure 5.

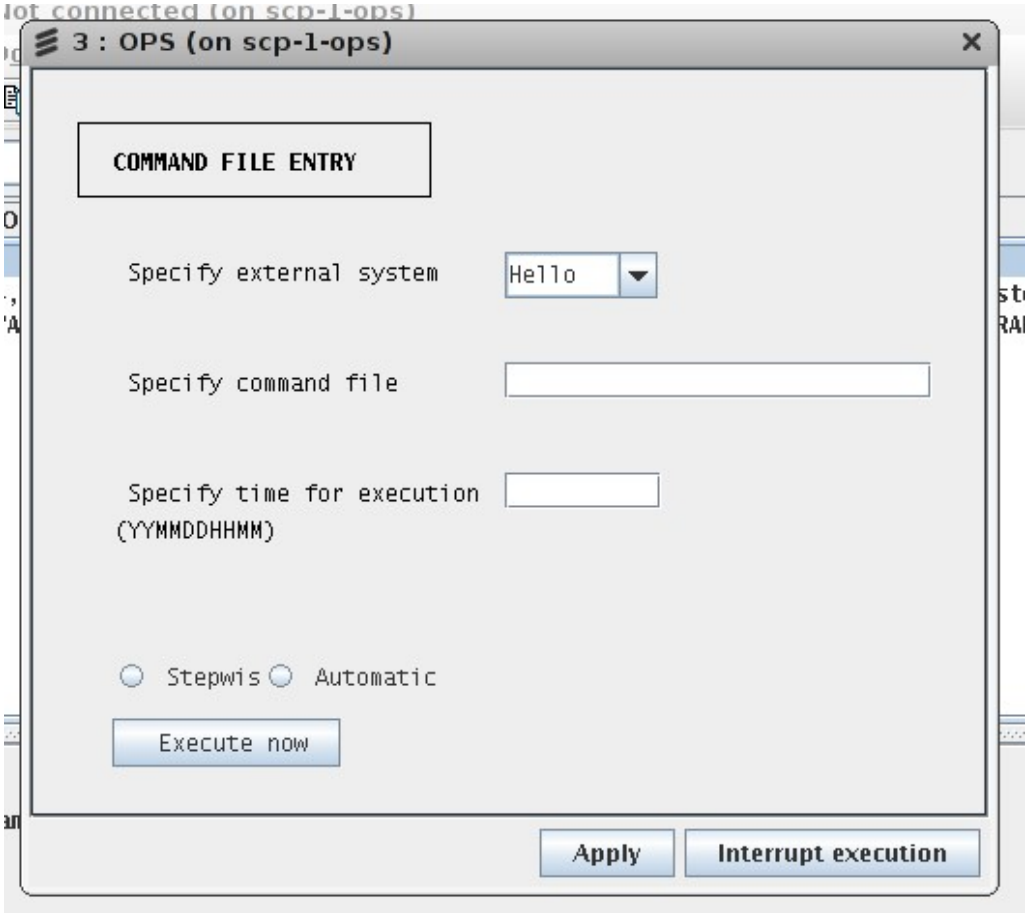

Figure 1 Figure 5 Input Output Window with Input Requests

Related Commands

@DRAW

@MENU

4.1.21 @FUNBUSY

Synopsis

@FUNBUSY({_}Exp1_ \[, _Exp2_ \]

Parameters

Exp1 The total number of attempts that will be made to execute an MML command

Exp2 The interval between the attempts, expressed in seconds (optional)

Description

This command specifies the total number of attempts that will be made to execute an MML command when the FUNCTION BUSY or FUNCTION BUSY - SESSION LOCKED response is encountered.

The default parameter values are hardcoded to 1 for both parameters. Omitted parameter or parameter=0 means the default value will be used.A failing renewed attempt does not affect the invisible buffer mark.

Error Handling

The command has no effect on the error flag.

4.1.22 @GOSUB

Synopsis

@GOSUB

Label Parameters

Label Starting point in a subroutine to which a jump is performed

Description

This command is used to call a subroutine that starts at the command @LABEL Label. The @GOSUB commands require an @LABEL command that indicates the position to which a jump is performed and an @RETURN command that orders a jump back to the line following the @GOSUB command. The @LABEL command, the @RETURN command, and the @GOSUB command must be in the same file.

Subroutine calls can be nested.

Error Handling

The command has no effect on the error flag.

In Example 33, a subroutine that checks the state of the connected external system, XGESTATE , is called. When this check has been performed, the

operator is asked to answer a question with yes or no. Depending on the answer, another subroutine is called.

@LABEL STEP1 @GOSUB XGESTATE CHREP;

@INKEY(CHOICE, "Are we ready to start?") @IF CHOICE = "Y" THEN GOSUB READY

@GOTO NOTREADY @LABEL XGESTATE

@COMMENT(" * Checking external system state..") ALLIP;

DIREP:EMG=ALL; STRSP:R=ALL; EXEPP:EMG={EMG}0,EM=ALL; STSTP:EMG={EMG}0,EMTS=ALL; @RETURN

...

@LABEL READY

... @RETURN

...

@LABEL NOTREADY

...

Example 33 @GOSUB

Note: For possible problems when using @GOSUB in command blocks, see Page 47.

Related Commands

@LABEL

@GOTO

4.1.23 @GOTO

Synopsis

@GOTO

Label Parameters

Label A point to which a jump is performed

Description

This command orders a jump to the command @LABEL Label, at which the execution continues. The @GOTO commands requires a @LABEL command that indicates the position to which a jump is performed. The @GOTO command and the @LABEL command must be in the same file.

Error Handling

The command has no effect on the error flag.

@LABEL START

@COMMENT("An endless loop...") @GOTO START

Example 34 @GOTO

Note: Note: For possible problems when using @GOTO in command blocks.

Related Commands

@LABEL

@GOSUB

4.1.24 @IF-THEN

Synopsis

@IF Cond_exp THEN [OPS_script_command]

where

Cond_exp ::=

Simple_cond_exp { Logical_op Simple_cond_exp }

Simple_cond_exp ::= Exp Rel_op Exp

| (Cond_exp)

Rel_op ::=one of < <= = <> >= >

Logical_op ::= one of AND OR

Parameters

Cond_exp The condition that must be fulfilled for OPS_script_command to be executed

OPS_script_command Any OPS script command except @LABEL, @IF-THEN, @IFERROR THEN, @ELSE, @ENDIF, @FOR , or @NEXT.

Description

This command evaluates the condition made by Cond_exp. If the condition is fulfilled, the command specified by OPS_script_command is executed. When the command has been executed, the execution continues on the first line after the @IF-THEN command that does not contain an @ELSE command.

If the condition made by Cond_exp , is not fulfilled, the command is not executed. Instead, the execution continues on the line following the @IF-THEN command.

Several commands can be defined for each of the two alternative conditions. This is achieved by letting the absence of information after the "THEN" part of the @IF-THEN statement indicate the use of this facility. The @ENDIF command is used to indicate the end of the last alternative.

In this case, the execution continues on the first line after the @IF-THEN command if the condition is fulfilled, up until the @ELSE or @ENDIF command. If the condition is not fulfilled, the execution starts after the @ELSE or @ENDIF command.

Error Handling

The command has no effect on the error flag.

With one command after THEN

With no command after THEN,

@IF a=0 THEN @SET b=1 @SET c=2 @ELSE

@SET b=2 @SET c=3 @ENDIF

Example 35 @IF-THEN

or (if action is needed only if condition is fulfilled), see Example 37:

@IF a=0 THEN @SET b=4 @SET c=5 @ENDIF

Example 36 @IF-THEN

Related Commands

@ELSE

@ENDIF

4.1.25 @IFERROR THEN

Synopsis

@IFERROR THEN\[{_}OPS_script_command_ \]

Parameters

OPS_script_command Any OPS script command except @LABEL, @IFERROR THEN, @IF-THEN, @ELSE, @ENDIF, @FOR, or @NEXT

Description

This command tests if the error flag is set. If the error flag is set, the command specified by OPS_script _command is executed. If the error flag is not set, the command on the line following the @IFERROR THEN command line is executed.

If the @IFERROR THEN command is used to check if a certain command has been properly executed, the @CHECK("OFF") command must first be executed to suppress the error message that would normally be generated if the command fails.

Several commands can be defined for each of the two alternative conditions. This is achieved by letting the absence of information after the "THEN" part of the @IFERROR THEN command indicate the use of this facility, like for the @IF-THEN command.

Error Handling

The command clears the error flag.

In Example 37, the @IFERROR THEN command is used to check if a specified file could be opened for reading. The purpose of the @CHECK("OFF") command is to suppress the error message that is normally generated.

```
@SET FILE= "/users/sune/cha/cmdfile/cmd1.oz" @CHECK("OFF)

@OPENREAD(FILE) @CHECK("ON")

@IFERROR THEN GOTO XERROR

...

@LABEL XERROR

@COMMENT("The file could not be opened for reading.")
```

Example 37 @IFERROR THEN

Related Commands

@ELSE

@CHECK

@ENDIF

4.1.26 @SWITCH

Synopsis

@SWITCH (var)

@CASE value

[OPS_script_command]

@DEFAULT value

[OPS_script_command]

@ENDSWITCH

Parameters

Var A defined variable that has a numeric value

Value Numeric value to the defined variable

OPS_script_command A statement of any OPS script command.

OPS_script_command A statement of any OPS script command.

Description

This command compares the result of the variable to each numeric value. If it finds a match, the corresponding statement (simple or compound) executes. If no match occurs, the default statement executes.

Error Handling

The command clears the error flag.

In Example 38, the @SWITCH command is used to choose specific comments depending on the value of the variable VAR.

@set var = 2 @SWITCH (var)

@CASE 1

@COMMENT("CASE 1")

@CASE 2

@COMMENT("CASE 2") @DEFAULT

@COMMENT("DEFAULT") @ENDSWITCH

Example 38 ©SWITCH

4.1.27 @INC

Synopsis

@INC(Var)

Parameters

Var A defined variable that has a numeric value

Description

This command increases the value of the specified variable by 1. If the resulting value is greater than 2,147,483,647 (the upper limit for decimal numbers), Var is set to -2,147,483,648 (the lower limit for decimal numbers)

Error Handling

The command has no effect on the error flag.

In Example 39, the @INC command is used to implement a loop counter that displays the numbers 1-10 on the screen.

@SET V = 1 @SET FINAL = 10

@LABEL STARTLOOP

@IF V > FINAL THEN GOTO ENDLOOP @COMMENT(V)

```
@INC(V)

@GOTO STARTLOOP @LABEL ENDLOOP

@COMMENT ("The loop is ended.")
```

Example 39 @INC

Related Commands

@DEC

4.1.28 @INCLUDE

Synopsis

@INCLUDE(Exp)

Parameters

Exp The path name of the file to be included

Description

This command calls another command file and executes this file from the beginning to the end, or until a !$$$$! command is encountered. Then, the execution continues on the line immediately following the @INCLUDE command in the original file.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

Error Handling

If the file to be included is not accessible, the error flag will be set.

In Example 40, the user is asked to specify a file. The specified file is then included in the original command file.

```
@CHDIR("~/cmdfile/") @EXECUTE("ls -l") @DELAYSEC("10")

@INPUT(FILE,"Enter the name of the EMRP function change file")
@COMMENT("Loading ", FILE)

@INCLUDE(FILE)
```

@@load EMRP function change file! @IFERROR THEN PROMPT

Example 40 @INCLUDE

Related Commands

!$$$$!

4.1.29 @INKEY

Synopsis

@INKEY\[(\[_Var_ \{, _Exp_ \} \]) \]

Parameters

Var A variable to receive the input character (optional)

Exp A text message to be displayed as a prompt (optional)

Description

The command opens the Input pop-up window, which asks you to enter a single-character value. If Exp is omitted, a standard prompting message is displayed. You enter a character and confirms the input. The input character is assigned to the variable Var.

The execution is suspended until the user confirms the input. If you confirm the input without specifying a character, an empty string is assigned to Var.

Although Var can be defined when the command is executed, the value of Var

is NOT displayed as default value in the input window.

Error Handling

Example 41 @INKEY

The command has no effect on the error flag.

@INKEY(PRFLAG,"Do you want the printer on line [Y/N]?")

@IF PRFLAG = "Y" THEN PRINTER("ON")

4.1.30 @INPUT

Synopsis

@INPUT(Var {, Exp })

Parameters

Var A variable name to hold the user input

Exp A text message to be displayed as a prompt (optional)

Description

The command opens the Input pop-up window, which asks the user to enter a text string. If Exp is omitted, a standard prompting message is displayed. The user enters a text string and confirms the input. The input is assigned to the variable Var.

The execution is suspended until the user confirms the input. If the user confirms the input without specifying a text, an empty string is assigned to Var.

There is no limit for the input field length.

If Var is defined when the command is executed, the value of Var will be displayed as default value in the input window.

Error Handling

The command has no effect on the error flag.

@COMMENT(" C N A 8 P R O C E D U R E")

@COMMENT(" This procedure loads CNA8 into the stand-by CP") @INPUT (EMG,

"Enter external system LNCD code (for example, TCKX)") @INPUT(RELFSW, "Enter reload file to stand-by CP load from")

Example 42 @INPUT

4.1.31 @LABEL

Synopsis

@LABEL Label

Parameters

Label Specifies a point to which a jump is performed from an

@GOTO command or an @GOSUB command

Description

This command is used together with the @GOTO and @GOSUB commands.

@LABEL specifies the location in the file to which a jump is performed.

The same label name can only appear once in a command file.

This command must not appear as part of the @IF-THEN or @IFERROR THEN

commands.

Label name must not be variable substituted.

Error Handling

The command has no effect on the error flag.

@LABEL START

@COMMENT("An endless loop...") @GOTO START

Example 43 @LABEL

Note: Note: For possible problems when using @LABEL in command blocks, see Page 47.

Related Commands

@GOTO

@GOSUB

4.1.32 @LOG

Synopsis

@LOG({_}Exp_ \[,{_}Exp1_ \])

Parameters

Exp The path name of the log file to be used

Exp1 Log file number, evaluated to a number >=1. Default value of Exp1 is 1.

Description

This command specifies a file on which communication is logged. When the command has been executed, all further communication that is displayed in the Command File Output area will be logged.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

If this command is already active for another log file started with the same log file number, it will close the other file and instead start logging in the new file.

If the specified file does not exist, a new file is created. If the specified file does exist, any new output is appended to the file. If the contents of the log file is to be overwritten, the @ERASE command must first be used.

The log file is closed by the @CLOSE command.

This command provides the possibility to have several log sessions in parallel.

Error Handling

If the new log file specified cannot be opened, the error flag is set.

@LOG("logfile-a") @LOG("logfile-b",2)

...

... @CLOSE(1)

Example 44 @LOG

It will start two logfiles, logging the same information until logfile-a is closed by the @CLOSE(1) command.

Related Commands

@CLOSE

@C

@L

4.1.33 @LOGOF

Synopsis

@LOGOF

Parameters

None

Description

If @ORDERED("OFF") is in effect, delayed responses will not be displayed until the @LOGOF command is executed. Thus, @LOGOF makes it possible to display the delayed responses when it is convenient, for example at the end of the command file. Note that the @LOGOF command does not wait for delayed responses to arrive; it displays the responses that have arrived at the point when @LOGOF is called. However, @LOGOF can be used in conjunction with @WAITFOR to wait for a response and then display it.

The @LOGON command suspends the effect of the @LOGOF command.

Error Handling

The command has no effect on the error flag.

The delayed response generated by the LABUP command in the following example will not be displayed until the @LOGOF command at the end of the command file is executed. If the delayed response takes longer than 30 seconds to arrive, the response will not be printed.

@ORDERED("OFF"); LABUP;

...

@WAITFOR("END", 30) @LOGOF

Example 45

Related Commands

@ORDERED

@LOGON

@WAITFOR

4.1.34 @LOGON

Synopsis

@LOGON

Parameters

None

Description

The @LOGON command is used to suspend the effect of a previously executed

@LOGOF command.

A @LOGOF command allows ordered delayed responses that are suppressed by the @ORDERED("OFF") command to be received.

If a delayed response is just being received when the @LOGON command is executed, the reception of this response is completed. However, further

pending delayed responses will not be received until another @LOGOF command is executed.

The purpose with the @LOGON command is to give priority to important communication and suspend the reception of delayed responses until a more convenient occasion.

Any MML command also suspends the effect of the @LOGOF command.

Error Handling

The command has no effect on the error flag.

Related Commands

@ORDERED

4.1.35 @MARK

Parameters

Exp The number of lines by which the invisible buffer mark is to be shifted

Description

When an MML command is sent, an invisible buffer mark is automatically set at the line of the command.

The invisible buffer mark is not affected by renewed MML command entry attempts by the @FUNBUSY command after the response FUNCTION BUSY or FUNCTION BUSY - SESSION LOCKED, nor is it affected by the @SEND command or by programs executed by the @EXECUTE command.

The @MARK command shifts the mark in the buffer by the number of lines specified by Exp.

This mark can be moved an arbitrary number of lines. A positive number means that the mark is moved to a later position in the buffer and negative value means that the mark is moved to an earlier position. If the new position is outside the buffer range, the mark will be set at the buffer boundary.

If the command @MARK is executed without parameter, the invisible buffer mark is set at the last response line, normally the line after the last visible response line

Error Handling

The command has no effect on the error flag.

Example 46 shows how @MARK can be used to find more than one occurrence of the same text string. The command file scans the ALLIP printout for the two first occurrences of "APZ+APT". The comment at the end of the command file will read "APZ+APT was found on line 4 and line 12".

ALLIP;

-

```
-
-
ALARM LIST
A1/NETWORK "APZ+APT 1-6,8 9" 001 000013 2021 TT-OUTPUT ERROR
FILE NOT FOUND FILE NAME TTFILE00
A1/PROC "APZ+APT 1-6,8 9" 002 000013 2021 EMG FAULT
EMG UNIT STATE
LSS EMRP-2-A ABLOCK
-
-
-
@SET var1 = SCAN("APZ+APT")
@MARK(var1)
@SET var2 = SCAN("APZ+APT")
@COMMENT("APZ+APT was found on line ", var1, "and line", var1+var2)
```

Example 46 @MARK

Related Commands

GETMARK

4.1.36 @MENU

Synopsis

@MENU(Var, Exp1 , Exp2 {, Expn })

Parameters

Var The name of the variable that holds the result of selection

Exp1, Exp2 Numeric values within the range 0-21 that indicate the

possible number of options in column one and column two respectively in the menu

Expn Obsolete parameter

Description

This command is used together with the @DRAW command to enable menu selection in either one or two columns. The @MENU command returns the number of the item selected. The command also creates the buttons for the options. Texts for the options must be created with the @DRAW command.

Selection of the first choice on the left column returns the value 1, selection of the second choice on the left column returns the value 2, and so on.

The command makes use of the Input Output window. If the Input Output window has not been opened, the command automatically opens it.

The first menu choice is displayed on the third row in the Input Output window. Error Handling

The command has no effect on the error flag.

In Example 47, a two-column menu with six options is defined. The @DRAW

command is used to create the texts for the options.

@CLRSCR @DRAW(CUP,16,3,"Option 1",

CUP,16,4,"Option 2",

CUP,16,5,"Option 3",

CUP,56,3,"Option 4",

CUP,56,4,"Option 5",

CUP,56,5,"Option 6") @MENU(RESULT,3,3)

Example 47 @MENU

When the command file is executed, the menu appears in the Input Output window as shown in Page 90.

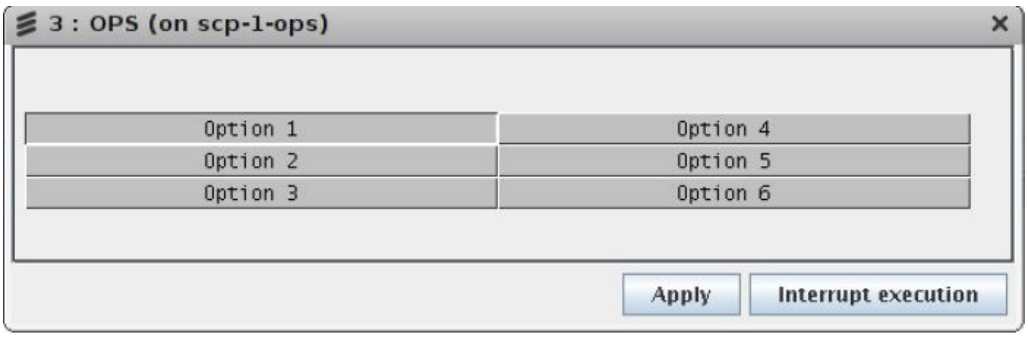

Figure 2 Figure 6 The Input Output Window

The user selects an option by clicking on the appropriate option and then clicking on the Apply button. One of the values 1-6 is assigned to the variable RESULT , depending on which option has been selected.

Related Commands

@DRA

@FOR

4.1.37 @MKDIR

Synopsis

@MKDIR(Exp)

Parameters

Exp The path name of the directory to be created

Description

This command creates a new UNIX directory. If only a directory name is specified by Exp, the new directory is created under the current directory.

If the parameter begins with the / character, the parameter is interpreted as an absolute path.

The directory to be created must be subordinate to an already existing directory. Thus it is not possible to create more than one directory at a time.

Error Handling

If the new directory cannot be created, the error flag is set. However, if the reason is that the specified directory already exists, the error flag is not set.

In Example 48, the @CHECK("OFF") and @IFERROR THEN commands are used to check if the directory could be created.

@CHECK("OFF")

@SET DIRNEW = "ROUTINES" @MKDIR(DIRNEW) @CHECK("ON")

@IFERROR THEN GOTO XERROR

Example 48@MKDIR

4.1.38 @NEXT

Synopsis

@NEXT Var

Parameters

Var A loop counter variable

Description

This command indicates the end of a for-loop iteration. Each @FOR-TO command requires one @NEXT command in a later portion of the same command file.

Error Handling

The command has no effect on the error flag. The command fails if there is no matching @FOR-TO command in the file.

In Example 49, a for-loop is used to clear the variables VAR1 to VAR20. @FOR X = 1 TO 20

@FLUSHVAR(VAR{X}) @NEXT X

Example 49 @NEXT

Related Commands

@FOR-TO

4.1.39 @ONDISCONNECT

Synopsis

@ONDISCONNECT (Exp)

Parameters

Exp A label in the current script to which a jump is performed

Description

The function of the command is to enable continued script execution at the indicated label if the NE connection is disrupted. The purpose of this is to allow for graceful script termination at this type of event. The command is active until deactivated by "Normal Program Termination", by the @STOP command and by the stop options of the @PROMPT command. The command must appear early in the script as it can only take effect if the script interpreter has found the command.

Error Handling

The command has no effect on the error flag.

@ONDISCONNECT ("DISCON")

...

@LABEL DISCON

...

Example 50 @ONDISCONNECT

4.1.40 @ONTIMEOUT

Synopsis

@ONTIMEOUT(Exp,Exp1 ,Exp2)

Parameters

Exp A label in the current script to which a jump is performed

Exp1 The amount of time

Exp2 The three values "S" (for seconds, default), "M" (for minutes) and "H" (for hours)

Description

This function of this command is to limit the time OPS must wait for user interaction. The pop-up windows requiring user input will have a time-out (the value range is => 2 and has no upper limit) at which the script execution is continued at the indicated label after closing the pop-up window. This command

resembles the @ONDISCONNECT command in that it cannot take effect until found by the interpreter. The command is active until deactivated by "Normal Program Termination", by the @STOP command and by the stop option of the @PROMPT command.

If several @ONTIMEOUT commands are found by the interpreter while the script is run the latest one will be valid (by simply overriding the previous one).

Error Handling

The command has no effect on the error flag.

In Example 51, the user must respond to the pop-up input request within 5 minutes, otherwise the script execution will be terminated.

@ONTIMEOUT("TIMEOUT", 5, "M")

... @INPUT(Var1)

...

@LABEL TIMEOUT @STOP

@@SESSION SHUTDOWN

Example 51 @ONTIMEOUT

4.1.41 @OPENREAD

Synopsis

@OPENREAD(Exp)

Parameters

Exp The path name of the file to be opened for reading

Description

Opens the file specified by Exp for reading. Only one file can be open at a time. If a file is already open, this file will automatically be closed when the @OPENREAD command is executed.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

When a file has been opened, the @READ command can be used to read the first line in the file.

The @IFERROR THEN command can be used to test if the file has been opened.

Error Handling

The error flag is set if this command fails.

Example 52 shows how the @OPENREAD command is used to open a file whose first line is then read by the @READ command.

```
@SET SEARCH = "ALARM LIST"

@SET FILE="/users/sune/cha/cmdfile/cmdl.oz" @CHECK("OFF")

@COMMENT("Checking file", FILE, "for",SEARCH) @OPENREAD(FILE)

@CHECK("ON")

@IFERROR THEN GOTO XERROR @READ(CONT)

@COMMENT("The first READ line in the ",FILE,"file is",CONT) @LABEL XERROR

@COMMENT("The file could not be opened.")
```

Example 52 @OPENREAD

Related Commands

@READ

@WRITE

4.1.42 @ORDERED

Synopsis

@ORDERED(Exp)

Parameters

Exp Evaluated to either "ON" or "OFF". The parameter value "ON" is default.

Description

This command specifies when delayed responses are to be received.

@ORDERED("ON") allows delayed response to arrive at any time. The execution of the command file is suspended until the delayed response is received.

If @ORDERED("OFF") is in effect, delayed responses will not be displayed until the @LOGOF command is executed.

Error Handling

The command has no effect on the error flag.

The delayed response generated by the LABUP command in the following example is not received until the @LOGOF command at the end of the command file is executed.

@ORDERED("OFF"); LABUP;

... @LOGOF

Example 53 @ORDERED for delayed response with @LOGOF

Related Commands

@LOGOF

@LOGO

4.1.43 @PROMPT

Synopsis

@PROMPT

Parameters

None

Description

This command displays a pop-up window with the following choices:

Continue Continues the execution

Stop Stops the execution (like @STOP, Section 7.1.66 on page 112)

Error Handling

The command has no effect on the error flag.

@LOGOF

@COMMENT("The first phase is now completed.") @PROMPT

Example 54 @PROMPT

4.1.44 @READ

Synopsis

```
@READ({_}Var_ \[,LN({_}Exp_ )\])
```

Parameters

Var A variable name.

LN(Exp) An option to control the choice of record/line to be read.

Exp indicates the record/line number (>0) within the file. The meaning of "next line" will therefore be equal to Exp+ 1 if LN(Exp) was used in the previous @READ command.

Description

This command reads the next line (default) or a specified line from the file that has been opened by the @OPENREAD command into the variable Var. A line is considered all characters up to, but not including, the line boundary character. The line boundary characters, which consequently will not be stored in Var, are:

1. NEWLINE (= LINE FEED), (ASCII code $0A)
2. FORM FEED, (ASCII code $0C)
3. VERTICAL TAB, (ASCII code $0B)

Error Handling

If the command fails, the error flag is set.

Example 55 shows how the @READ command is used to read the two first and the fourth and fifth lines of a file that has been opened by the @OPENREAD command.

@SET SEARCH="ALARM LIST"

```
@SET FILE="/user/sune/cha/cmdfile/cmd1.oz" @CHECK("OFF")

@COMMENT("Checking file",FILE,"for",SEARCH) @OPENREAD(FILE)

@CHECK("ON")

@IFERROR THEN GOTO XERROR @READ(CONT1) @READ(CONT2)
@READ(CONT4,LN(4)) @READ(CONT5)

@COMMENT("The first line in the",FILE,"file is ",CONT1) @COMMENT("The
second line in the",FILE,"file is ",CONT2) @COMMENT("The fourth line in
the",FILE,"file is ",CONT4) @COMMENT("The fifth line in the",FILE,"file is
",CONT5)

...

@LABEL XERROR

@COMMENT("The file could not be opened.")
```

Example 55 @READ

Related Commands

@WRITE

@OPENREAD

4.1.45 @RENAME

Synopsis

@RENAME(Exp1 , Exp2)

Parameters

Exp1 The name of an existing file

Exp2 A new name for the file specified by Exp1, or a directory to which the file specified by Exp1 is to be moved

Description

This command renames an existing file, or moves the file to another directory and renames the file.

For example, the command can be used in the following ways:

Table 2 Table 4 @RENAME examples

Exp1	Exp2	Result
path/file1	file2	./file2
path1/file1	path2/file2	path2/file2
path1/file1	path2	path2/file1
file1	path/file2	path/file2
file1	file2	./file2
file1	path	path/file1

Note: Note: Files cannot be moved to other file systems.

Error Handling

If the command fails, the error flag is set.

In Example 56, a file is renamed and moved to the current directory.

@RENAME("/users/sune/cmdfiles/cmd1", "cmd1.oz.cmd")

Example 56 @RENAME

4.1.46 @RESET

Synopsis

@RESET

Parameters

None

Description

All run-time data except the subroutine stack is reset to default values. This has the same effect as pressing the Reset button in the OPS GUI.

Error Handling

The command has no effect on the error flag.

4.1.47 @RESTART

Synopsis

@RESTART

Parameters

None

Description

This command initiates a jump to the first line of the command file that is being executed. The subroutine stack is cleared and all other run-time information is retained.

Error Handling

The command has no effect on the error flag.

@COMMENT("An endless loop...") @DELAYSEC("10")

@RESTART

Example 57 @RESTART

4.1.48 @RETURN

Synopsis

@RETURN

Parameters

None

Description

This command indicates the end of a subroutine, which has been initiated by a @GOSUB command. @RETURN orders a jump back to the line following the @GOSUB command.

When @RETURN appears in a @FOR - loop within a subroutine, it will interrupt the @FOR - loop and return to the line following the @GOSUB command.

The @GOSUB and the @RETURN must be in the same file.

Error Handling

The command has no effect on the error flag.

In Example 58, the @RETURN command indicates the end of the subroutine XGESTATE. When this subroutine has been performed, the execution continues at the CHREP command.

```
@LABEL STEP1 @GOSUB XGESTATE CHREP;

:

:

@LABEL XGESTATE

@COMMENT(" ^ Checking external system state..") ALLIP;

DIREP:EMG=ALL; STRSP:R=ALL; EXEPP:EMG={EMG)0,EM=ALL;
STSTP:EMG={EMG}0,EMTS=ALL; @RETURN
```

Example 58@RETURN

Related Commands

@GOSUB

@LABEL

4.1.49 @SEND

Synopsis

@SEND(Exp {, Exp })

Parameters

Exp {, Exp} Any valid expression

Description

This command can be used to send arbitrary characters to the connected external system. @, can be sent by @SEND.

The expressions are concatenated and the resulting character sequence is sent and interpreted as if it were an MML command. The

execution waits for responses.

Note: Note: The command does not affect the invisible buffer mark.

Error Handling

The error flag is set if the command fails.

In Example 59 the @SEND command is used to switch between command mode and dialog mode.

```
! Enter dialogue mode IMLCT:SPG=0;

! Send dialogue command MCDVP;

! Interrupt dialogue mode(@) @SEND($40)

! Send command CACLP;

! Re-enter dialogue mode (EOT) @SEND(#04)

! End dialogue mode END;
```

Example 59 @SEND

See Example 59 also:

```
@IF time="NOT OK" THEN SEND("caclp;")
```

Example 59@SEND

Related Comments

@H

4.1.50 @SET

Synopsis

@SET Var = Exp

Parameters

Var A variable name

Exp Any expression

Description

This command sets the variable specified by Var to the value obtained when the expression Exp is evaluated.

Error Handling

The command has no effect on the error flag.

@SET PREV = 1

@SET SAD = PREV + 1

@SET NEXTONE = SAD / PREV @SET MODE = "FD"

@SET DEST = trim(copy(LINE,45,20))

Example 60@SE

Parameters

Exp Evaluated to either "ON" or "OFF"

Description

This function sets the errorflag. The parameter value "ON" is default.

Error Handling

The command sets the error flag.

4.1.51 @SETPHASE

Synopsis

@SETPHASE(Exp)

Parameters

None

Description

This command sets a value to an internal system variable indicating the current status of a script execution. Typical values could be "Started", "Waiting", "Loading", "Phase 2 ongoing" and so on.

Error Handling

The command has no effect on the error flag.

In Example 61, the the script execution progress status is set to "Loading new SW":

@SETPHASE("Loading new SW")

Example 61@SETPHASE

Related Commands

GETPHASE

4.1.52 @SETPROGRESS

Synopsis

@SETPROGRESS(Exp)

Parameters

Exp A value to which the progress counter will be set.

Description

This command sets the progress counter to the value indicated by Exp. The command is used when the progress cannot easily be represented by stepping the progress counter.

Error Handling

If the command fails the error flag is set.

In the script execution progress counter, the value of the variable is set tonumber_of_ne. In Example 68, the variable is used to indicate the number of network elements so far handled by the script.

@SETPROGRESS(number_of_ne)

Example 62@SETPROGRESS

Related Commands

1. @STEPPROGRESS
2. @SETTOTALCOUNT
3. GETPROGRESS
4. GETTOTALCOUNT

4.1.53 @SETREPLY

Synopsis

@SETREPLY(Exp1,Exp2)

Parameters

Exp1 Evaluated to either "OK" or "FAIL"

"OK" adds a normally not accepted response to the set of accepted responses

"FAIL" adds a normally accepted response to the set of not accepted responses

Exp2 A response

Description

In the OSS system, a response is either defined as accepted or as not accepted. The not accepted responses generate error messages and stop the file execution.

This command makes it possible to override the normal sets of accepted and not accepted responses from the external system.

A normally not accepted response, specified by Exp2 , is added to the set of accepted responses if Exp1 is set to OK.

In the same way, a normally accepted response, specified by Exp2 , is added to the set of not accepted responses if Exp1 is set to FAIL.

Each subsequent @SETREPLY with a given flag replaces the previous @SETREPLY for that flag. Accordingly, only one accepted response and one not accepted response can be overridden at a time.

A response that has been added to the set of accepted responses can be removed again by the @SETREPLY("OK"," ") command.

@SETREPLY can be used in conjunction with @WAITREPLY. If @SETREPLY("OK") has been executed, @WAITREPLY returns as soon as the OK string is found. @WAITREPLY does not wait for the complete response.

Similarly, if the @SETREPLY("FAIL") has been executed, @WAITREPLY returns with an error message as soon as the FAIL string is found in the partial response received.

Error Handling

The command has no effect on the error flag. This command will not stop the execution:

@SETREPLY("OK","NOT ACCEPTED") LABUP:SPG=0;

Example 63@SETREPLY

This command will stop the execution:

@SETREPLY("FAIL","ORDERED") LABUP;

Example 64@SETREPLY

Related Commands

@IFERROR THEN

@WAITREPLY

4.1.54 @SETTOTALCOUNT

Synopsis

@SETTOTALCOUNT(Exp)

Parameters

Exp The target value for the progress counter

Description

This command sets the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress.

Error Handling

If the command fails the error flag is set.

The target value for the progress counter is set to the value of the variable total_ne. In Example 71, the variable is used to indicate the number of network elements to be handled by the script.

@SETTOTALCOUNT(total_ne)

Example 65@SETTOTALCOUNT

Related Commands

@STEPPROGRESS

@SETPROGRESS

GETPROGRESS

GETTOTALCOUNT

4.1.55 @STEPPROGRESS

Synopsis

@STEPPROGRESS[(Exp)]

Parameters

Exp A value with which the progress counter is modified.

Default value is +1

Description

This command steps the script execution progress counter a number (positive or negative) of steps indicated by Exp.

Error Handling

If the command fails the error flag is set.

@STEPPROGRESS steps the script execution progress counter +1.

@STEPPROGRESS

Example 66@STEPPROGRESS

Related Commands

@SETPROGRESS

@SETTOTALCOUNT

GETPROGRESS

GETTOTALCOUNT

4.1.56 @STOP

Synopsis

@STOP

Parameters

None

Description

This command stops the execution of the command file with the run-time information preserved. The command is used to end the command file.

Error Handling

The command clears the error flag.

4.1.57 @VIEWFILE

Synopsis

@VIEWFILE(Exp)

Parameters

Exp The path name of the file that is to be displayed

Description

This command displays a text file in a pop-up window. The script execution is stopped until the user resumes execution by closing the pop-up window.

Error Handling

If the file name specified is invalid, or if the file cannot be opened, the error flag is set.

@INPUT(COMPSITE," Enter name of file for PCORP compare ")
@VIEWFILE(COMPSITE)

Example 67@VIEWFILE

4.1.58 @WAITFOR

Synopsis

@WAITFOR({_}Exp1_ \[\[, _Exp2_ \], _Exp3_ \])

Parameters

Exp1 A text string in a response

Exp2 A text string in a response (optional)

Exp3 The time that the command waits, expressed in number of seconds (optional)

Description

This command waits for strings to appear in a response from a delayed-response command. It is possible to wait for both one and two

response strings. When waiting for two response strings, both strings must appear for the wait to end. The time that the command waits can be limited by the Exp3 parameter. A timer indicating elapsed waiting time is shown in the 4th field from the left in the window footer.

The MML command that generates the response must be executed before the

@WAITFOR command.

The @WAITFOR command searches back to the latest occurrence of an @MARK or @LOGOF to check if the text strings already are received. Therefore it is possible to use @WAITFOR even in @ORDERED("ON") mode if desired.

Error Handling

The error flag is set if the specified time expires.

If the delayed response generated by the LASYP command in this example has not been received within 30 seconds, the error flag is set.

```
@CHECK("OFF") @ORDERED("OFF") LASYP:SPG=0,VNODE=A; @LOGOF
@WAITFOR("END", 30)
```

Example 68@WAITFOR

Related Command

@LOGOF

@ORDERED

4.1.59 @WAITREPLY

Synopsis

@WAITREPLY[(Exp)]

Parameters

Exp The time that the command waits, expressed in number of seconds.

Description

This command waits for a response. If desired, the time that the command waits can be limited by the Exp parameter. A timer indicating elapsed waiting time is shown in the 4th field from the left in the window footer.

The MML command that generates the response must be executed before the @WAITREPLY command.

@WAITREPLY can be used in conjunction with @SETREPLY. @SETREPLY is used to change the set of accepted responses or the set of not accepted responses.

When the @WAITREPLY command is executed, a search back to the latest occurrence of an @MARK or @LOGOF command is performed. The purpose of this search is to check if the text strings have already been received.

Error Handling

The error flag is set if the time expires. Example 77 checks if a response was received.

@CHECK("OFF") @ORDERED("OFF") LASYP:SPG=0,VNODE=A;
@WAITREPLY("30")

@IFERROR THEN GOSUB CHECKRE1

Example 69@WAITREPLY

Related Commands

@LOGOF

@ORDERED

@SETREPLY

@WAITFOR

4.1.60 @WINDEND

Synopsis

@WINDEND

Parameters

None

Description

This command closes the Comment window, which is opened by the @WINDOW command. It also closes the Input Output window, which is opened by the commands @DRAW, @FORM, and @MENU.

Error Handling

The command has no effect on the error flag.

Related Commands

@DRAW

@FORM

@MENU

@WINDOW

4.1.61 @WINDOW

Synopsis

@WINDOW

Parameters

None

Description

This command opens and clears the Comment window.

The Comment window is a pop-up window that consists of a scrollable text pane. When the window isopened, the size is approximately 8 lines, but it can be resized.

When @WINDOW is in effect, output created by the @COMMENT command is directed to the Comment window, and output is consequently not logged.

The Comment window is closed by the @WINDEND command.

Error Handling

The command has no effect on the error flag.

Related Commands

@WINDEND

@COMMENT

4.1.62 @WRITE

Synopsis

@WRITE({_}Exp0_ \[,LN({_}Exp_)\]\{, _Expn_ \})

Parameters

Exp0 The path name of the file to which one or more values are written.

Exp The optional LN(Exp) is record/line number. "LN" is used to distinguish Exp from Expn.

Expn An expression whose value is written to the file

Description

This command concatenates the values of the parameters Expn , appends the NEWLINE character (ASCII code 10), and writes the resulting string to the end of the file named Exp0.

The file is opened automatically. If the file exists, the text is appended to the file. However, if the LN(Exp) parameter is used, the text is written on a selected

line in the file. When Exp takes a value equal to or less than the current number of lines in the file the command scans the file to find the proper starting point for the desired record/line. Then the current contents are overwritten. If the length of the new contents differs from the current contents it can cause the number of records/lines to change as the file record separator is a new line character in a flat file. The PAD function (Section 7.2.32 on page 139) can be used by the script designer to circumvent this problem. If the file does not exist, a new file is created.

If the Exp0 parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

If the text parameters are omitted, a blank line is written to the file. The file is closed after the operation of this command.

Error Handling

The error flag is set if the write operation fails. For example, if LN(Exp) indicates a higher line number than the current number of lines + 1, the error flag is set.

@SET FILE = "~/log/SEQAR.LOG" @IF BTCNT = 0 THEN GOTO EXIT

@WRITE(FILE, "----------",DATE,"")

@SET BTPTR = 0 @LABEL SEQLOOP @INC(BTPTR)

@IF BTPTR > BTCNT THEN GOTO EXIT @SET DEV = CONCAT("BT-",BTPTR) @COMMENT(" Doing ",DEV,"..") SEQAR:DEV={DEV}; STDEP:DEV={DEV};

@SET ROUTE = COPY(REPLY(SCAN(DEV)),41,10) @@find route @COMMENT(" Route=^",ROUTE,"")

@WRITE(FILE,"BT-",BTPTR," ",ROUTE) @GOTO SEQLOOP

Example 70@WRITE

Example 71 shows how the LN(Exp) parameter can be used to change information in a file. Assume the file named fruits exists with the following contents:

apple fig banana orange

Example 72 Fruits

A command @WRITE("fruits",LN(2),"lemon") results in the following new file contents:

apple lemon nana orange

Note: Note: The third line banana has been cut short due to the fact that the new entry on line two is longer than the previous.

Now an additional command @WRITE("fruits",LN(2),"grapefruit") would completely destroy line three and therefore reduce the number of lines in the file:

apple grapefruit

orange Using the PAD function eliminates the problem of different line lengths. If the file above was originally created with

the use of the PAD function making all entries 10 characters long, the 'fig' could safely be changed to 'lemon' by the following command:

@WRITE("fruits",LN(2),PAD("lemon",10,"L")) resulting in the following file contents (the dots indicate empty spaces in the file): apple.....

lemon.....

banana....

orange. Then the line could safely be changed to grapefruit by the

following command:

@WRITE("fruits",LN(2),PAD("grapefruit",10,"L")) resulting in the following file contents:

apple.....

grapefruit banana....

orange....

Related Commands

@READ

4.1.63 @RMDIR

Synopsis

@RMDIR(Exp)

Parameters

Exp The path name of an empty directory to be deleted

Description

This command deletes the specified directory.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

Error Handling

This command fails, and the error flag is set, in the following cases:

• The user has no permission to remove the directory.

• The directory is not empty.

However, if the specified directory does not exist, the error flag is not set.

@SET PATH = ”/users/sune/cmdfiles/”

```
@SET DIROLD = "/users/sune/cmdfiles/asmfiles"
@CHECK("OFF")
@CHDIR(DIROLD)
@CHECK("ON")
@IFERROR THEN GOTO XERROR
@EXECUTE("rm —f asm_axe15")
@CHECK("OFF")
@CHDIR(PATH)
@RMDIR(DIROLD)
@CHECK("ON")
@IFERROR THEN GOTO XERROR
```

Example 73 @RMDIR

Related Commands

@MKDIR

4.1.64 @QUIT

Synopsis

@QUIT

Parameters

None

Description

If the @QUIT command is executed when the command file is run in the Output mode, without displaying the command file, the command file is terminated

and the window closes. If the command is executed in the Run mode, the execution will stop with the run-time information preserved.

Error Handling

The command clears the error flag.

Related Commands

@STOP

4.1.65 @REPORT

Synopsis

@REPORT(Exp)

Parameters

Exp Evaluated to either "ON" or "OFF"

Description

This command specifies whether commands can be sent to an external system of the type System 12 between the reception of the REPORT FOLLOWS and the LAST REPORT.

@REPORT("ON") allows the next command to be executed when the response REPORT FOLLOWS has been received.

@REPORT("OFF") means that no further commands can be executed until the printout LAST REPORT has been received

Error Handling

The command has no effect on the error flag.

In the @REPORT("OFF") command is executed to allow the meter reading information to be displayed before the user is asked whether another meter must be read.

```
@CLRSCR
@WINDOW
@DRAW(CUP,20,6,"SYSTEM12 METER READING PROGRAM")
@DELAYSEC(3)
@LABEL REPMETER
@SET DN1 = 0
```

```
@CLRSCR
@DRAW(BOX,5,5,21,7,CUP,7,6,"ENTER NUMBER")
@INPUT(DN1, "EITHER CUSTOMERS SERVICE NUMBER")
display-subsc-meter:DN1=K'{DN1};
@REPORT("OFF")
@INKEY(CHOICE, "DO YOU WISH TO READ ANOTHER METER [Y/N]?")
@IF CHOICE = "Y" THEN GOTO REPMETER
```

Example 74 @REPORT

4.1.66 @SPONTREP

Synopsis

@SPONTREP(Var , Exp1 , Exp2 {, Expn })

Parameters

Var The variable in which the return value is stored

Exp1 The name of the external system to which a subscription

is set up

Exp2{, Expn} The report ID, or IDs, to be received from the external

system

Description

This command sets up a subscription to spontaneous reports from an external system. The external system is specified by the Exp1 parameter. One or more report IDs are specified by the Exp2 {, Expn} parameters.

The report ID can be indicated as a numeral equivalent to the printout category (PRCA) or as a text string equivalent to all of or a part of a printout slogan.

For numerals an interval of printout categories can be specified by the Exp parameter. For text strings only a single slogan or part thereof can be specified by the Exp parameter. However, as several Exp parameters are allowed, several PRCA intervals or printout slogans can be specified in the command. A printout slogan must be indicated in the same case (upper/lower) as the printout (normally upper case). Wildcards in the text string are not allowed.

If the subscription has been successfully set up, Var is assigned the value 0. If the subscription could not be set up, Var is assigned the value 1. If the variable does not exist when the command is executed, it is automatically created.

A subscription remains active until the command file interpreter is terminated, or until the @SPONTREPOFF command is executed.

Error Handling

The command has no effect on the error flag.

Example 75 shows how subscriptions to all spontaneous printouts from AXE17 are set up and how subscriptions to the alarm "CP FAULT" and to

all spontaneous printouts with a slogan including ‘‘ANALYSIS ACTIVATED’’

from AXE19 are set up.

@SPONTREP(result_1,"AXE17","0-255")

@SPONTREP(result_3,"AXE19","CP FAULT", "ANALYSIS ACTIVATED")

Example 75 @SPONTREP

Related Commands

@SPONTREPOFF

4.1.67 @SPONTREPOFF

Synopsis

@SPONTREPOFF[([Var][Exp1 [,Exp2 {, Expn }]])]

Parameters

Var An optional variable in which the return value is stored

Exp1 The name of the external system for which a

spontaneous report subscription is to be terminated

Exp2 {, Expn} The report ID, or IDs, to be received from the external system

Description

This command terminates one or more spontaneous report subscriptions, or a part of a subscription, that has been set up with the @SPONTREP command. If no parameters are specified, all subscriptions are terminated. If only the Exp1

parameter is used to specify an external system, all subscriptions related to this external system are terminated. With the Exp2 parameter, it is possible to terminate certain printout categories or printout slogans or part thereof for the external system specified by Exp1.

In order for the command to be successful for printout slogans (or part thereof), the Exp2 parameter must exactly match the parameter specified in the @SPONTREP command. An interval of printout categories is considered as one subscription per printout category. It is therefore possible to remove one or more printout categories from an interval specified by the @SPONTREP command. If the subscription is successfully removed, the optional variable Var is assigned the value 0. If any of the specified subscriptions could not be removed, Var is assigned the value 1. If the variable does not exist when the command is executed, it is automatically created.

Error Handling

The command has no effect on the error flag.

In , all subscriptions to spontaneous reports from AXE17 are terminated. For AXE21 the subscription to spontaneous reports in printout category 38

(precessor alarms) and to the spontaneous report ”HB” is terminated.

@SPONTREPOFF("AXE17")

@SPONTREPOFF("AXE21", 38, "HB")

Example 76 @SPONTREPOFF

Related Commands

@SPONTREP

4.1.68 @MAIL

Synopsis

@MAIL(Exp1 ,Exp2 ,Exp3 |FILE(Exp4))

Parameters

Exp1 The mail address components

Exp2 The subject

Exp3 The message body of the e-mail

Exp4 The path name of a file which contains the message

body of the e-mail

Description

This command is used to send an email. Several addresses separated by comma or space can be included in Exp1.

Error Handling

The command has no effect on the error flag.

@SET s=GETSESSIONID()

@MAIL("eric@ericsson.com", "status report",

CONCAT("The present status for OPS session {s} is ",

GETPHASE()))

Example 77 @MAIL

4.2 OPS Script Function Descriptions

This chapter describes functions that can be used in OPS scripts.

4.2.1 ADVANCE

Synopsis

ADVANCE(Exp1 , Exp2)

Parameters

Exp1 An expression with a date and time value of the format YYMMDDHHMM

Exp2 An expression with a value of the format nnx where nn is a number within the range 1-99 and x one of the values W (week), D (day), H (hour), or M (minute)

Return Value

If the date and time specified by Exp1 is invalid, that is at least one of the properties year, month, date, and time is incorrect, the script execution is stopped.

If the date and time specified by Exp1 is correct, the value of Exp1 is increased by the value specified in Exp2.

Description

This function allows the date and time specified by Exp1 to be increased by a specified amount. This function is intended to be used for scheduled events. The result of advancing Exp1 always results in a time in the future. This is achieved by repeatedly advancing the parameter by the specified interval until the result is a time and date which has not yet passed.

The ten first characters in the parameter Exp1 are valid. If further characters are supplied, these characters are ignored. Thus, a date returned by the GETDATE function will be accepted. The return value is always 10 characters long.

In Example 80, the current file is scheduled to start executing at label L1 four hours after the current time.

```
@SET sched = ADVANCE(GETDATE(),"4H")
```

Example 78 ADVANCE

4.2.2 CENTRALDIR

Synopsis

CENTRALDIR()

Parameters

None

Return Value

The return value is the path to the central directory, shared in the OPS server environment variable OPS_CENTRALDIR.

Description

Centrally developed command files and data files shared among the users must be stored in a public directory.

The CENTRALDIR function returns the path to the central directory, which is defined by the system administrator.

The function appends a slash to the returned path if the path specified by

OPS_PRIVATEDIR does not end with a slash.

As Example 81 shows, the CENTRALDIR function can be used to include a centrally stored command file in the current command file.

```
@INCLUDE(CONCAT(CENTRALDIR(),"cmd_12.oz.cmd"))
```

Example 79 CENTRALDIR

4.2.3 CONCAT

Synopsis

CONCAT(Exp1 , Exp2 {, Expn })

Parameters

All parameters, Exp1 , Exp2 , and so on, can be set to any expression.

Return Value

The return value is a string made up of the specified parameters.

Description

This function concatenates (joins in one string) the parameters in the same order they are supplied.

Example 80 displays the string ABCDEFGHI.

```
@COMMENT(CONCAT("ABC","DEF","GHI"))
```

Example 80 CONCAT

4.2.4 COPY

Synopsis

COPY(Exp1 , Exp2 , Exp3)

Parameters

Exp1 A text string

Exp2 A numeric value that indicates the starting position for the copy operation in the Exp1 string

Exp3 A numeric value that indicates the number of characters from the Exp2 position onwards to be copied

Return Value

The return value is a substring that has been copied from the Exp1 parameter.

Description

This function copies a part of the text string specified by Exp1. The starting position for the copy operation in the Exp1 text string is specified by the Exp2 parameter. The length of the string to be copied, from the starting position onwards, is specified by the Exp3 parameter.

Due to the impact on the number of characters in text strings by the STYLE function, it is recommended that the COPY function not operate on expressions containing the STYLE function

@SET LDATE = GETDATE() @SET DATE = COPY(LDATE,1,6) @SET TIME = COPY(LDATE,7,4) @SET DAY = COPY(LDATE,13,3)

@COMMENT("Check system time and date..") @COMMENT("DATE=", DATE, " DAY=", DAY, " TIME=", TIME)

Example 81 COPY

4.2.5 DECIMAL

Synopsis

DECIMAL(Exp)

Parameters

Exp An expression with a hexadecimal value

Return Value

The return value is the decimal value corresponding to the specified hexadecimal value.

Description

This function converts a specified hexadecimal value to its corresponding decimal value.

The following example displays the decimal value 248 on the screen.

@SET D = DECIMAL("F8") @COMMENT(D)

Example 82 DECIMAL

4.2.6 DISKFREE

Synopsis

DISKFREE(\[_Exp_ \])

Parameters

Exp A path or a file system name (optional)

Return Value

The return value is the amount of free disk space, in kilobytes, in the specified file system.

Description

This function returns the amount of free disk space, expressed in kilobytes, in the specified file system. If no parameter is supplied, the free space of the file system on which the current working directory resides is returned.

@SET F = DISKFREE()

@COMMENT("Free space =" , F, "kilobytes")

Example 83 DISKFREE

4.2.7 GETDATE

Synopsis

GETDATE([Var])

Parameters

Var Stores the number of the day of the week

Return Value

The return value is the system date and time in the format YYMMDDHHMMSSDDD.

Description

The function returns the system date, time, and day of the week. The information is returned as a text string, YYMMDDHHMMSSDDD. The positions in the text string have the following meanings:

YY Year

MM Month

DD Day

HH Hour

MM Minute

SS Second

DDD Day of the week (SUN, MON, TUE, WED, THU, FRI, or SAT)

If the optional parameter Var is specified, the number of the day of the week is returned and stored in this variable. The value 0 means Sunday, 1 means Monday, and so on.

In Example 84, the GETDATE function is used to retrieve the system date, time, and day of the week. The COPY function is then used to copy the date to one parameter, the time to another parameter, and the day of the week to yet another parameter.

```
@LABEL CHECDATE

@COMMENT("Checking system time, date, and day of the week")

@SET LDATE = GETDATE()

@SET DATE = COPY(LDATE, 1, 6)

@SET TIME = COPY(LDATE, 7 ,4)

@SET DAY = COPY(LDATE, 13, 3)
@COMMENT("DATE=",DATE,"DAY=",DAY,"TIME=",TIME)
```

Example 84 GETDATE

Part of the return value can be stored directly in a variable by using the COPY function. In the following example, the current hour is stored in a variable. @SET curhour = COPY(GETDATE(), 7,2)

4.2.8 GETDEST

Synopsis

GETDEST()

Parameters

None

Return Value

The return value is the string used when connecting to the external system.

Description

This function returns the string used at the time of connecting to the external system to which the connection is currently established. If no external system is connected, an empty string is returned.

Example 8

If a connection was established with @CONNECT("Msc15"), GETDEST()

returns the string "Msc15".

In the same way, @CONNECT("NE=abc,CPNAME=CP1") results in the return string "NE=abc,CPNAME=CP1".

Related Commands

@CONNECT

@DISCONNECT

4.2.9 GETDIR

Synopsis

GETDIR()

Parameters

None

Return Value

The return value is the path of the current directory.

Description

This command retrieves the path of the current directory, for example

/usr/home/fred/cha/cmdfile/ . When a script starts, the default and recommended current directory is /cha/cmdfile under the user's home directory.

The returned path ends with a slash.

In Example 90, the current directory is retrieved and displayed on the screen.

@COMMENT("The current UNIX directory is:",GETDIR())

Example 85 GETDIR

4.2.10 GETFILELENGTH

Synopsis

GETFILELENGTH(Exp)

Parameters

Exp The path name of a file for which the length is to be checked

Return Value

The return value is the length of the file, in terms of the number of lines in the file.

Description

This function counts the number of lines in a file and presents the number in the return value.

@SET file=CONCAT(Path of directory, FILE1")

@READ(contents,LN(GETFILELENGTH(file)))

Example 86 GETFILELENGTH

Error Handling

The error flag is set if this command fails.

Related Commands

@READ

@WRITE

4.2.11 GETIOTYPE

Synopsis

GETIOSTYPE(Exp)

Parameters

Exp The name of the external system

Return Value

If the external system is found, the IO node type is returned (APG or IOG), otherwise an empty string is returned.

Description

This function returns the IO node type of an external system.

@COMMENT (GETIOTYPE("STP50"))

Example 87 GETIOTYPE

4.2.12 GETLOG

Synopsis

GETLOG([Exp])

Parameters

Exp The log file number, default number=1

Return Value

If a log file is open, the full path and file name of this log file is returned, otherwise an empty string is returned. The use of a log file number provides the possibility to have several log sessions in parallel.

Description

This function allows an OPS script to determine whether a log file is open. If a log file is open, the function returns the full path of this log file. If several

log files are in use, the status must be checked for every log file by using the log file number.

The function allows the OPS script to save the log file name, open a new log file, and log its activities in the new log file, as the following example shows. When the task is completed, the original log file can be reopened.

```
@SET OLDLOG = GETLOG()

@LOG("cmd_file.log")

:

:

@CLOSE @LOG(OLDLOG)
```

Example 88 GETLOG

Related Commands

@LOG

4.2.13 GETMODE

Synopsis

GETMODE()

Parameters

None

Return Value

The return value is text string that specifies the external system type of the currently connected external system, which is always "AXE" in OPS.

Description

This function returns the external system type of the currently connected external system. As OPS is only used with AXE-based systems, the return string is always "AXE".

If no connection is established, an empty string is returned.

4.2.14 GETSESSIONID

Synopsis

GETSESSIONID()

Parameters

None

Return Value

The return value is the identity of the session.

Description

Each time OPS is started (from the work space menu or with a shell command), a session number is allocated and displayed in the window header. This function returns the current OPS session.

4.2.15 HEX

Synopsis

HEX({_}Exp1_ \[, _Exp2_ \])

Parameters

Exp1 A positive or negative decimal number to be converted to a hexadecimal number

Exp2 The number of characters required for the hexadecimal number (optional)

Return Value

The return value is the hexadecimal value corresponding to the decimal value specified by the parameter Exp1. The number of characters in the returned string can be specified by the Exp2 parameter. The least significant characters are returned.

By default, the number of characters needed to reflect the converted value, up to eight characters, is returned.

Description

This function returns the hexadecimal value of the hexadecimal number specified by the Exp1 parameter.

Example 100 displays the decimal value FE64 on the screen.

```
@SET DECI_NUM = 65124

@SET HEX_NUM = HEX(DECI_NUM) @COMMENT(HEX_NUM)
```

Example 89 HEX

4.2.16 LABELEXIST

Synopsis

LABELEXIST(Exp)

Parameters

Exp An assumed label in the current script

Return Value

The function will return the value 0, 1 or 2 where 2 indicates that the label occurs at least two times in the script. The value 0 indicates that the label does not occur in the script.

Description

This function checks whether the indicated label exists in the current OPS script.

@SET line=SCAN("NOT ACCEPTED") @IF line=0 THEN GOTO OK

@@Extract the fault code from the response printout: @SET fc=TRIM(COPY(REPLY(line+1),12,4))

@@Check if special action is provided for the fault code: @IF LABELEXIST("FC{fc}")=0 THEN GOTO OTHER-FC

@GOTO FC{fc}

@label OTHER-FC

@comment("OTHER-FC has been reached")

Example 90 LABELEXIST

4.2.17 LENGTH

Synopsis

LENGTH(Exp)

Parameters

Exp Any expression

Return Value

The return value is the length of the string specified by Exp.

Description

This function determines the length of a specified text string.

@IF LENGTH(EMG) <> 4 THEN GOSUB GETEMG

Example 91 LENGTH

4.2.18 LOWCASE

Synopsis

LOWCASE(Exp)

Parameters

Exp A text that is to be converted into lowercase

Return Value

The return value is the text string Exp converted into lowercase.

Description

When a text string Exp is given as input, this function returns the text string in lowercase.

Related Commands

UPCASE

4.2.19 PAD

Synopsis

PAD({_}Exp1_ , _Exp2_ \[{_},Exp3_ \])

Parameters

Exp1 The text string to be printed

Exp2 The column width

Exp3 It indicates whether padding will result in left (default if Exp3 is omitted) adjustment ("L"), right adjustment ("R") or centred adjustment ("C")

Return Value

The return value is a copy of the string specified by Exp1. The column width is defined by Exp2 and the location of the string is determined by Exp3.

If length of Exp1 is more than indicated by Exp2 then Exp1 is cut at the end, regardless of Exp3. PAD("ABCD",2, "R") therefore results in "AB".

Description

This function produces a padded text string with a selected adjustment and a fixed length for use in files or in windows to get a nice layout.

It is recommended that the function operates on the @COMMENT, @DRAW, @INPUT, @INKEY, and @WRITE commands. However, there is no restriction for this function to be used with other commands

This function recommends a fixed width font to be used. Due to the impact on the number of characters in text strings by the STYLE function, it is

recommended that the PAD function not operate on expressions containing the

STYLE function.

@SET file1="AB" @SET volume="1234"

@DRAW(CUP,3,2,CONCAT(STYLE(UNDERLINE,PAD("FILE",6,"C"))," "," ", STYLE(UNDERLINE,PAD("VOLUME",8,"C"))))

```
@DRAW(CUP,3,3,CONCAT(PAD(file1,6,"C")," "," ",
STYLE(BOLD,PAD(volume,8,"R"))))
```

Example 92 PAD

Example 92 produces the following text if file1=AB and volume=1234 :

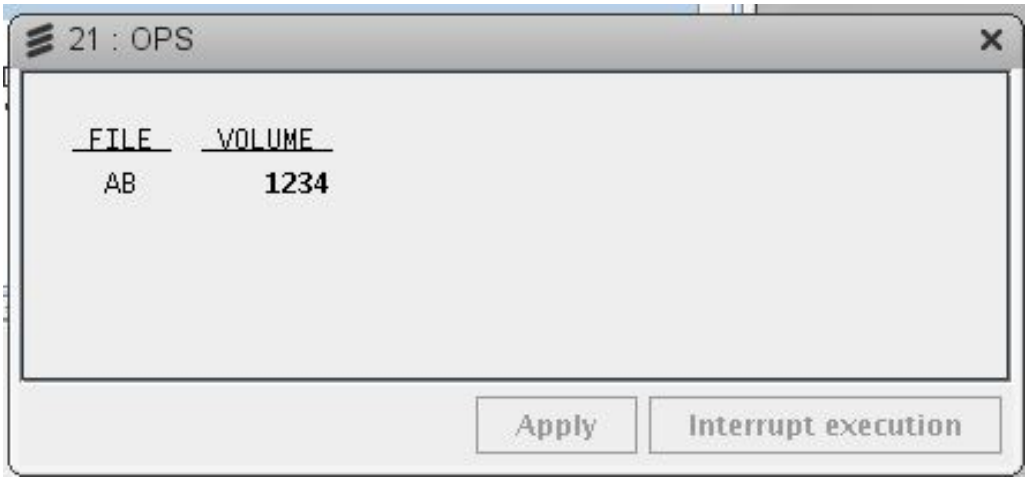

Figure 3 Figure 7 Example of PAD Command

Related Commands

See also the example of the use of the @WRITE command.

4.2.20 POS

Synopsis

POS(Exp1 , Exp2)

Parameters

Exp1 The text searched for

Exp2 The text that is scanned to find the text specified by Exp1

Return Value

The return value is the start position of the string Exp1 within the string Exp2. If the text specified by Exp1 is not found, 0 is returned.

Description

This function returns the start position of one specified string, Exp1 , within another specified string, Exp2.

In Example 93, the POS function is used to specify the start position for a copy operation.

The last received response is scanned for the characters "LI- " by the SCAN

function. The result of the scanning operation is assigned to the variable LINE.

If "LI- " is not found, the command file returns from the subroutine. If "LI- " is found, a copy of a part of the response line that includes "LI- " is created.

```
@SET LINE = SCAN("LI-") @IF LINE = 0 THEN RETURN

@SET LI_NUMBER = COPY(REPLY(LINE), POS("LI-",REPLY(LINE)) + 3,5)
```

Example 93 POS

4.2.21 REGRESS

Synopsis

REGRESS(Exp1 , Exp2)

Parameters

Exp1 An expression with a date and time value of the format YYMMDDHHMM

Exp2 An expression with a value of the format nnx where nn is a number within the range 1-99 and x one of the values W (week), D (day), H (hour), or M (minute)

Return Value

If the date and time specified by Exp1 is invalid, that is, at least one of the properties year, month, date, and time is incorrect, the script execution is stopped.

If the date and time specified by Exp1 is correct, the value of Exp1 is decreased by the value specified in Exp2.

Description

This function allows the date and time specified by Exp1 to be decreased by a specified amount. This function reverses to the function of ADVANCE . This function is intended to be used for scheduled events.

The result of regressing Exp1 always results in a time in the past. This is achieved by repeatedly regressing the parameter by the specified interval until the result is a time and date which has already passed.

The ten first characters in the parameter Exp1 are valid. If further characters are supplied, these characters are ignored. Thus, a date returned by the GETDATE function is accepted. The return value is always 10 characters long.

@DRAW(CUP,2,2,"ROUTINE TEST RESULT", CUP,2,4,"TEST TIME RESULT LAST TEST TIME", CUP,2,5,COPY(GETDATE(),1,10),

CUP,14,5,result, CUP,23,5,REGRESS(GETDATE(),"2W"))

Example 94 REGRESS

4.2.22 REPLY

Synopsis

REPLY(Exp)

Parameters

Exp A line number (>0) relative to the invisible mark in the buffer

Return Value

The return value includes all characters on the response line specified by Exp.

Description

This function returns all text on a response line that is specified by the Exp

parameter. The Exp parameter refers to the invisible mark in the buffer.

When an MML command is sent, the invisible buffer mark is automatically set at the line of the command. The invisible buffer mark can also be set by the @MARK command.

In Example 95 the REPLY function is used to specify a line that is to be copied by the COPY function.

@LABEL CHECKFCODE @SET RESULT = 1 @SET RESULT = 1

@SET LINE = SCAN("FCODE",POS) @@ check for FCODE 0! @SET FC = COPY(REPLY(LINE + 1),POS,3)

@IF FC = 0 THEN RETURN @SET RESULT = 0

@INKEY (CHOICE, "Command failed, FCODE", FC, ":press any key...")

@RETURN

Example 95 REPLY

4.2.23 REPLYLEN

Synopsis

REPLYLEN()

Parameters

None

Return Value

The return value is the number of lines displayed since the invisible buffer mark was set.

Description

This function determines the number of lines that have been displayed since the last MML command was executed or since the last @MARK command was executed.

When an MML command is sent, the invisible buffer mark is automatically set at the line of the command. The invisible buffer mark can also be set by the @MARK command.

ALSTP;

@COMMENT("The reply was ", replylen(), "lines long") @SET line2 = reply(2)

@COMMENT("Line 2 was ", line2)

Example 96 REPLYLEN

4.2.24 SAVEDIR

Synopsis

SAVEDIR()

Parameters

None

Return Value

The function returns the value of the OPS server environment variable OPS_RESPONSEDIR, which specifies the private response file directory of the user.

Description

Each user has a private response file directory.

The function SAVEDIR returns the value of the OPS_RESPONSEDIR OPS server environment variable. This variable specifies the response file directory of

the current user.

The value returned by SAVEDIR can be used for constructing sensible paths for commands like @LOG and @WRITE.

The SAVEDIR function appends a slash to the returned path if the path specified by OPS_RESPONSEDIR does not end with a slash.

@SET DIRSAVE = SAVEDIR()

@COMMENT("The value of CHA_RESPONSE variable is", DIRSAVE)

Example 97 SAVEDIR

4.2.25 SCAN

Synopsis

SCAN({_}Exp_ \[, _Var_ \])

Parameters

Exp String of text to be searched for in the last MML command response (case sensitive)

Var The position in the string of Exp on the line (optional)

Return Value

The return value is a numeric value that indicates the number of the line within the response that contains the Exp string.

If the Exp string is not found, the value 0 is returned.

Description

This function scans the response of the last executed MML command (and any other subsequent contents of the command File Output Area) for the string specified by the Exp parameter. The string search is case sensitive and it starts right after the line pointed at by the invisible buffer mark. The command returns the number of the line, with reference to the invisible buffer mark, that includes the string. If the string is not found, the value 0 is returned.

If the invisible buffer mark has not been shifted by the @MARK command, it is located at the line of the MML command.

The position at which the Exp string starts on the line is stored in the optional variable Var.

```
@LABEL RELTES1

DIVAP:BNR=5, BAN=22, VLN=H0;

@IF SCAN("00FF") <> 0 THEN GOTO RELTES2
```

Example 98 SCAN

4.2.26 STYLE

Synopsis

STYLE(Style, Exp)

Parameters

Style The value of text style such as PLAIN (default), BOLD, UNDERLINE or ITALIC

Exp The text string

Return Value

A styled text string.

Description

This function controls the text style (plain, bold, underline, italic) in pop-up and input output windows (invoked by the @DRAW, @INPUT, and @INKEY commands).

See the comments on the use of STYLE in the description of functions COPY

and PAD.

STYLE operates on the subsequent text string. This will thus allow for different styles on the different text strings in the text message in the window, for example by using the existing CONCAT function.

@INPUT(command, CONCAT("Could you ",STYLE(BOLD,"Please"), " enter some command!"))

Example 99 STYLE

The command produces the following text in a pop-up input request window:

Figure 4 Figure 8 Example of the Style Command

4.2.26 TRIM

Synopsis

TRIM(Exp)

Parameters

Exp Evaluated to a text string

Return Value

The return value is a copy of the string specified by Exp , with the only difference that all leading and trailing "whitespace" is removed.

Description

This command removes all leading and trailing characters described as "whitespace" from the string specified by Exp. "Whitespace" is defined as any sequence of the following characters:

1. TAB
2. SPACE
3. CARRIAGE RETURN
4. NEWLINE
5. FORM FEED
6. VERTICAL TAB

@SET DEST = TRIM(COPY(LINE,45,20)) @SET MODE = TRIM(COPY(LINE,42,3))Example 116TRIM

4.2.27 UPCASE

Synopsis

UPCASE(Exp)

Parameters

Exp A text string that is to be converted into uppercase

Return Value

The return value is the text string Exp converted into uppercase.

Description

When a text string Exp is given as input, this function returns the text string in uppercase.

Related Commands

LOWCASE

4.2.28 VAREXIST

Synopsis

VAREXIST(Var)

Parameters

Var The name of a variable

Return Value

If the variable does not exist, the value 0 is returned. If the variable does exist, the value 1 is returned.

Description

This function allows a command file to determine whether a certain variable exists.

The function can be used to check if a variable has been set earlier in the command file.

@IF VAREXIST(VAR1) = 0 THEN SET VAR1 = 9600

Example 100 VAREXIST

4.2.29 VERSION

Synopsis

VERSION()

Parameters

None

Return Value

The return value of this function is a string indicating the product number and revision of Operations Procedure Support (OPS).

Description

This function allows a command file to determine the product number and revision of OPS that is currently used.

An example of a printout generated by Example 118 is "You are currently running version APR 901 134 R6A".

@COMMENT("You are currently running version ", VERSION())

output: You are currently running version ERICopsparser_CXP9033907-1.2.3-1.noarch

Example101 VERSION

4.2.30 PRIVATEDIR()

Synopsis

PRIVATEDIR()

Parameters

None

Return Value

The return value is the value of the OPS server environment variable OPS_PRIVATEDIR, which specifies the private command file directory of the user.

Description

Each user has a private command file directory.

The PRIVATEDIR function returns the value of the OPS server environment variable OPS_PRIVATEDIR, which specifies the private command file directory

of the user. This OPS server environment variable usually contains the path /cha/cmdfile, that is /home/<user>/cha/cmdfile.

The function appends a slash to the returned path if the path specified by OPS_PRIVATEDIR does not end with a slash.

As Example 102 shows, the PRIVATEDIR function can be used to include a file

that is stored in a private directory.

```
@INCLUDE(CONCAT(PRIVATEDIR(),"myfile_8.oz.cmd"))
```

Example 102 PRIVATEDIR

4.2.31 GETPHASE

Synopsis

GETPHASE()

Parameters

None

Return Value

The return value is the value of the script progress status indication.

Description

This function is used to retrieve the current progress status set by the command @SETPHASE.

See the example of the @MAIL command.

Related Commands

@SETPHASE

4.2.32 GETPROGRESS

Synopsis

GETPROGRESS()

Parameters

None

Return Value

The return value is the value of the script execution progress counter.

Description

This function is used to retrieve the value of the script execution progress counter set by the commands @SETPROGRESS and @STEPPROGRESS.

```
@COMMENT("So far we have reached script
execution level ",GETPROGRESS())
```

Example 103 GETPROGRESS

Related Commands

• @SETPROGRESS

• **@STEPPROGRESS**

• **@SETTOTALCOUNT**

• **GETTOTALCOUNT**

4.2.33 GETTOTALCOUNT

Synopsis

GETTOTALCOUNT()

Parameters

None

Return Value

The return value is a value to use to retrieve the target value for the progress

counter.

Description

This function is used to retrieve the target value for the progress counter, that is a value that the progress counter must be compared to for assessing the script execution progress. The target value is set by the command @SETTOTALCOUNT.

@COMMENT("So far we have done ",

100*GETPROGRESS()/GETTOTALCOUNT(),"% of the job")

Example 104 GETTOTALCOUNT

Related Commands

• GETPROGRESS

• @SETPROGRESS

• @STEPPROGRESS

• @SETTOTALCOUNT

4.2.34 SELFILE

Synopsis

SELFILE(Exp1 {, Expn })

Parameters

Exp1 A directory mask

Expn The prompt to display when a file name is asked for

Return Value

The function returns the file name specified by the user. If no file is chosen, the directory is returned.

Description

This function displays a file browser and prompts the user to enter a file name. The directory mask Exp1 can contain a directory and a file component (for example, /users/fred/*ozt). For the file component, the wild-card characters (*) and (?) are supported.

If the directory component is specified, the file browser shows the specified directory when the file browser appears. If there is no directory component specified, or the specified directory cannot be accessed, the current directory is shown. In both cases, the user can use the file browser to navigate through the file system.

4.2.35 GETERRORFLAG

Synopsis

GETERRORFLAG()

Parameters

None

Return Value

The return value is the error flag.

Description

This function indicates whether the error flag is set (1) or not (0).

The error flag is stored for later use.

```
@SET a =GETERRORFLAG()
...
@IF a=1 THEN GOTO INVALIDAFTERALL
```

Example 105 GETERRORFLAG

4.2.36 GETEXCHHDR

Synopsis

GETEXCHHDR()

Parameters

None

Return Value

The return value is an empty string.

Description

The exchange header of the connected exchange is not available for OPS. The function fveny is not supported in OPS

4.2.37 GETMARK

Synopsis

GETMARK ()

Parameters

None

Return Value

The return value is the line number in the responding buffer area.

Description

This function retrieves the line number being pointed at by the invisible buffer mark within the Command File Output Area. The primary use of this function is for script design and debugging.

@COMMENT (GETMARK())

Example 106 GETMARK

Related Comments

@MARK

4.2.38 GETPROTOCOLTYPE

Synopsis

GETPROTOCOLTYPE(Exp)

Parameters

Exp The name of the external system

Return Value

If the external system is found, the protocol type is returned (MTP_IOG, V24_IOG, SSH_APG40 or TELNET_APG40), otherwise,an empty string is returned.

Description

This function returns the protocol type used for communication with the external system.

@COMMENT (GETPROTOCOLTYPE("STP50"))

Example 107 GETPROTOCOLTYPE

4.2.39 CHECKCHILDSESSION

Synopsis

CHECKCHILDSESSION(Exp1)

Parameters

Exp1 The session id of the child session to check

Return Value

The execution state of the child session. The value can be one of the following:

• RUNNING for a child session which is executing

• FINISHED for a child session that has successfully finished executing

• STOPPED for a child session that has been stopped (from the GUI) by the user

• FAILED for a child session that has been terminated either due to error conditions or due to execution of the TERMINATECHILDSESSION command

• NOT_STARTED for a child session which has not yet started to execute its script

• INTERRUPTED for a child session which is being interrupted

If the command fails the OPS error flag is set and the return value is set to "" (empty string)

Description

This function checks the execution state of a child session.

Example 108 shows how to use the command to implement “thread join”

functionality.

```
@LABEL JOIN_LOOP
@SET STATUS = CHECKCHILDSESSION(CHILD_SESSION_ID)
@IF STATUS = "RUNNING" THEN GOTO JOIN_LOOP
```

Example 108 CHECKCHILDSESSION

4.2.40 CREATECHILDSESSION

Synopsis

CREATECHILDSESSION(Exp1 {, Expn })

Parameters

Exp1 A text string specifying the name of the script to execute as a child. If the script name specified is a relative path it will start from the directory containing the parent script. An absolute path will start from the root directory.

Expn Variable or variables in the child session which are to be readable from the parent script using the READVARIABLE function.

Return Value

The session id of the child session. If the creation fails the OPS error flag is set and the return value is set to 0.

Note: Note: The number of parallelly running sessions is limited to 400. Sessions and child sessions are counted together. Any attempt to create a new child session over the limit fails.

Description

This function allocates resources for a child session. It does however not start execution of the child session. The child session is set to execute the script specified by Exp1. The child script variables

Expn are readable by using the

READVARIABLE function.

The command in creates a child session running the script script1.ccf and

sets the variables var1 and var2 as readable from the parent script.

@SET CHILD_SESSION_ID = CREATECHILDSESSION("script1.ccf", "var1", "var2")

@IFERROR THEN

@COMMENT("Child creation failed")

@ENDIF

Example 109 CREATECHILDSESSION

4.2.41 READVARIABLE

Synopsis

READVARIABLE(Exp1, Exp2)

Parameters

Exp1 The session id of the child session from which to read

the variable.

Exp2 The name of the variable to be read.

Return Value

The value of the read variable. If the function fails the OPS error flag is set and the return value is set to "" (empty string).

Description

This function reads the value of a variable in a child session.

The following command reads the value of the variable var in a child session with session id CHILD_SESSION_ID. A subscription to the variable must have been set up when the session was created using the CREATECHILDSESSION function.

```
@SET VAR_VALUE = READVARIABLE(CHILD_SESSION_ID, "var")
@IFERROR THEN
@COMMENT("Failed to read variable value")
@ENDIF
```

Example 110 READVARIABLE

4.2.42 STARTCHILDSESSION

Synopsis

STARTCHILDSESSION(Exp1 {, Varn, Valn })

Parameters

Exp1 The session id of the child session to be started

Varn, Valn Pair or pairs of variable name and value to be set before

the script starts.

Return Value

There are two possible return values with the following meanings:

0 The function has failed and the OPS error flag is set.

1 The function has successfully started the session.

Description

This function starts execution of a child session which has been created by the CREATECHILDSESSION function.

The following command starts execution of a child session with session id CHILD_SESSION_ID and sets the variables var1 to 11 and var2 to 22 before starting.

@SET RESULT = STARTCHILDSESSION(CHILD_SESSION_ID, "var1", "11", "var2", "22")

@IFERROR THEN

@COMMENT("Failed to start session")

@ENDIF

Example 111 STARTCHILDSESSION

After execution of a child session is started the corresponding session number can be seen in the “Children” window of the GUI (Page 148). The session number disappears from the window once the child script has terminated.

4.2.43 TERMINATECHILDSESSION

Synopsis

TERMINATECHILDSESSION(Exp1)

Parameters

Exp1 The session id of the child session to terminate

Return Value

There are two possible return values with the following meanings:

0 The function has failed and the OPS error flag is set.

1 The function has successfully terminated the session.

Description

This function terminates a child session.

The following command terminates a child session with session id

CHILD_SESSION_ID.

```
@SET RESULT = TERMINATECHILDSESSION(CHILD_SESSION_ID)
@IFERROR THEN
@COMMENT("Failed to terminate child session")
@ENDIF
```

Example 112 TERMINATECHILDSESSION

4.3 FIOL Command Descriptions

This chapter describes FIOL commands. The FIOL commands have been retained within the OPS script language to guarantee backward compatibility.

FIOL commands can be used in the same command files as the OPS script commands described above.

Note: Note: A FIOL command can have the same effect as a certain command in

the OPS script language.

Also note that parameters are not written within parentheses and that they must

not have quotation marks. There must be a space between the command

and the parameter.

4.3.1 Remark!

Synopsis

Parameters

None

Description

When an ! command occurs on a command line, the text after the ! character is treated as a remark. (The only exception is the !$$$$! command.) The remark will not be displayed on the screen.

The ! command can also be used on the same line as an MML command.

The remark can be written between two exclamation marks and appended after the semicolon, as the following example shows:

CACLP; !Displays the current date and time!

The remark can also be written between the MML command and the semicolon. Then only one exclamation mark in front of the remark is required:

CACLP !Displays the current date and time;

The remark will be logged together with the MML command.

Error Handling

The command has no effect on the error flag.

4.3.2 Display in Command File Output Area /* .... */

Synopsis

/Char_sequence /

Parameters

None

Description

All characters in between /* and / are displayed in the *Command File Output area of the user interface. A comment cannot be nested nor span multiple lines. Only one comment is allowed on one line.

Error Handling

The command has no effect on the error flag.

Related Command

@COMMENT

4.3.3 @/W1, W2, .../

Synopsis

@/Char_sequence1 , Char_sequence2 , ... Char_sequencex /

Parameters

Char_sequence A character sequence that must occur in the next immediate response

Description

This command checks that the next immediate response from the external system contains the character sequences Char_sequence1 , Char_sequence2 , and so on, in correct order. The command must be executed before the MML command whose response is to be tested.

A maximum of 80 characters are allowed between the slashes. A character sequence must not contain commas (,) and must not be within quotes.

The following message is displayed if no match occurs:

** RESPONSE TEST FAILED WITH STRING: Char_sequencex

The command file execution stops.

However, if the command @R+ has been executed before the @/W1,W2.../ command, the execution continues even if no match occurs. This override can be reset by the @R- command.

Error Handling

The command has no effect on the error flag.

Related Commands

@R

4.3.4 @A

Synopsis

@A +

@A -

Parameters

+ Turns on automatic confirmation of MML commands by

a semicolon (;)

- Turns off automatic confirmation of MML commands by

a semicolon (;)

Note: Note: As the OPS script language expects commands to be automatically confirmed, the @A - setting can cause conflicts.

Description

This command turns automatic confirmation of MML commands by a semicolon

(;) on or off.

@A + is the default value.

Error Handling

The command has no effect on the error flag.

```
@AIOROL:
CANCEL;
;!Semicolon needed!
@A+
IOROL:CANCEL;!Semicolon not needed!
```

Example 113 @A

4.3.5 @C

Synopsis

@C

Parameters

None

Description

This command closes the log file. The command has the same effect as the command @CLOSE or @CLOSE(1).

Error Handling

The command has no effect on the error flag.

Related Commands

- @CLOSE
- @L
- @LOG

4.3.6 @E

Synopsis

@E

Parameters

None

Description

This command has the same effect as the command @LOGOF.

If @ORDERED("OFF") is in effect, delayed responses will not be displayed until the @E command, or the @LOGOF command, is executed.

Thus, it is possible to display delayed responses after the command file has been executed.

Error Handling

The command has no effect on the error flag.

Related Commands

- @LOGOF
- @ORDERED

4.3.7 @I

Synopsis

@I Char_sequence

Note: Note: A space is required between the command and the parameter.

Parameters

Char_sequence The path name of a command file to be included in the execution

Description

This command calls another command file and executes this file from the beginning to the end, or until a !$$$$! command is encountered. Then, the

execution continues on the line immediately following the @I command in the original file.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

Error Handling

The error flag is set if the file to be included is not accessible.

See the example for @INCLUDE.

Related Commands

- @INCLUDE
- !$$$$!

4.3.8 @L

Synopsis

@L Char_sequence

Note: Note: A space is required between the command and the parameter.

Parameters

Char_sequence The path name of the log file to be used

Description

This command specifies a file on which communication is logged. The command has the same effect as the command @LOG(Exp)or @LOG(Exp,1). When the command has been executed, all information that is displayed in the Command File Output area will also be logged on the file.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

If the specified file does not exist, a new file is created. If the specified file does exist, its contents will be overwritten.

The logging is deactivated by any of the @C command and the @CLOSE command.

Error Handling

If the log file specified cannot be opened, the error flag will be set.

Related Commands

- @LOG
- @C
- @CLOSE

4.3.9 @R

Synopsis

@R +

@R -

Parameters

+ Command sending continues after the message

RESPONSE TEST FAILED WITH STRING:

Char_sequence

- Command sending stops after the message RESPONSE

TEST FAILED WITH STRING: Char_sequence

Description

If the command @R+ is executed before the @/W1,W2.../ command, the execution continues even if the text string specified by @/W1,W2.../ does not match the printout from the external system.

@R- is the default value. In this mode, the command file execution stops if no match occurs.

Error Handling

The command has no effect on the error flag.

@R+

@/ORDERED/

LABUP;

/^No order made!^/

Example 115 @R

Related Commands

@/W1,W2.../

4.3.10 @S

Synopsis

@S

Parameters

None

Description

This command has the same effect as the @LOGON command. It suspends the effect of a previously executed @E or @LOGOF command.

Error Handling

The command has no effect on the error flag.

Related Commands

- @LOGON
- @LOGOF
- @E

4.3.11 @T

Synopsis

@T Exp

Note: Note: A space is required between the command and the parameter.

Parameters

Exp An expression with a numeric value

Description

This command pauses the command file execution for Exp number of seconds.

If six seconds or more is specified, the countdown in seconds is displayed.

The maximum time is 3200 seconds.

Error Handling

The command has no effect on the error flag.

Related Commands

@DELAYSEC

4.3.12 @W

Synopsis

@W

Parameters

None

Description

This command displays a pop-up window with the following choices:

Continue Continues the execution

Stop Stops the execution

Stop and Quit Stops the execution and quits the application

Error Handling

The command has no effect on the error flag.

Related Commands

@PROMPT

4.3.13 @Z

Synopsis

@Z Char_sequence

@Z -

Note: Note: A space is required between the command and the parameter.

Parameters

Char_sequence Turns automatic response check on, and logs the result

in the file specified by Char_sequence

- Turns the automatic response check off

Description

@Z Char_sequence turns the automatic response check on. For each response, a check is made that the printout contains END , EXECUTED , or ORDERED.

The last line of the result is logged in the file specified by the Char_sequence parameter. A sequence number in the range 00-99, depending on how many log files that exist, is added to the log file name. The lowest vacant number is used.

Note: Note: These response check log files are different from the log files controlled by the@LOG and @CLOSE commands.

If the parameter begins with the / character, the parameter is interpreted as an absolute path. Otherwise, it is interpreted as a path relative to the current directory.

@Z - turns off the automatic response check.

Error Handling

The error flag will be set if this command fails.

5 Syntax Summary

This chapter contains a syntax summary.

5.1 Labels Syntax

Label ::= Label_char

Label Label_char

Label_char ::= Any printable character except ,)

5.2 Variables

Var ::= Var_char

Var Var_char

Var_char ::= Any printable character except < > = + - * / ( ) , ' " { } @. Variable names cannot start with a digit, $ or #.

5.3 Expressions

Exp ::= Prefixed_exp

Exp Arith_op Exp Prefixed_exp ::= + Simple_exp

Simple_exp

Simple_exp

Simple_exp ::= Var

OPS_script_function

Constant

(Exp)

Arith_op ::= one of + - * /

5.4 Constants

Constant ::= String_constant

Char_constant

Numeric_constant

String_constant ::= " Char_sequence "

' Char_sequence '

Char_sequence ::=Any member of the source character set except the new-line character, or the enclosing quotation mark, if any.

Char_constant ::= #Decimal_number

$Hex_number

Decimal_number ::= Decimal_digit

Decimal_number Decimal_digit Decimal_digit ::= one of 0 1 2 3 4 5 6 7 8 9
Hex_number ::= Hex_digit

Hex_number Hex_digit

Hex_digit ::= one of 0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

Numeric_constant ::= Decimal_number

5.5 Operators

Rel_op ::= one of < <= = <> >= >

Logical_op ::= AND | OR

5.6 Commands

Command ::= @OPS_script_command

@FIOL_command

!$$$$!

/{}Char_sequence* */

Every line that does not match this rule is interpreted as an MML command and sent to the network element.

Note: !$$$$! is an OPS script command, and /*Char_sequence */ is a FIOL command.

5.6.1 OPS_script_command

ops_script_command::= BREAK

| CASE(Exp)

| CHDIR(Exp)

| CHECK(Exp)

| CLOSE[(Exp)]

| CLRSCR

| COMMENT[([Exp {, Exp }])]

| CONNECT(Exp [, Exp1])

| DEC(Var)

| DEFAULT

| DELAY(Exp)

| DELAYSEC(Exp)

| DISCONNECT

| DRAW(Draw_exp {, Draw_exp })

| ELSE

| ENDIF

| ENDSWITCH

| ERASE(Exp)

| EVDELETE[([Exp])]

| EXECUTE(Exp)

| FLUSHVAR(Var)

| FOR Var = Exp1 TO Exp2

| FORM(Form_exp { , Form_exp })

FUNBUSY(Exp1 [, Exp2])

| GOSUB Label

| GOTO Label

| IF Cond_exp THEN [OPS_script_command]

| IFERROR THEN [OPS_script_command]

| INC(Var)

| INCLUDE(Exp)

| INKEY [([Var {, Exp }])]

| INPUT(Var {, Exp })

| LABEL Label

| LOG(Exp [,Exp1])

| LOGOF

| LOGON

| MAIL(Exp1,Exp2, Exp3 |FILE(Exp4))

| MARK [([Exp])]

| MENU(Var, Exp1 , Exp2 {, Expn })

| MKDIR(Exp)

| NEXT Var

| ONDISCONNECT (Exp)

| ONTIMEOUT(Exp,Exp1 ,Exp2)

| OPENREAD(Exp)

| ORDERED(Exp)

| PRINTER(Exp)

| PROMPT

| QUIT

| READ(Var [,LN(Exp)])

| RENAME (Exp1 , Exp2)

| REPORT(Exp)

| RESET

| RESTART

| RETURN

| RMDIR(Exp)

| SELPRINT(Exp)

| SEND(Exp {, Exp })

| SET Var = Exp

| SETERRORFLAG[(Exp)]

| SETPHASE(Exp)

| SETPROGRESS(Exp)

| SETREPLY(Exp1 , Exp2)

| SETTOTALCOUNT(Exp)

| SPONTREP(Var , Exp1 , Exp2 {, Expn })

| SPONTREPOFF[([Exp1 [, Exp2 {, Expn }]])]

| STEPPROGRESS[(Exp)]

| STOP

| SWITCH(Exp)

| VIEWFILE(Exp)

| WAITFOR(Exp1 [[,Exp2], Exp3])

| WAITREPLY [([Exp])]

| WINDEND

| WINDOW

| WRITE(Exp0 [,LN(Exp)]{,Expn })

Form_exp ::=Exp1, Exp2, Exp3, Var |

FIELD(Exp1, Exp2, Exp3, Var)|

COMBO(Exp1, Exp2, Var, Exp4 |Expn {,Expn })|

CMDBUTTON(Exp1, Exp2, Exp5, Exp6)|

CHECKBOX(Exp1, Exp2, Exp5, Var)|

RADIOBUTTON(Exp1, Exp2, Exp5 { , Exp1, Exp2, Exp5 }, Var)

Draw_exp ::= BOX, LeftX , UpperY , RightX , LowerY

| CUP, X , Y

| Exp

Cond_exp ::= Simple_cond_exp

{ Logical_op Simple_cond_exp }

Simple_cond_exp ::= Exp Rel_op Exp

| (Cond_exp)

Rel_op ::= one of < <= = <> >= >

Logical_op ::= AND

| OR

5.6.2 FIOL Commands

```
FIOL Commands
```

```
FIOL _command ::= /Char_sequence1 {, Char_sequencen }/
```

```
| A [+ | -]
```

```
| C
```

```
| E
```

```
| H Hex_number
```

```
| I Char_sequence
```

```
| L Char_sequence
```

```
| P [ + | - ]
```

```
| R [ + | - ]
```

```
| S
```

```
| T Exp
```

```
| W
```

| Z [Char_sequence | -]

6.6.3 OPS SCRIPT FUNCTIONS

OPS_script_function ::= ADVANCE(Exp1 , Exp2)

| CENTRALDIR( )

| CHECKCHILDSESSION(Exp)

| CONCAT(Exp1 , Exp2 {, Expn})

| COPY(Exp1 , Exp2 , Exp3)

| CREATECHILDSESSION(Exp1 {, Expn})

| DECIMAL(Exp)

| DISKFREE([Exp])

| EVLOADED([Exp])

| GETDATE([Var])

| GETDEST( )

| GETDIR( )

| GETERRORFLAG( )

| GETEXCHHDR( )

| GETFILELENGTH(Exp)

| GETIOTYPE(Exp)

| GETLOG([Exp])

| GETMARK( )

| GETMODE( )

| GETPHASE( )

| GETPRINT( )

| GETPROGRESS( )

| GETPROTOCOLTYPE(Exp)

| GETSCHED( )

| GETSESSIONID( )

| GETTOTALCOUNT( )

| HEX(Exp1 [, Exp2])

| INHISTORY(Exp)

| LABELEXIST(Exp)

| LENGTH(Exp)

| LOWCASE(Exp)

| PAD(Exp1 , Exp2 [,Exp3])

| POS(Exp1 , Exp2)

| PRIVATEDIR( )

| READVARIABLE(Exp1 , Exp2)

| REGRESS(Exp1, Exp2)

| REPLY (Exp)

| REPLYLEN( )

| SAVEDIR( )

| SCAN(Exp [,Var])

| SELFILE(Exp1 {,Expn })

| STARTCHILDSESSION(Exp1 {, Varn, Valn})

| STYLE(Style, Exp)

| TERMINATECHILDSESSION(Exp1)

| TIMEEVSET(Exp1 , Exp2 [, Exp3])

| TRIM(Exp)

| UPCASE(Exp)

| VAREXIST(Var)

| VERSION( )

Style ::= PLAIN|BOLD|ITALIC|UNDERLINE

6 OPS NUI Invocation via REST Endpoint

Authentication

To establish a valid connection with the NBI interface, see *Establish a User Session Over REST* section in ENM Identity and Access Management Programmers Guide.

Description

OPS has provided support for invoking OPS NUI through REST Call, which user can invoke with the help of Curl Command.

User invokes OPS initiation endpoint with the below parameters.

scriptFilePath – Mandatory. Fully qualified path of OPS script file (location and name of the OPS script). The script file passed must be accessible for the logged in user from OPS VM.

nodeName – Optional. Network element name to execute the scripts on.

routeFile – Optional. redirects the output of the commands given in script file at the specified path.

routemail – Optional. redirects the output of the commands to the mail of any specified user.

Command

```
curl --cookie cookie.txt --cacert ENM_PKI_Root_CA.pem -X POST -d '{"scriptFilePa →
th":"/ericsson/enm/dumps/ops_test/test"}' -H "Content-Type: application/json" 'h →
ttps:<customer-domain>/file/v1/files'
```

Example 9

```
curl --cookie cookie.txt --cacert ENM_PKI_Root_CA.pem -X POST -d '{"scriptFilePa →
th":"/ericsson/enm/dumps/ops_test/test"}' -H "Content-Type: application/json" 'h →
ttps://ieatenm5223-7.athtem.eei.ericsson.se/ops-service/launchers/v1/ops-nui-lau →
ncher'
OPS NUI requested to start by user administrator on Host scp-1-ops with Response →
 Starting the ops_nui...
Running...Wed Nov 25 13:22:50 GMT 2020
Completed
Run stopped: Normal Program Termination
-------------------------------------
Command file execution report
-------------------------------------
Date: Nov 25, 2020 1:22:50 PM
User: administrator
Session id: 17
Start file: /ericsson/enm/dumps/ops_test/test
Ext System: No Connection
Description: Normal program termination
```

7 Schedule OPS Execution

7.1 Crontab Job OPS Execution

Description

OPS supports scheduling via Cron Tab. Operator can schedule OPS CLI commands to be executed at certain time internal by putting required Cron Tab configurations.

Example 10

Example 117

schedule below 2 Jobs –

1. Execute Comment command from root user and save the output in Output1.txt.
2. Execute Connect to NE and 1 MML command from administrator user (switching to administrator from root via cronjob) and save the output in Output2.txt.

Command Used –

crontab -e - To Schedule Job

30 16 * * * /opt/ops/ops_client/bin/ops_nui -file /home/shared/administrator/cha/cmdfile/hellooo1 >> /root/Output1.txt

31 16 * * * sudo -H -u administrator bash -c '/opt/ops/ops_client/bin/ops_nui -file /home/shared/administrator/cha/cmdfile/hellooo2 >> /home/shared/administrator/Output2.txt'

Scheduling Job –

Figure 5 Figure 10

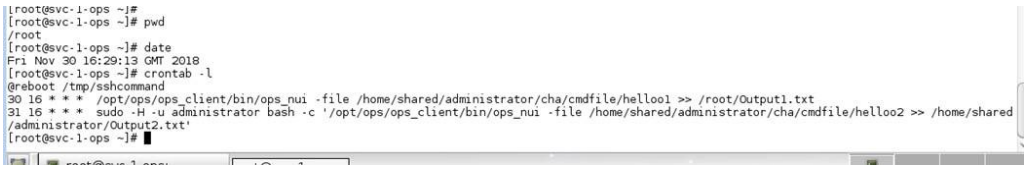

Figure 6 Figure 11

Figure 7 Figure 12

Figure 8 Figure 13

Input File:

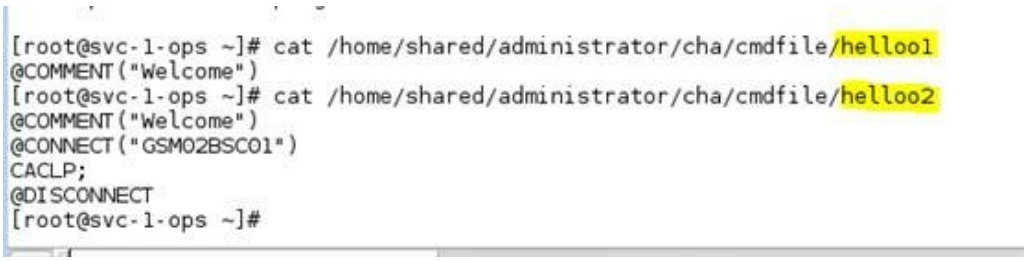

Figure 9 Figure 14

Output Files:

#1:

```
[root@svc-1-ops ~]# cat Output1.txt
Starting the ops_nui...
Running...Welcome

Run stopped: Normal Program Termination

------------------------------------
    Command file execution report
------------------------------------
       Date: 30-Nov-2018 16:20:59
       User: root
 Session id: 67
 Start file: /home/shared/administrator/cha/cmdfile/helloo1
 Ext System: No Connection
Description: Normal program termination

Starting the ops_nui...
Running...Welcome

Run stopped: Normal Program Termination

------------------------------------
    Command file execution report
------------------------------------
       Date: Nov 30, 2018 4:25:10 PM
       User: root
 Session id: 69
 Start file: /home/shared/administrator/cha/cmdfile/helloo1
 Ext System: No Connection
Description: Normal program termination

[root@svc-1-ops ~]#
```

Figure 10 Figure 15

#2:

Figure 11 Figure 16

Output:

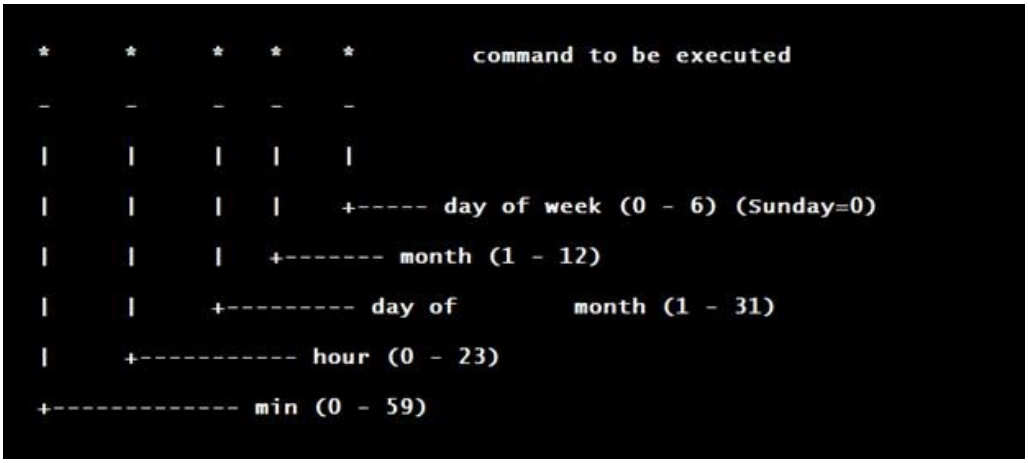

Figure 12 Figure 17

8 DUAL APG Support

8.1 Description

— If an NE has a second APG defined, OPS adds a new entry in the Network Element drop-down with the node's name with an appended _1 (example MSC01_1).

— If a script uses this, then connectivity is to the second APG.

— Only Aploc commands can be executed on Dual APG nodes.

Note: As OPS uses the string "_1" to identify the dual APG nodes, it is recommended to avoid naming Nodes ending with _1 explicitly (example:MSC17BSC34_1). This might cause incorrect behavior in OPS.

Example 11

Example 117

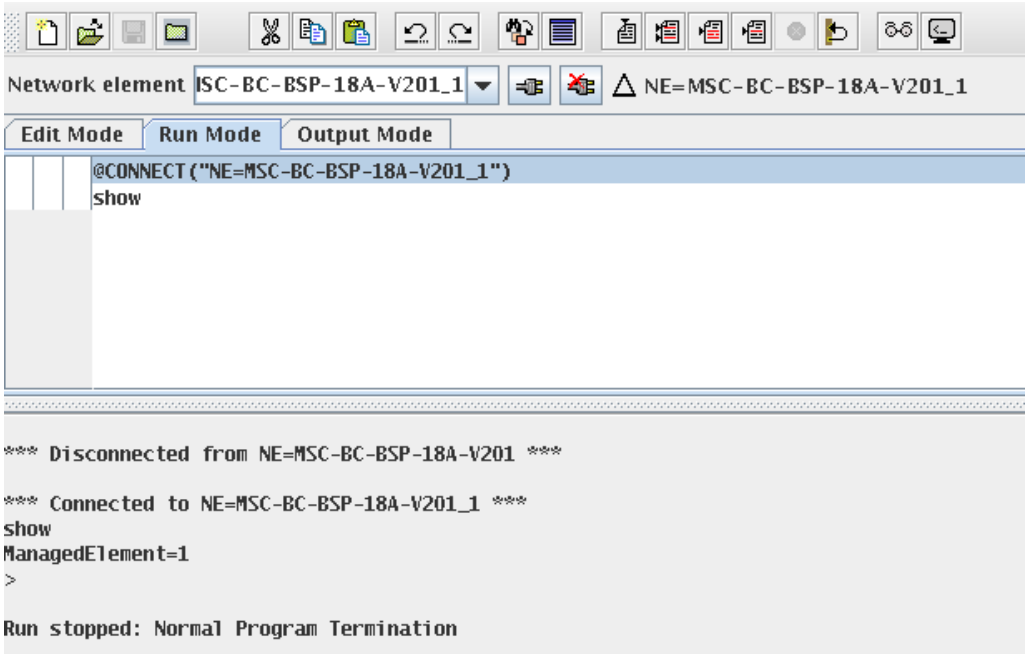

Figure 13 Figure 18 Dual APG Example

9 OSS-RC Utilities supported in ENM

The following are some of the functionality of OSS-RC utilities which is supported in ENM as indicated.

Table 3 Table 5 Utilities

OSS RC Utility	Utility Description	ENM Utility	Type	Usage in ENM
axs_cp	This utility is used to fetch the current version of ENM. The version returned is the R-State of the current ENM installed. Along with it is the OPS parser version.	getVersion.sh	Shell Script	`$ ./getVersion .sh AOM 901 151 R1 BX(1.3.3-1)`
infoXtract.sh	This utility is used to fetch the current version of ENM. The version returned is the R-State of the current ENM installed. Along with it is the OPS parser version.	getVersion.sh	Shell Script	`$ ./getVersion .sh AOM 901 151 R1 BX(1.3.3-1)`

eac_esi_config	This utility is used to fetch the list of network elements available and also to verify if the network element exists in ENM. When used with argument -l, it returns the list of network elements. When used with argument -v, it verifies if the given network element exists in ENM or not.	getNetworkElements.sh	Shell Script	To get complete list of network elements
				`$ ./getNetwork Elements.sh -l GSM01BSC01 GSM01BSC02`
				To verify if a specific network element exists in ENM
				`$ ./getNetwork Elements.sh -v GSM01BSC01 GSM01BSC01 $ ./getNetwork Elements.sh -v GSM01MSC01 NOT FOUND`

smoaxe	This utility is used to transfer files from ENM to Network Element and vice versa. Also, it can be used to perform some operations over ftp. It accepts multiple parameters and the usage of the same can be found using the argument -h. As of ENM release 18.3, only PUT and	`smoaxe.sh`	Shell Script	To perform file operations over FTP `$ ./smoaxe.sh -h Available options: -n Network Element on which the action to be performed -a Action to perform. Below`

OSS RC Utility	Utility Description	ENM Utility	Type	Usage in ENM
	GET actions are supported. Other operations will be supported in 18.4 release.			are the available actions PUT - Transfer file to Network Element GET - Transfer file from Network Element RENAME - Rename file on Network Element LIST - List files on the Network Element DELETE - Delete file from Network Element COPY - Copy file from source to destination within Network Element -s Source/From File Path -d Destination/To File Path -l Remote folder to list the contents. (To be used along with LIST action) -r Remote file to be deleted. (To be used along with DELETE action) ---------- USAGE: To transfer file to network element from ENM/SMRS: smoaxe.sh -a PUT -s <Source file name> -d <Destination file name> -n <Network Element name> To transfer file from network element to ENM/SMRS: smoaxe.sh -a GET -s <Source file name> -d <Destination f

OSS RC Utility	Utility Description	ENM Utility	Type	Usage in ENM
				ile name> -n <Network Element name> To rename file in network element: smoaxe.sh -a RENAME -s <Current file name> -d <New file name> -n <Network Element name> To copy file from one location to another location in network element: smoaxe.sh -a COPY -s <Source file name> -d <Destination file name> -n <Network Element name> To list contents of a directory in network element. Directory path must end with / smoaxe.sh -a LIST -l <Directory path to list> -n <Network Element name> To delete file in network element. smoaxe.sh -a DELETE -r <File to be deleted> -n <Network Element name>

10 BSC Sync Handling by OPS

The following cmedit commands are supported in ENM for BSC sync handling:

Table 4 Cmedit Commands Supported in ENM for BSC Sync Handling

ENM Utility	Utility Description	Type	Usage in ENM
node_sync.sh	This utility is used to enable or disable sync within OPS script in BSC. The operations supported include to enable BSC sync, to disable BSC sync, and to check the existing status of the attribute "active" in sync.	Shell Script	Available options: -n Network Element on which the action to be performed. -a Action to perform. The following actions are available: **CHECK** Usage: CHECK operation checks the current sync status. node_sync.sh -n <Network Element name> -a CHECK The output messages displayed are: SYNC_STATUS: DISABLED SYNC_STATUS: ENABLED FAILED: NODE <Node_Name> NOT AVAILABLE FAILED: EXECUTION FAILED **Note:** The status reported in this operation is whether the sync is enabled or not. The script does not verify

ENM Utility	Utility Description	Type	Usage in ENM
			if the node is synchronized or not **ENABLE** Usage: ENABLE operation sets the "active" attribute under CmNodeHeartbeatSupervision to true. node_sync.sh -n <Network Element name> -a ENABLE The output messages displayed are: SUCCESS: SYNC ENABLED FAILED: NODE <Node_Name> NOT AVAILABLE FAILED: EXECUTION FAILED **DISABLE** Usage: DISABLE operation sets the "active" attribute under CmNodeHeartbeatSupervision to false. node_sync.sh -n <Network Element name> -a DISABLE The output messages displayed are: SUCCESS: SYNC DISABLED FAILED: NODE <Node_Name> NOT AVAILABLE

ENM Utility	Utility Description	Type	Usage in ENM
			FAILED: EXECUTION FAILED

11 ENM Online Help For OPS

OPS Help Document

We can launch ENM online Help for OPS in following ways:

1. ENM Launcher → Operation Procedure Support → Help → App Help →Operation Procedure Support

2. ENM Launcher → Help Center →Operation Procedure Support