

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

BP30

Script Runner Specification

Edition 2006

Published by Neonseven s.r.l.,
Viale Stazione di Prosecco, 15
34010 Sgonico (Trieste) Italy

© Neonseven.
All Rights Reserved.

For questions on technology, delivery and prices please contact the Neonseven Offices in Italy Sgonico and Gorizia

Attention Please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics.

Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact Neonseven.

Neonseven technologies may only be used in life-support devices or systems with the express written approval of Neonseven, if a failure of such technologies can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Author	Diego Barbana	Department:	S2	Page: 1/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –		Confidential	

Table of Contents

1	Document Mission/Scope	3
1.1	Mission	3
1.2	Scope	3
2	List of Acronyms	3
3	Introduction	4
4	Reasons for using a script interpreter	5
4.1	Idea behind Runner	5
4.2	Runner structure	5
4.3	Using Runner to perform RF calibration	6
4.4	Using Runner to perform TESTS	7
5	Scripts	8
5.1	Remark lines and info strings	8
5.2	Script structure and execution sequence	8
5.3	Redirection concepts and mechanism	9
5.4	Redirection pipes	9
6	Configuration	11
6.1	INI file	11
6.2	Stimuli.txt driven configuration	11
6.2.1	Band specific parameters	12
6.2.2	Common parameters	12
6.3	Script driven configuration	12
7	Variables	15
7.1	Evaluating expressions	15
7.2	Using arrays	16
8	Report generation	17
8.1	Debug output	17
8.2	Logging	17
8.3	Variables dump	17
8.4	Template driven reports	17
9	References	19
9.1	External	19
9.2	Internal	19
10	Document change report	19
11	Approval	19
12	Annex	21

1 Document Mission/Scope

1.1 Mission

Runner is a tool developed in NeonSeven to test BP30 platform system and features. It is not intended for pre-production or mass-production.

1.2 Scope

This document summarizes functionalities and features of Runner environment, taking into account Runner setup on different workbench.

2 List of Acronyms

Abbreviation / Term	Explanation / Definition
EEPROM	Electrically Erasable PROM
DUT	Device under test
HOST	PC linked to DUT

3 Introduction

Runner.exe is a tool developed in NeonSeven to test BP30 platform system and features: as regard RF it was developed to calibrate boards and test their RF behavior. Some of the functionalities of the application are hard coded internally, others are accessible through scripts.

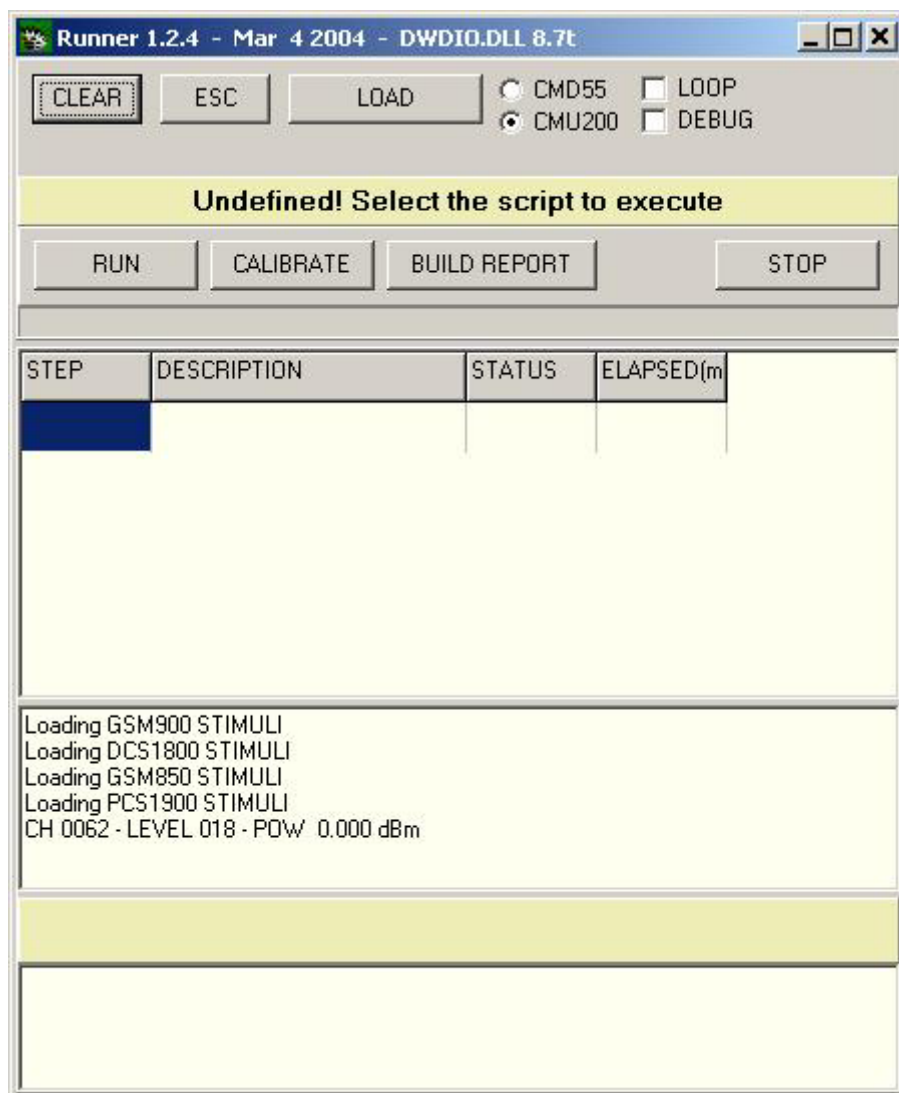


Figure 3-1

Author	Diego Barbana	Department:	S2	Page:	4/21
Filename	Script_runner.doc				
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

4 Reasons for using a script interpreter

Testing DUTs in laboratory (i.e. during development of project) and during mass production are two completely different tasks. In the latter case speed, time, system overall conformity (PASS or FAIL) and process yield are the main targets: the project is not under test. During project development and debugging a more elastic view takes place: the target is testing particular features and collecting data, being able to change target features and data output format as fast as possible. Test time is not the lead problem. Input and output data should be accessible.

4.1 Idea behind Runner

Many instruments and devices are able to communicate using simple command strings: this is the case of many radio communication testers and programmable power suppliers. Many mobile stations implement AT commands which are characters sequences.

From a user point of view asking a RF tester for a power measurement or a mobile station for software version means changing command: it is not important neither the hardware interface between HOST and instrument, nor the used low level protocol.

In such a vision, running scripts means redirecting command and answers to the proper destination in a predefined order.

4.2 Runner structure

Runner loads scripts and execute them line by line. Reserved words and symbols are used to drive special actions and to set environmental variables. The reserved strings are the following:

STRING	Meaning
//	At the beginning of line (first and second characters , no black allowed) means remark
#n#	Init device number <n> as specified
°n°	Performs special actions on device <n>
@n@	Redirect output to device n for following script lines or set output for following commands in the same line
@*@	Redirect output to DWDIO.DLL parser
[#]name[#]	Is replaced by content of variable 'name': first pre-processor run
##*name**	Is replaced by content of variable 'name' : second pre-processor run
>>>>	Stores result in following variable
FN	Executes special function (hard coded) : e.g. #FN#HW_CHECK_CALL_OPERATIVO
WAIT	Suspends execution for specified milliseconds (Windows Sleep API) : e.g. @WAIT@2000
INCLUDE	Specifies the name of a script file to load and execute nested : e.g. @INCLUDE@setup.seq

RESET_INSTRUMENT	Set script to execute at the end of test : e.g. @RESET_INSTRUMENT@reset_instruments.seq
REPORT	Add a template to the list of report generation : e.g. @REPORT@first_report.htm @REPORT@second_report.htm
INC	Increment following result variable : e.g. @INC@?>>>TIMES
DEC	Decrement following result variable : e.g. @DEC@?>>>TIMES
LABEL	Mark script line number with label (used for flow control) : e.g. @LABEL@start_label
TEST	Set FlowControl flag according to 0 if test succeeds (equivalence) : e.g. @TEST@*##TIMES*##:10
JMP	Unconditional jump to label : e.g. @JMP@start_label
JMPZ	Conditional jump to label (if flowControl == 0 , used after TEST)
JMPNZ	Conditional jump to label (if flowControl != 0 , used after TEST)
END_DO	Set flags to stop execution

Runner doesn't initialize any resource at startup (i.e. no comport is open, no GPIB link is established ...). It bases its behavior on scripts, also for environmental initialization.
 As the name let you understand, Runner interprets scripts, and according to these it opens comport, hpib links, uses DWDIO.DLL.....

4.3 Using Runner to perform RF calibration

The minimum HW/SW configuration to execute a calibration is:

- DWDIO.DLL (17.1x or higher versions : x is for customer specific version : e.g. 17.1i for Globe)
- PIPEDLL.DLL and related DLL, as needed by configuration.
- 1 serial link to DUT (the link used by DWDIO.DLL)
- 1 serial (or GPIB) link to BS (base station)
- 1 RF connection between DUT and BS
- 1 power supply (automatic or manual)

To handle calibration Runner uses some internal functions, accessible using CALIBRATE button.

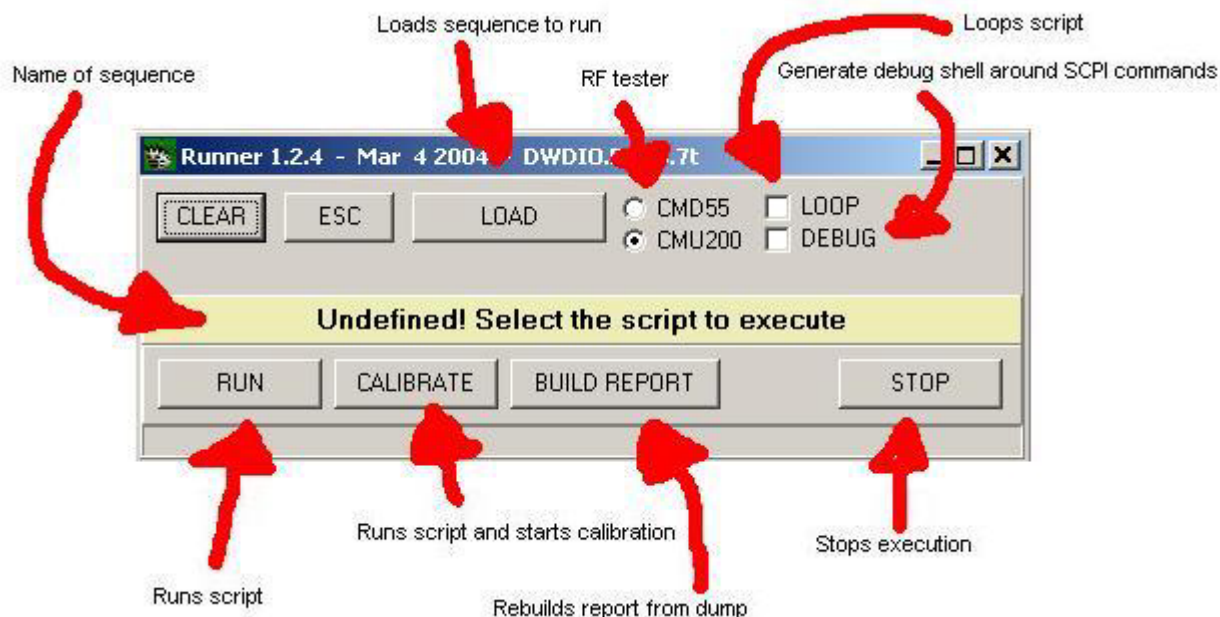


Figure 4-1

The correct procedure is the one below:

- Press LOAD button and choose the SEQ (sequence) file you want to execute: you always need a SEQ file, at least to initialize comport and DWDIO.DLL
- Check everything is OK: serial cables linked (and correctly set,...,RF cable connected to BOARD and to Base Station,...)
- Select the correct RF tester type (CMD55 or CMU200 supported)
- Press CALIBRATE button
- Enter the name you want to be used for log.
- The script is executed, and every resource is allocated
- If you have chosen to use manual power supply you will be asked to switch output on (from the supply): after that switch on also the DUT (device under test).
- In case of manual power supply you will also be asked to enter current consumption: it is just needed to collect data.
- After that the calibration starts.
- At the end you will be prompted to turn off VBAT.

Sometimes the test fails, something is not working...: just press STOP button and start again: If troubles don't disappear, maybe you'll need to exit and restart the application (or check your settings and scripts).

Warning: In order to allow calibration routines the DUT should be started in pTestMode (use PhoneTool to switch between modes!).

4.4 Using Runner to perform TESTS

The RUN button executes the script exactly the same way as CALIBRATE does, but at the end of execution no Calibration is started: it is intended to run measurements tests and scripts.

To run any kind of test, the user should create the scripts that initialize resources, symbols and constants, and use scripts to drive the script parser through test.

Author	Diego Barbana	Department:	S2	Page:	7/21
Filename	Script_runner.doc				
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

5 Scripts

The scripts loaded from Runner are simple test files with SEQ extension.
 When a script is executed the set of inputs and outputs strings generated are stored in a file with the same path and name of the script, and extension RES.

5.1 Remark lines and info strings

Remarks lines are preceded by double slash “//” string, like C++. Everything after double slash is not displayed neither to user debug application.
 To show any kind of message to user a valid console device should be used for redirection.

5.2 Script structure and execution sequence

In order to allow structured and simpler handling, test scripts can be split in different files, linked together through INCLUDE command.

Developer should keep in mind the way this works: every included script is interpreted and ran with a function call.

The same function handles include functionality, thus starting recursion: at the end of every script execution the recursion level is decremented. This means the scripts can be structured as nested sequences, but the maximum deepness of this nesting should not grow to high.

Every script can be included specifying its path: otherwise the default path for scripts will be used by application.

Here is an example of script calls inside a script (called for instance mainscript.seq)

```
[...]
@INCLUDE@setup.seq
@INCLUDE@reset_instruments.seq
[...]
```

In the example above the script *setup.seq* in the script default folder is executed, then *reset_instruments.seq* is executed, then the application continue the execution of the *mainscript.seq* sequence

While @INCLUDE@ command forces immediate execution, @RESET_INSTRUMENT@ that setups execution of a script at end of all sequences, both in normal ending and in stopped or error condition. Every time this command is found in a script the sequence to be ran is reassigned. See below for example:

```
[...]
@RESET_INSTRUMENT@reset_instruments.seq
[...]
```

Another command allow stop of execution inside script: @END_DO@. This string does not force stop of test as forced by an external (user) decision, it stops the test as normally ended.

Related to execution is also the @WAIT@ command, witch suspends execution for a specified number of milliseconds (the accuracy can change according to OS behavior). For example:

```
@WAIT@1500
```

Author	Diego Barbana	Department:	S2	Page:	8/21
Filename	Script_runner.doc				
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

5.3 Redirection concepts and mechanism

Except to some internal functionality every I/O works through redirection (better I/O string channeling).
 The typical structure of a script is as follows:

```
[...]INITIALIZATION OF RESOURCES
USE OF RESOURCES
SETUP OF RESOURCES
USE OF RESOURCES
TEMPORARY Use OF RESOURCES
USE OF RESOURCES
[...]
```

Here follows an example

```
[...]
// Power supply: initialization of COM2 as resource number 0
#0#COM:2
[...]
// Power supply: setup of resource number 0
°0°9600
°0°CRLF
°0°QUERYRXCLEAR
°0°QUERYWAIT=50
°0°RXCLEAR
[...]
// Power supply: general SCPI commands
@0@
OUTP:STAT OFF
*RST
@1@
//commands to resource 1
@0@
//following commands back to resource 0
*IDN?
[...]
[...]
```

5.4 Redirection pipes

Typical pipes redirection allows user to access serial communication port, interact with console, handle GPIB input/output, perform echo with variable...

Here follows a list of implemented pipe types:

- LOOP: handles strings. Allows output from script to be received (in loop) as input, thus enabling a simple processing of string literals.
- COM: handles serial com port
- CEC: handles GPIB parallel ports and devices (implemented for Capital Equipment Corporation GPIB interface hardware)

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

- NI: handles GPIB parallel ports and devices (implemented for National Instruments Corporation GPIB interface hardware)
- CON : handles console (input and output from/to operator)
- F(X) : handles expression execution and evaluation

Author	Diego Barbana	Department:	S2	Page: 10/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

6 Configuration

Resource allocation is handled by scripts; logical devices are also handled by INI file. Sometimes other values are needed (especially during calibration: RF power levels, RX channels...) and these are loaded from a stimuli.txt file.

6.1 INI file

In the folder where Runner.exe lies there is a runner.ini file. Here are few options settable:

- [STATION]
 - [leave empty: backward compatibility]
- [RF_TESTER]
 - TYPE : 2 is for CMD55
 - LOGICAL_ADDRESS : 1 if the scripts access Base Station as @1@
- [POWER_SUPPLY]
 - TYPE : 0 for manual; 1 for HP6632B or HP66332A
 - LOGICAL_ADDRESS : 0 if the scripts access BS as @0@
- [SEQUENCE]
 - Turn_on = 0 : performs the test specified and related calibration task (leave disabled)
 - Stand_by_current_test = 1 : performs the test specified and related calibration task
 - Inline_test_mode = 1 : performs the test specified and related calibration task
 - dwd_get_sw_version = 1 : performs the test specified and related calibration task
 - Init_RF_analyzer = 1 : performs the test specified and related calibration task
 - Check_RTC = 1 : performs the test specified and related calibration task
 - Battery_voltage_ADC_adjust = 1 : performs the test specified and related calibration task
 - TX_current_test = 1 : 1 means performs the test specified and related calibration task
 - VCXO_adjust = 1 : performs the test specified and related calibration task
 - P2002_PA_adjust = 1 : performs the test specified and related calibration task
 - RX_level_adjust = 1 : performs the test specified and related calibration task
 - Store_NV_memory = 1: stores calibration data to EEPROM
 - Turn_off = 1: performs the test specified and related calibration task
 - Write_data_report = 0: performs the test specified and related calibration task

For example, if no power supply is connected (through serial or GPIB cable 9 to HOST pc, then the following lines should be present in the executed scripts

```
[...]
[POWER_SUPPLY]
TYPE = 0
LOGICAL_ADDRESS = 0
[...]
```

All other setting occur in scripts.

6.2 Stimuli.txt driven configuration

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

In the same folder as Runner.ini there is also a file named STIMULI.TXT : it consists of TARGET dependant configuration parameters which need to be changed moving from older HW/SW VERSION to newer ones or viceversa.

These parameters are loaded by application at startup (so changing them during application execution won't effect execution behaviour until next startup) to handle internal hardcoded calibration routines.

Possible sections are: GSM900, DCS1800, GSM850, PCS1900 and COMMON.

6.2.1 Band specific parameters

First four sections contain RF TX target power levels and RX and TX channel used during calibration.

TARGET_POWER are the target power levels to be reached after calibration (their index is between 0 and 19 (PCL), so depending on band they could also be NULL).

TX_ARFCN are the 3 channel used to compute TX power compensation versus frequency.

RX_ARFCN are the 8 channel used to compute RX power compensation versus frequency.

VCXO_CAL_PCL is the power level used to perform AFC calibration.

VCXO_ARFCN is the channel used to perform AFC calibration.

HF_CORR_VALUE is the HF_COOR_VALUE used internally by DUT software to perform RX measurements.

The value must match the value used by target on per-band basis.

6.2.2 Common parameters

COMMON section contains startup, RXlevel and ADC calibration related parameters.

INITIAL_TX_ARFCN and INITIAL_TX_PCL are the starting TX channel and power level during calibration.

CW_FREQ_OFFSET is the offset used for cw mode signal generator (if used)

RX_GAIN sets 3 fixed gains used during RX level measures calibration, and RX_INPUT_LEVEL sets the RF power input level (set on Signal Generator).

SD_MODEL is the calibration compensation model used by Runner: if SD_MODEL is 0 (smarti DC continous model) then SMOOTH_RX is enabled. Using Smarti SD hardware SD_MODEL is fixed to 1.

As regard ADC measures and calibration, the value of the following parameters can change to adapt to different HW and SW (please find close to the string literals the default values):

```

LOW_SUPPLY_VOLT = 3.22
MEDIUM_SUPPLY_VOLT = 3.8
HIGH_SUPPLY_VOLT = 4.3
CHARGER_MEASURES = 0

```

Last setting (CHARGER_MEASURES) enables ADC measures through charger driver (not working in INLINE mode). The default value (CHARGER_MEASURES = 0) set direct ADC measures through inline driver (working only in INLINE mode).

6.3 Script driven configuration

In the scripts collection delivered every main script should start like this:

```

#####
@RESET_INSTRUMENT@reset_instruments.seq
@INCLUDE@setup.seq
@INCLUDE@serial_create_CMU.seq
//@INCLUDE@hpib_create_CMU.seq

```

Author	Diego Barbana	Department:	S2	Page: 12/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -		Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

@INCLUDE@reset_instruments.seq

or there could be a call to hplib_create.seq instead (in order to use GPIB link)

@INCLUDE@ means that the file will include the one indicated after the second @: in this case serial_create.seq. hplib_create.seq is in remark (//).

In the SEQ file SETUP.seq RF cable correction loss and the com port settings used by DWDIO.DLL or to communicate to instruments are specified this way:

```
// local echo
#3#LOOP
@3@
// cable loss ( in dB ) for GSM,DCS and PCS band
0.25?>>>>$GSM_CABLE_LOSS
0.4?>>>>$DCS_CABLE_LOSS
0.4?>>>>$PCS_CABLE_LOSS

// RF connector ( usually used for CMU200 ) for GSM,DCS and PCS band
1?>>>>$GSM_RF_CONN
1?>>>>$DCS_RF_CONN
1?>>>>$PCS_RF_CONN

// UART port number ( 1 based ) for DUT, RF_TESTER and POWER_SUPPLY
4?>>>>$DUT_COM_PORT
1?>>>>$RF_TESTER_COM_PORT
3?>>>>$POWER_SUPPLY_COM_PORT

// UART baudrates for RF_TESTER and POWER_SUPPLY
115200?>>>>$RF_TESTER_COM_BAUD
9600?>>>>$POWER_SUPPLY_COM_BAUD

//prefix for EEP_CFG file
BP30?>>>>$EEP_PREFIX
```

Every variable set this way is then used in the scripts, for example:

```
o *#$DUT_COM_PORT *#?
```

This means open com 4 port and check for result

The serial_create_CMU.seq looks like this:

```
//Create communication, reset all instruments and ask identification

// Power supply
#0#COM:*#$POWER_SUPPLY_COM_PORT*#
°0°*#$POWER_SUPPLY_COM_BAUD*#
°0°CRLF
°0°QUERYRXCLEAR
°0°QUERYWAIT=50
°0°RXCLEAR

// CMU base
```

Author	Diego Barbana	Department:	S2	Page: 13/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -		Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

```
#1#COM:***$RF_TESTER_COM_PORT**
°1°***$RF_TESTER_COM_BAUD**
°1°CRLF
°1°QUERYRXCLEAR
°1°QUERYWAIT=500
°1°PREFIX=0;
```

```
// CMU GSM900
#4#COM:***$RF_TESTER_COM_PORT**
°4°***$RF_TESTER_COM_BAUD**
°4°CRLF
°4°QUERYRXCLEAR
°4°QUERYWAIT=500
°4°PREFIX=1;
```

```
// CMU DCS1800
#5#COM:***$RF_TESTER_COM_PORT**
°5°***$RF_TESTER_COM_BAUD**
°5°CRLF
°5°QUERYRXCLEAR
°5°QUERYWAIT=500
°5°PREFIX=2;
```

```
// CMU PCS1900
#6#COM:***$RF_TESTER_COM_PORT**
°6°***$RF_TESTER_COM_BAUD**
°6°CRLF
°6°QUERYRXCLEAR
°6°QUERYWAIT=500
°6°PREFIX=3;
```

```
// Operator
#2#CON
```

It creates one serial link to Power supply (#0#COM:***\$POWER_SUPPLY_COM_PORT**), one serial link to GSMTester (#1#COM:***\$RF_TESTER_COM_PORT**), and a Console connection to the operator (#2#CON).

The init string specified above is structured in the way described below:

#n#COM:m → create a serial link with LOGICAL ADDRESS n, using com port m

Author	Diego Barbana	Department:	S2	Page: 14/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

7 Variables

Any time user needs storage of temporary data or variables are needed, a particular syntax is used. Every input (coming out from instruments, console, loop pipes...) can be stored in variables, using the following syntax:

```
//assuming device 2 is console 8 input from operator )
@2@
Connection loss?>>>> $GSM_CABLE_LOSS
```

Everything following ">>>>" string (except for remarks) is intended as variable name (not case sensitive: converted to capitals letters). The environment manages a list of variables, and this list grow up during execution (variables are global, and reset every time a test is started (RUN or CALIBRATE button). The name of the variable defines its type: variable names starting with '\$' character indicates strings, others numeric ("%6.3f" float format).

To retrieve the content of variable the name must be used enveloped in double "***" tag.

```
SENS:CORR:LOSS ***$GSM_CABLE_LOSS***
```

If input from operator in preceding step was 1.1, then the line above is converted in

```
SENS:CORR:LOSS 1.1
```

Numeric variables can be incremented or decremented using INC and DEC operators: is value of variable is not integer then it is truncated to integer.

INC and DEC operators works without redirection, so they can be used on a single line.

```
@INC@?>>>>TIMES
```

Variables can be compared using TEST operator, and the result can be used to perform controlled sequence jumps (here @3@ is assumed to be an ECHO pipe).

```
@3@
0?>>>>TIMES
@LABEL@start
[...]
@INC@?>>>>TIMES@TEST@***TIMES***:10
@JMPNZ@start
```


After @JMPZ@ and @JMPNZ@ there must be at least an empty line, since the line is processed before branch evaluation.

7.1 Evaluating expressions

Using numerical expressions pipes (created by F(X) reserved word) scripts can perform mathematical computations. Writing to F(X) pipe of C-like expressions (';' between expressions) cause variables (double float precision) to be stored in F(X) environment. If expression is the name of a variable, then successive reading out form F(X) pipe will return the value of variable. See example below:

```
// Expression evaluator creation
```

Author	Diego Barbana	Department:	S2	Page:	15/21
Filename	Script_runner.doc				
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG -			Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

```
#0#F(X)
@0@
t=17*3;
t^>>>>temp_value
t=sqrt(t*t+3);
t^
t+=[#]temp_value[#]
t^
```

7.2 Using arrays

Using arrays is possible using variable substitution order. The tags “[#]” and “*#*” have different priority during substitution: “[#]” is replaced as first. See following example:

```
@0@
MEAS:VOLT?>>>>V_INPUT_[#]INDEX[#]
@F(X)@(*#*V_INPUT_[#]INDEX[#]*#*)/2.75>>>>TEST
```

Assuming INDEX value is ‘2’; the script above is preprocessed as described below:

- In the second line V_INPUT_[#]INDEX[#] is set to the result of a measurement, so the result (eg. 3.2345) is stored in a variable with name dependant on INDEX value → V_INPUT_2
- In the third line *#*V_INPUT_[#]INDEX[#]*#* is pre-processed in *#*V_INPUT_2*#*, then the value of V_INPUT_2 replaces variable name → (eg. @F(X)@(3.2345)/2.75>>>>TEST)

Author	Diego Barba	Department:	S2	Page: 16/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

8 Report generation

Report generation is not the main point of Runner.exe, but is an important feature for analysis of results. There are four levels of report system: debug output, logging, variables dump and template driven reporting

8.1 Debug output

Almost everything happening inside Runner is traced to debug Monitor. Using Monitor log or simply monitor display window enables developer to analyze results and execution behavior.

8.2 Logging

Every scripts executed generate a log of input and output strings. It is logged to a file with the same name as the script, with the extension RES, placed in the DEBUG path. Output commands are only logged,

*OPC

While commands waiting for input are followed by the input itself

*IDN? --> ROHDE&SCHWARZ,CMD55,835651/027,V 4.10 16.06.00

This can be useful to analyze test environment condition and settings.

8.3 Variables dump

Every time a test is stopped or finishes normally the list of variables created and used during execution is dumped to a text file in the default debug folder. The name of dump file is

VARDUMP_hhnnssddmmyyyy.TXT

Where <hhnnssddmmyyyy> is the time and date identification of end of test (hour, minute, second, day, month, year).

This file can be used to store test results (can be easily loaded by spreadsheets or used for postproduction..), or can be reworked by Runner later together with templates to generate Runner report based on variables values (press REPORT button and select dump file and template).

8.4 Template driven reports

Runner understands the REPORT string as an internal command: it indicates a name of file to be used as template for report generation. Every time the REPORT command is reached, a new report template is added to report generation list (thus allowing more than one report using the same values). At the end of execution all report templates are parsed for occurrence of variables placeholders, and substitutes value from test if existing.

The document format of the template file is not important, but it should be a text file (HTML, HTML XLS files, XML, TXT, CSV.... are fine).

Here is an example of HTML XLS template

```
<tr height=17 style='mso-height-source:userset;height:12.95pt'>
  <td height=17 class=xl39 style='height:12.95pt'>&nbsp;       </td>
```

Author	Diego Barbana	Department:	S2	Page: 17/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –		Confidential	

Author	Diego Barbana	Department:	S2	Page:	18/21
Filename	Script_runner.doc				
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential	

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

9 References

9.1 External

None

9.2 Internal

Title	Doc ID
RF_calibration.pdf	AH01.SW.TN.000100

10 Document change report

	Change Reference		Record of changes made to previous released version	
Rev	Date	CR	Section	Comment
1.0	20/02/2004	Diego Barbana	Document created	
2.0	04/03/2004	Diego Barbana	General	R&S CMU200 supported
3.0	18/03/2004	Diego Barbana	General	PCS1900 supported, Smarti SD RX calibration supported
4.0	25/03/2004	Diego Barbana	General	Small bugs fixed and optimization added: Runner version 1.2.6
5.0	13/05/2004	Diego Barbana	General	Added support for direct ADC measures in INLINE mode. Added flow control instruction and numeric variables basic operators. Runner version 1.2.7
6.0	20/07/2004	Diego Barbana	General due to template modification	Template changed. Added description of pipes and calcpipe example
7.0	02/03/2005	Diego Barbana	General due to template modification	Template changed.
8.0	06/02/2006	Diego Barbana	General	Adapted to BP30.

11 Approval

Revision	Approver(s)	Date	Source/signature
5.0	Stefano Godeas	13/05/2004	Document stored on server

Author	Diego Barbana	Department:	S2	Page: 19/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

Revision	Approver(s)	Date	Source/signature
6.0	Stefano Godeas	20/07/2004	Document stored on server
7.0	Stefano Godeas	02/03/2005	Document stored on server
8.0	Stefano Godeas	06/02/2006	Document stored on server

Author	Diego Barbana	Department:	S2	Page: 20/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential

	Technical Specification	Doc. ID: AH01.SW.TS.000005 Rev.:8.0 Date:07/02/2006
---	--------------------------------	---

12 Annex

None

Author	Diego Barbana	Department:	S2	Page: 21/21
Filename	Script_runner.doc			
M06-N7 Rev. 2	Copyright (C) 2005NeonSeven S.R.L. All rights reserved - Exclusive property of Infineon Technologies AG –			Confidential