OPENCARAC USER MANUAL

Jonathan Certes

May 26, 2016

Copyright

Third-party software components

Contents

Chapter 1

Introduction

Abstract: Welcome to openCarac user manual. This document's purpose is to introduce the tool that is openCarac and explain how to use it in order to automatize your Spice simulator *runnings*. Assumption is made that the reader already has basic knowledges about circuit simulation using Spice. Section Brief description gives an overview of openCarac, section openCarac output files lists the output files, section Glossary defines the openCarac specific vocabulary and frequently asked question are answered in section FAQ.

openCarac is a work in progress; reports about bugs, missing items, wrongly described items, bad English style are deeply appreciated.

1.1 Brief description

In electronics, a designer who creates a circuit has to verify its functionality by performing various simulations such as distortion measurement, noise calculation, stability evaluation and so on ; if he wants to test his circuit's robustness, devices corners (typical, slow, fast, etc.) should vary and simulation conditions such as temperature, power supply voltage or bias current must be tuned. When crossing all those conditions, it results a huge amount of simulations to run that can be difficult to handle ; also, Spice simulators generate formated results files that are not always easy to treat or summarize and getting useful informations out them can be difficult.

This is when openCarac could be useful ; this tool has been implemented to automatize tasks by running the user's favorite Spice simulator in predefined conditions. The user only has to create a *configuration* file in a simulation environment organized to this purpose and run openCarac in order to make all his simulations run and have his data summarized in a results folder.

Indeed, by using openCarac, the designer will be in possession of easily exploitable files to treat his data and create quality document out of them as described in section openCarac output files.

Note: Since many Spice simulator exist and do not accept the same syntax, give the results files in the same format, a choice had to be made to make openCarac compatible with some of them. As a consequence, openCarac is not natively compatible with every Spice simulators. Nevertheless, an API permits to add other features and extend openCarac compatibility.

1.2 openCarac output files

When simulating using a Spice simulator, the designer usually extracts data values out of his work using a `.MEAS` or `.MEASURE` directive[1]. It permits the user to access a value of what he wants to measure. This is the kind of data openCarac focuses on: this tool is used to save and organize what is given by the simulator. Indeed, after having executed the simulator, openCarac parses the results files and creates its own output files by summarizing what it has found.

In these results files, one can find the following formats:

- **Tables in HTML format:** navigate easily through your simulation results using hyper-links in a web browser.

[1] Such a directive may vary from a simulator to an other

- **LaTeX code files:** make quality documents with your results saved in tables.
- **GNU Octave ^{TM} script files:** treat your data easily by calculating anything with your results saved into matrices.

1.3 Glossary

In this documentation, some openCarac specific vocabulary is used, this section gives a definition for each word:

application refers to the openCarac application: when global settings are altered, this affects all the openCarac application behaviour.

configuration refers to a directory where various *caracs* can be run: there must be only one main file, various simulation and netlist files in the directory root. When calling the Spice simulator for a *running*, a temporary folder is created at the same hierarchy level as the configuration directory.

carac refers to a group of *simulations* loading the same "model" file and the same "libparam" file and having the same specifications. A *carac* is defined between two `.CARAC` or `.CHANGE` directives in a *configuration* file. See subsection .CARAC directive for more informations.

simulation refers to a group of *runnings* depending on the same simulation file to be included in the main file. A *simulation* is defined through a `.SIMU` directive if in a *configuration* file. See subsection .SIMU directive for more informations.

running refers to one execution of the Spice simulator in a temporary folder. A *running* is defined for each corner/param/netlist tuple available in a *carac* for each available *simulation*.

results extraction refers to the access of every saved measure for each *running*, it usually comes with the creation of HTML, LaTeX and GNU Octave ^{TM} files.

openCarac main executable refers to the `openCarac.tcl` file that is available with openCarac, it uses openCarac API functions to form a stand-alone program.

main file refers to the file containing *MODEL*, *PARAM* and *SIMU* markers; it must be unique in each *configuration*. In this file are substituted *model* and *libparam* loadings, so as *netlist* and *simulation* files inclusions.

model refers to the file that must be loaded after the *MODEL* marker in the main file, its path can be defined through a `.MODELLIN` or `.MODELWIN` directive in the *configuration* file.

libparam refers to the file that must be loaded after the *PARAM* marker in the main file, its path can be defined through a `.LIBPARAMLIN` or `.LIBPARAMWIN` directive in the *configuration* file.

corner refers to the library to load in the model file when substituting in the main file.

param refers to the library to load in the libparam file when substituting in the main file.

netlist refers to the separated file to be included in the main file, it can be identified by openCarac through its extension.

1.4 FAQ

1.4.1 Where does the name "openCarac" come from?

Since openCarac is a tool to simulate an electronic circuit in various conditions, it is used to characterize it; so, I, being French, have chosen this name as follows:

The "carac" suffix comes from the French word *caractérisation*, meaning "characterization".

Of course, the prefix "open" has been added since it is an open-source project.

1.4.2 Can openCarac be natively compatible with other Spice simulators?

There are many available Spice simulators and being natively compatible with all of them represents a huge amount of work: they all accept a different syntax for their directives and output files formats are always different. So, adapting openCarac to the most popular simulators is the goal that has been fixed but, this project being a work-in-progress, we will have to wait a little to see it compatible with other simulators.

Nevertheless, even if your favourite simulator is not natively compatible with openCarac, a custom command with its syntax can be defined to automatize the *runnings* anyway. The openCarac API functions also permit to incorporate your own output files parsers into the execution and have the results treated as wanted.

1.4.3 Why openCarac has been coded with TCL?

The main reason openCarac has been coded in TCL (tool command language) is because it is the language I wanted to learn.

This language being mainly used to command simulators using their API (application programming interface), it can easily be integrated to their environment. Also, since TCL is a script language that does not need to be compiled, execution of an external simulator becomes even easier ; so does the execution of openCarac from the TCL interpreter of the same simulator.

1.4.4 Does openCarac come with a Graphical User Interface

The simplest answer is "not yet". Adding a GUI is the next step in openCarac development, it should be available soon.

1.4.5 Why is there a duck printed in my result files?

I believe that duck is the most beautiful animal I ever saw and I wanted to share my passion with any potential user. Adding a lemon could also be a nice touch.

Chapter 2

Your first carac

Abstract: In order to use openCarac, the user has to adapt his environment to make it compatible with. Since this installation requires a little comprehension of how the tool works, this chapter describes the steps to prepare your environment to welcome openCarac: section Add openCarac to your environment details how to configure the execution of your favorite simulator through openCarac ; section Organize your Spice circuit files shows how to write your Spice files in order to welcome openCarac and section Run a carac explains how to have your first *carac* working.
Note that openCarac is highly configurable and this chapter aims to provide an overview of how to use it. As a consequence, assumption is made that openCarac environment settings are set to default in the whole chapter. More informations about the execution are given in chapter openCarac usage.

2.1 Add openCarac to your environment

First, know that openCarac is a cross-platform[1] TCL script that automatizes the execution of a Spice simulator. Then, it must be executed through a TCL interpreter and a compatible Spice simulator must be available.

So, before being able to run openCarac, make sure that a TCL interpreter is available (openCarac works fine with version 8.5) and so is a Spice simulator present in the following list:

- Ngspice ^{TM} (tested with version 26)
- Gnucap ^{TM} (tested with version 2009.12.07)
- Xyce ^{TM} (tested with version 6.2)
- Dolphin Integration SMASH ^{TM} (tested with version 6.0)

2.1.1 Run openCarac:

Basically, as any TCL script, openCarac must be executed through a TCL interpreter, then you must know the command to execute it. We suggest that you run your interpreter from your command line interface, because when openCarac terminates, TCL command "exit" is executed and TCL interpreter quits; if it has been executed from a command line interface, only the interpreter does and previously printed text remains visible.

Since openCarac is not only a program but also a TCL package, there are various ways to execute it ; to make its use easier, it comes with a main executable named `openCarac.tcl` in the source folder. Executing the openCarac program is done through this main executable.

So, execute the command to run your TCL interpreter followed by the path to access openCarac main executable. This may look like one of the following commands.
If you are using a Unix like OS:

[1] openCarac has been tested under both Linux and Windows.

```
tclsh ~/openCarac/src/openCarac.tcl
```

If you are using Windows:

```
C:\tcl\bin\tclsh.exe C:\openCarac\src\openCarac.tcl
```

If you have an error message from your command line interface, make sure that both TCL interpreter command and openCarac script path are correct.

Otherwise, you can see the copyrights printed in your command line interface, this means that the execution has occurred correctly, even if you see an error message printed by openCarac : this only means that guidelines must be followed to ensure a perfect execution sequence.

2.1.2 Define your simulation environment

Congratulations, your are now able to run openCarac. But you may wonder how does it know how to execute your simulator? The answer is simple: it does not. So, openCarac is not yet integrated to your environment: you must then tell it what is the command and its options to execute your favorite simulator the way you want.

To do so, just give the argument to your TCL interpreter to make openCarac create a environment file, adapt and execute the following command. In case you are executing openCarac through a TCL interpreter different from *tclsh*, you can set your argument by setting the variable argv to "--configure". See section Execution options for more informations.
If you are using a Unix like OS:

```
tclsh ~/openCarac/src/openCarac.tcl --configure
```

If you are using Windows:

```
C:\tcl\bin\tclsh.exe C:\openCarac\src\openCarac.tcl --configure
```

As a consequence, openCarac will create a folder named `.openCarac` and an environment file into your user home directory, which you can modify to adapt openCarac application settings to your environment. A message should be printed giving the location of this file.

Note that another file, `customProcedures.tcl`, has also been created, its usage is for experts with TCL language and is described in chapter Expert mode.
In this environment file are described the settings of openCarac: parameters are defined and set to particular value, comments are also present starting with an hash (#). Now, just open this environment file with your favorite text editor, this may look like something as follows:

```
[application]

modelMarker = **MODEL**
paramMarker = **PARAM**
simuMarker  = **SIMU**

filesExtensionFilter    = .sp .cir .spi .ckt
simulationFileExtension = .inc
netlistFileExtension    = .nsx

# some other settings here

[ngspice]

command = ngspice

checkOptions = --batch
runOptions   = --batch --autorun

saveFilter   = .raw .data .ps .eps .plt .log
logExtension = .log

# some other settings here
# etc.
```

These are the definitions of the parameters to set openCarac environment: the parameters are separated in categories having their name between brackets ; on each line, the name of the parameter is located between the beginning of the line and the equal (=), its value is then defined between the same equal and the end of the line. You can modify them in order to change the commands to execute the simulators. A detailed description of the file syntax is given in section Environment file syntax.
So, in order to make openCarac compatible with your simulation environment, just change the values of the command to execute the appropriate simulator ; i.e. the "command" parameter in the category having its name matching the simulator's. Here is an example of a environment file to use openCarac under Windows[2]:

```
[application]

# Default output log file:
logFile = ../myOpenCaracLogFile.txt

# Default simulator:
simulator = ngspice

# some other settings here

[ngspice]

command = C:\\Program\ Files\\spice\\bin\\ngspice.exe
checkOptions = --batch
runOptions   = --batch --autorun

# some other settings here
# etc.
```

As soon as this environment file has been properly filled, it seems that openCarac has been added to your environment, you are now ready to use it!

2.2 Organize your Spice circuit files

The purpose of openCarac is to automatize runnings using your simulator: when characterizing a circuit, the designer must run various type of simulations such as AC, transient, noise... Also, he might tune *simulation* parameters: bias current,

[2]Note that, when defining a path, you must use escape syntax for backslashes or space characters.

power supply voltage, temperature... He may make his components corners vary too: slow, fast, best, worst, whatever the definition is.

So, openCarac provides a way to run every needed simulations in the previously designed conditions. In order to do so, the user must organize his Spice files to make them fully compatible with openCarac; a way to organize them has been chosen and is described as follows:

- A main file contains inclusions of other files or loadings of libraries. openCarac can identify it when it contains *MODEL*, *PARAM* and *SIMU* markers.
- Devices models are defined in only one file containing various libraries, openCarac identify this as the "model"; each library contains the definitions of the devices, openCarac identifies this as a "corner".
- Simulation conditions are established into one only file containing various libraries, openCarac identify this as the "libparam"; each library contains the definition of parameters, openCarac identifies this as a "param".
- Each *simulation* is defined into a separated file that must be included into the control file only once at the time. The extension for *simulation* files must be unique.
- The circuit definition can be defined in a separated file as soon as there is a unique inclusion of a file with its extension in the main file. openCarac identifies this as a "netlist".

In the previously defined control file, loadings of libraries in both *model* and *libparam*, so as inclusions of *simulation* files, can be operated by openCarac at the location stipulated by markers. So, the user finally has to add these markers into his control file and then, openCarac will use them to tune it for various *runnings*. Since comment character in Spice syntax is usually the star (∗), it is highly recommended to choose these marker starting with a star. By default in openCarac, these markers are:

- ∗∗**MODEL**∗∗ : marker to explicit the location where to change the *corner* loading in the *model*.
- ∗∗**PARAM**∗∗ : marker to explicit the location where to change the *param* loading in the *libparam*.
- ∗∗**SIMU**∗∗ : marker to explicit the location to include the *simulation* file with the appropriate extension (default value for this extension is `.inc`).

Every time that one of these keywords is encountered by openCarac, the following line in the file is altered before a *running* of the simulator occurs.

After having prepared the files like as previously described, the user will be able to configure openCarac to make it change the library to load for *model* selection, for *libparam* selection or the files to include. Here is an example of a Spice main file that allows to run a *carac* using openCarac:

```
** openCarac example circuit **
*******************************

**MODEL**
.LIB "../../foundryModels/models.lib" TYPICAL

**PARAM**
.LIB "../../parameters/conditions.lib" TYP

**SIMU**
.INC "op.inc"
*.INC "openLoop.inc"
*.INC "transient.inc"
*.INC "noise.inc"

** The netlist is a separated file:
.INC "nampli.nsx"

.END
```

In this example is represented the control file that contains the markers. We can note that, in the *model* file named `models.lib` is loaded a *corner* library called `TYPICAL`. Respectively, in the *libparam* file named `conditions.lib` is loaded a *param* library called `TYP`. In here, four *simulations* are needed to characterize the circuit, they are separated into different files: `op.inc`, `openLoop.inc`, `transient.inc` and `noise.inc`. The *netlist* that contains the circuit definition has its extension set to `.nsx`, it can be altered by an other file with the same extension.

2.3 Run a carac

Now that your Spice circuit files are properly organized, your first *carac* is now ready to run: you are now able to automatize the execution of various *simulations* while altering both *corner* and *param* libraries to load.

Note that openCarac will not include every *simulation* files located in the same folder by itself, only you can choose the list of these simulations. So, in order to choose *corner* and *param* libraries to load for both *model* and *libparam* and the *simulations* to run, the last step is to add a *configuration file* in the same folder as your Spice main file; the name of such a file is, by default, set to `carac.conf`. Its syntax is described in chapter openCarac usage.

Supposing that we want to run a *carac* with the previous Spice control file where we want to alter both *corner* and *param* libraries and run all of the simulations; let's pretend that *corner* libraries names are as follows:

- TYPICAL
- WORST
- BEST

and *param* libraries are:

- TYP
- VDD_IBIAS_TEMP (where supply voltage, bias current and temperature vary by steps)

Then the `carac.conf` *configuration* file may look like as follows (details about the syntax are given in chapter openCarac usage):

```
## openCarac configuration file example

.CARAC myCarac

.CORNER  TYPICAL WORST BEST
.PARAM   TYP VDD_IBIAS_TEMP
.SIMU    op openLoop transient noise
```

To run your *carac* through your command line interface, get to the folder where your `carac.conf` file is located and run openCarac main executable according to your command as described in section Add openCarac to your environment; all the informations about the execution are printed in the standard output or the log file you have specified, openCarac will execute your simulator, extract the results and then exit.

When executing, openCarac will run the four *simulations* op, openLoop, transient and noise by including the files and alter *corner* and *param* libraries. So, since there are in this case 4 *simulations*, 3 *corners* and 2 *params*, there will be $4 \times 3 \times 2 = 24$ *runnings*.

As soon as every *running* is terminated, openCarac parses your simulator results files looking for the values of measures (data values extracted using a `.MEAS` or `.MEASURE` directive) and creates HTML files, LaTeX code and GNU Octave scrips. These files are saved into a `openCaracFiles` folder created next to your `carac.conf` file.

If you can see these results files, it seems that you have perfectly integrated openCarac into your environment and that you are now able to automatize your simulator *runnings*. Congratulations!

Detailed informations about openCarac execution and its configuration are given in both chapters Execution details openCarac usage.

Chapter 3

Execution details

Abstract: In this chapter is described how does openCarac mainly work. Section Overview gives an overview about its execution; section Native functions describes native functions that can be used to do more than a simple results extraction; section In batch mode details openCarac execution when batch mode is activated and section Through the graphical user interface shows what can be done using the GUI.

3.1 Overview

Since openCarac is initially used to execute your favorite simulator many times whilst altering libraries loadings and files inclusions using Spice directives, relative path links to access these files must be preserved; and, since the simulator executions may take some time, it can be useful to keep access to your files when having a *running* in progress.

As a consequence, choice has been made to create temporary folders to process the *runnings* into when openCarac is executed: copies of necessary files are created in these folders located next to the *configuration* directory. In such a case, relative links to access Spice libraries and included files remain corrects[1]. See chapter Files alteration for more informations about the creation of temporary folders.

Note that if you have a lot of *runnings*, every temporary folders are created first and then the simulator is executed into each one of them. These directories are not created one at the time in order to keep the capability to process the *runnings* simultaneously with different computers; a solution to do so requires the alteration of hooks. A description of hooks is given in chapter Expert mode; openCarac also comes with an API example detailing how to make it wait for the end of executions in background.

When creating these temporary folders, openCarac modifies your Spice files to alter library loadings and file inclusions. Note that you can also alter parameter values when defining your *simulation* as described in subsection .SIMU directive. It uses the markers that are available in your main file to select the line to alter (see section Organize your Spice circuit files), only lines following these keywords are modified. So, beware your Spice inclusion directive locations before running your *carac* and make sure that they are always following a keyword if they are to be altered; otherwise, more than one file inclusion could appear and your *runnings* may not work properly.

Your simulator command which has been defined in your openCarac *application* environment file (see section Add openCarac to your environment) is executed whilst passing as argument the copy of your main file for each temporary folder, outputs are redirected into log files. When every *runnings* have been proceeded, these files are being parsed to get the values of your measures (a measure is what is calculated by your simulator through a `.MEAS` or `.MEASURE` directive) and saved into an `openCaracFiles` folder created in the *configuration* directory; then temporary folders are deleted. Gathered informations are used for the results extraction, i.e. creation of output files such as HTML or LaTeX tables and GNU Octave ^{TM} scripts, also located in the `openCaracFiles` folder.

3.2 Native functions

openCarac has originally been created to treat the simulator data and generate output files. Custom functions can be created using the API to treat the same data other ways, but openCarac also comes with native functions. These functions are described bellow:

[1] Of course, this also work with absolute paths.

3.2.1 Archives creation

When running openCarac while activating the "archive creation mode", such as when using the `--archive` option (see section Execution options for more informations about execution options), archive folders are created after results extraction, each of them containing every files openCarac needs to run; i.e. every files to copy in temporary folders. Also, if having *model* and *libparam* loaded in the *configuration*, both library files are also saved into the archive folder. Two folders with the same names as the directories containing these library files are created and the *model* and *libparam* files are copied into them. The name of the archive folders is formated using keyword `archive`, the date and time of their creation, the name of both library files and their parent directories.

Also, openCarac output files are created in it: these files are re-formated in order to contain every informations and results from various *caracs* in the same *configuration*. Note that an archive folder is created for each group of *caracs* with the same *model* and *libparam* files in the same *configuration*.

3.2.2 Full summary creation

When running openCarac while activating the "full summary creation mode", such as when using the `--fullsummary` option (see section Execution options for more informations about execution options), a full summary file is created after results extraction. In such a file can be found a tabular summarizing every results having a `checkmeas` in a *carac*. Each result having the same name will have its value printed in the same line of the tabular; columns are separated function of their library files, i.e. their *model* and *libparam*, in the *carac* they are extracted from.

This summary file is created in the current directory, i.e. where openCarac has been executed. The names of these files are formated using keyword `opencarac_Check_Summary` and the date and time of their creation.

3.3 In batch mode

When calling openCarac main executable and activating "batch mode", a whole sequence is processed without human intervention.

First, the openCarac environment is loaded and any potential argument is parsed. Every *configuration* file to load is then opened and parsed.

In case "extractres mode" is activated, results are extracted without executing the simulator for any potential *running*. Otherwise, for each loaded *configuration*, the simulator is executed for every *running* before results extraction. There are two possibilities:

- either "run by step mode" is activated: for each *carac* in each *configuration*, the simulator is executed for every *runnings* and, for the whole *carac*, temporary folders are deleted unless "debug mode" is activated; then results are extracted.
- or "run by step mode" is deactivated: for each *carac* in each *configuration*, the simulator is executed for every *runnings*. After having executed the simulator with any potential *running*, for every *carac* of every *configuration*, temporary folders are deleted unless "debug mode" is activated; then results are extracted.

If "archive creation mode" is activated, archives are then created for each *configuration* as described in subsection Archives creation.
If "full summary creation mode" is activated, a full summary is then created as described in subsection Full summary creation.

3.4 Through the graphical user interface

openCarac graphical user interface is not yet implemented. This will be available in a future release.

Chapter 4

Files alteration

Abstract: In this chapter is described the procedure openCarac follows to tune simulator files to have them ready for characterization. Section Overview gives an overview of the files alteration, section Creation of temporary folders details the necessity of creation of temporary folders; both sections Substitution after a marker and Replacing netlist inclusion show how the main file is altered, sections Substitution of parameters and Add parameters settings describe the parameters settings, section Comment of possible inclusions details the "comment of possible inclusions" mode and section Check mode details the alterations in check mode.

4.1 Overview

First of all, know that openCarac limits its file alteration to every files having its extension matching one of the "files extension filter" (see subsection filesExtensionFilter for more informations) in the *configuration* folder hierarchy. In order to proceed to the substitutions, openCarac also has to read these files looking for patterns to substitute; such an operation may lead to execution time increase if the matching files contain a lot of lines. As a consequence, it is highly recommended to move to parent folders any huge file that does not contain data to substitute.

Also, one must know that Spice simulator files syntax changes from one simulator to an other; thus, it remains impossible to have a file parser in openCarac as it might not be compatible with every simulator. As a consequence, openCarac has its own way to process the substitutions.

In every case, substitutions are performed line by line and openCarac is not capable of detecting data defined on various lines. Indeed, some Spice simulators support (just like openCarac *configuration* file parser) the possibility to add a plus (+) at the beginning of a line to make it the continuation of the previous one. Such a syntax is not supported by openCarac for substitutions in Spice simulator files.

openCarac detects keywords in the files that are parsed, since it has been made for substituting in Spice simulator input files, keywords are usually searched at the beginning of the line. The definition of the simulator keywords, as described in subsection Simulators sections, allows to define what they are. openCarac uses space character as a delimiter between different elements; so, when a substitution occurs, spaces are added between the elements.

4.2 Creation of temporary folders

While openCarac executes the Spice simulator for each *running*, it would be nice to be able to continue working in the *configuration* folder; also, making possible to execute the Spice simulator for various *runnings* at the same time could be great. For these reasons, choice has been made to create temporary folders next to the *configuration* directory: indeed, relative file paths remain correct since the hierarchy is identical and more than one temporary folder can be created at once. So, when executing the simulator for a specific *running*, openCarac starts by creating this temporary folder and the simulator is executed whilst loading the copy of the main file in this directory.
Note that, when creating temporary folders, not every file from the *configuration* folder is copied in them, only the ones having their extension matching one of the "files extension filter", see subsection filesExtensionFilter for more informations. The hierarchy in the temporary folder remains identical. Also, every duplicated file is parsed by openCarac looking for substitution patterns according to the main file markers and simulator files syntax.

4.2.1 Simulator files syntax

When temporary folders are created, openCarac substitutes various lines according to the *running* settings (*model*, *libparam*, *corner*, *param*...) and the simulator files syntax. Depending on the selected simulator, substitutions might be case sensitive (see subsection isCaseSensitive for more informations) and depend on the Spice directives (`.INC`, `.LIB`, `.PARAM`) as defined in the openCarac environment (see subsections incDirective, libDirective, paramDirective and paramEquality for more informations). Comment syntax and string delimiter might be added by openCarac when substituting and are also simulator dependent (see subsection commentSyntax and stringDelimiter for more informations).

4.3 Substitution after a marker

For each marker in the main file, openCarac substitutes the line following it by setting the directive, the file path surrounded by the simulator string delimiter and the parameter (in case of the "simu marker", no parameter is added; the condition of its existence is ignored). If the file path is not defined, openCarac prints a warning and, on the following line, adds the directive and the parameter at the first and third position while keeping the second element unchanged, the end of the line is also copied identically. If the parameter is not defined, openCarac prints a warning and, on the following line, adds the directive and the file path surrounded by the simulator string delimiter while keeping the end of the line unchanged. If neither the file path or the parameter is defined, no substitution occurs after the marker.

After the "model marker", openCarac adds the simulator `.LIB` directive, the *carac* selected *model* surrounded by the simulator string delimiter and the *running* selected *corner*.

After the "param marker", openCarac adds the simulator `.LIB` directive, the *carac* selected *libparam* surrounded by the simulator string delimiter and the *running* selected *param*.

After the "simu marker", openCarac adds the simulator `.INC` directive and both the selected *simulation* name and the "simulation file extension" surrounded by the simulator string delimiter.

Note: Substitutions after a marker only occur in the main file.

4.4 Replacing netlist inclusion

openCarac replaces a netlist inclusion in the main file. A replacement occurs when only one line matching a netlist inclusion is found, otherwise an error message is printed and no replacement occurs. A netlist inclusion is detected when a line starts with the simulator `.INC` directive and contains the "netlist file extension". Case sensitivity for the detection of the simulator `.INC` directive depends on the simulator settings.

Note: Netlist inclusion replacement only occurs in the main file.

4.5 Substitution of parameters

When parameters are to be substituted, openCarac proceeds to the parameter alteration in every file that is copied in the temporary folder. A parameter substitution can only occur on a line starting with the simulator `.PARAM` directive. To substitute a parameter, the line must contain the following pattern: the parameter name followed by zero or more spaces, followed by the simulator parameter equal delimiter, followed by either "one or more spaces and a non-space string" or "no space and a string that does not start with any character of the simulator parameter equal delimiter". In such a case, the detected string is substituted by the parameter value.

Case sensitivity for the detection of the simulator `.PARAM` directive, the parameter name and the simulator parameter equal delimiter depends on the simulator settings.

4.6 Add parameters settings

If, after having parsed every file to copy in the temporary file folder, some parameters have not been substituted, openCarac proceeds to the parameters settings addition. For each parameter name that has not been substituted, a warning is printed and a line is added at the end of the *simulation* included file: the line starts with the simulator `.PARAM` directive and is

followed by the parameter name, the simulator parameter equal delimiter and the parameter value; elements are separated by spaces.

Case sensitivity for the detection of the parameter name depends on the simulator settings.

Note: Parameter setting additions only occur in the *simulation* included file.

4.7 Comment of possible inclusions

When openCarac "comment of possible inclusions" is activated, in the main file, the simulator comment syntax is added at the beginning of every line starting with the simulator `.INC` directive and containing the name of a possible *simulation* followed by the *simulation* file extension. The simulator comment syntax is also added at the beginning of every line starting with the simulator `.LIB` directive and containing the file tail of a possible *model* or *libparam*. A possible *simulation* name is the name of a *simulation* defined in a *carac* of the same *configuration*. A possible *model* or *libparam* is a *model* or *libparam* defined in a *carac* of the same *configuration*.

Case sensitivity for the detection of simulator `.INC` and `.LIB` directives, *simulation* name, *simulation* file extension, *model* or *libparam* file tails depends on the simulator settings.

Note: Comments of possible inclusions only occur in the main file.

4.8 Check mode

When openCarac "check mode" is activated, in every file that is copied in the temporary folder, the simulator comment syntax is added at the beginning of every line starting with one of the element of the "to remove in check mode" list of the selected simulator, see subsection toRemoveInCheckMode for more informations.

Case sensitivity for the detection of elements depends on the simulator settings.

Chapter 5

Simulator specificities

Abstract: In this chapter is described the integration of Spice simulators in openCarac. Section Execution of the simulator details the execution of third party software components in openCarac, section Saving the results describes the calling of the parsers and the saving of the results while sections Gnucap files parser, Ngspice files parser, Xyce files parser and Smash files parser describe the behaviour of native files parsers in openCarac; then section User defined command shows how to replace the execution of a native Spice simulator by a custom command.

5.1 Execution of the simulator

During its own execution, openCarac calls external commands with a copy of the main file in a temporary folder as an argument. Note that the execution of this command is done from the current directory unless the simulator "directory change" attribute is activated (see subsection changeDirectory for more informations); in such a case, openCarac proceeds to a directory change into the temporary folder before executing the command and changes back to the previous directory right after.

If "custom execution mode" is not activated, whatever the selected simulator, openCarac executes its command and collects informations the same way. As a consequence, the description given in this section remains correct for every Spice simulator or command executed through openCarac.

If "custom execution mode" is activated, a custom procedure is used to execute the Spice simulator; a description of how to tune it is given in chapter Expert mode. What is described in this section does not apply to this mode.

When executing the Spice simulator, openCarac uses settings as defined in its environment and the path to access the main file. In these settings, one can find the "command", the "check options" and the "run options". See subsections command, checkOptions and runOptions for more informations. If "check mode" is activated, the executed command is the concatenation of the Spice simulator "command", the "check options" and the absolute path to to access the main file. If "check mode" is not activated, the executed command is the concatenation of the Spice simulator "command", the "run options" and the absolute path to to access the main file.

In both cases, the concatenation is evaluated by openCarac. So, in order to make sure that the execution is done correctly, escape syntax must be used for spaces and `"  { } \ $ [ ]` characters.

During the execution, what is returned in the standard output or the error output by the Spice simulator is caught by openCarac and written in a log file. The log file is defined with the same root name as the main file and the simulator "log file extension", see subsection logExtension for more informations. Note that whatever is returned through a graphical user interface cannot be caught by openCarac.

If "monitor error code" attribute is activated, as defined in subsection monitorErrorCode, openCarac prints an error if the code returned by the Spice simulator is not equal to 0.

5.2 Saving the results

After having executed the Spice simulator, openCarac proceeds to a results saving: it calls the appropriate native files parser to extract measurements and, if "files copy" is activated (see subsection filesCopy for more informations), it copies every file from the temporary folder having their extension matching one of the simulator "save filter" list (see subsection saveFilter for more informations) into the `openCaracFiles` folder in the *configuration* directory.

Files parsing depends on the Spice simulator: openCarac is natively compatible with some of them, its behaviour is described in the following sections.

5.3 Gnucap files parser

openCarac files parser for Gnucap is identical for every file that is found in the temporary folder with the "log file extension"; these files are sorted using dictionary mode to define the parsing order.

Step name: If the parsed file has the same root name as the main file, the step name has no prefix; otherwise, relative path to access the parsed file in the temporary folder is used as a prefix for the step name. In a single file, the step number is defined through an integer that is incremented each time a hash (#) is encountered and is used as the step name; if this hash is followed by a keyword (time, freq), this keyword is used as a suffix for the step name.

Checkop: If, in a line starting with a hash (#) with no keyword (time, freq), is available a string `V(theNet)` with "theNet" matching a *checkop* pattern, openCarac extracts its value in the following lines.

Extractop: If, in a line starting with a hash (#) with no keyword (time, freq), is available a string `theParam(theDevice)` with "theParam" matching an *extractop* pattern and "theDevice" matching an *extractop* filter pattern, if any, openCarac extracts its value in the following lines.

Measures: For each line having an equal sign (=) in it, openCarac considers it as a measure; characters from the beginning of the line to the equal sign are used as the measure name, characters from the equal sign to the end of the line or an other equal sign are used as the value. In case of measures, openCarac automatically converts mnemonics to scientific notation.

Running state: openCarac considers that an error occurred if one of the following patterns is found in the parsed files:

- `^ ?`
- `did not converge`

Guidelines:

- Gnucap supports both `.PARAM` directive with and without equal sign (=); however openCarac is set, both syntaxes cannot be used at the same time in order to permit parameter substitutions.
- By default, Gnucap uses mnemonics for operating point extracted values; for openCarac not to print "not a number" warnings, scientific notation must be used using Gnucap "Basic" option: `.OP BASIC`.

5.4 Ngspice files parser

openCarac files parser for Ngspice is identical for every file that is found in the temporary folder with the "log file extension"; these files are sorted using dictionary mode to define the parsing order.

In Ngspice log file parser, openCarac reads the file line by line and identifies categories, this identification is not case-sensitive. Parsing ends when pattern `total elapsed time` is encountered. Lines containing one of the following pattern are ignored:

- `trying gmin`

- `failed`
- `note:`

The category can take different values: "initial" (used for measure and *checkop* extractions), "analysis" (used for measure extractions), "device" (used for *extractop* extractions) and "legend" (where informations are ignored by openCarac). openCarac considers that the category is "initial" when keyword `node voltage` or `doing analysis` is encountered; it considers that the category is "analysis" when keyword `analysis` is encountered; it considers that the category is "device" when keyword `device` is encountered. The rest of the file is considered as "legend" when one of the following pattern is found:

- `models`
- `legend:`
- `index`
- `total cpu time:`

Step name: If the parsed file has the same root name as the main file, the step name has no prefix; otherwise, relative path to access the parsed file in the temporary folder is used as a prefix for the step name. In a single file, the step name is defined through an integer that is incremented each time pattern `doing analysis` is encountered.

Checkop: When parsing lines in the "initial" category, if the line contains two elements while the first one matches a *checkop* pattern and the second one is a double, openCarac extracts its value according to the *checkop* patterns list.

Extractop: When parsing lines in the "device" category, devices names are saved from parsing of the line containing the `device` keyword: from columns 2, 3 and 4 in the same line; the element in the first column is considered as the parameter name and elements in columns 2, 3 and 4 are considered as the parameter value for the previously saved device name. If the parameter name matches an *extractop* pattern and the device name matches an *extractop* filter pattern, if any, openCarac extracts its value. An exception is made in case the parameter value is not a double and the parameter name differs from "model": in such a case, the value is ignored.

Measures: When parsing lines in the "initial" or "analysis" category, if the second element of the line is an equal sign (=), openCarac considers it as a measure; the first element is used as the name and the third element is used as the value.

Running state: openCarac considers that an error occurred if one of the following patterns is found in the parsed files:

- `error on line`
- `ngspice stopped due to error`
- `fatal error in ngspice`

5.5 Xyce files parser

openCarac has two different files parsers for Xyce: one of them is dedicated to parse every file that is found in the temporary folder with the "log file extension"; the other one parses every file that is found in the temporary folder with `.ic` extension. In both cases, files with the same extension are sorted using dictionary mode to define the parsing order.

In Xyce log file parser, openCarac reads the file line by line and identifies categories, this identification is not case-sensitive. The category can take different values: "operating point information" (used for *extractop* extractions), "measure functions" (used for measure extractions) and "legend" (where informations are ignored by openCarac). openCarac considers that the category is "operating point information" when keyword `operating point information` is encountered; it considers that the category is "measure functions" when keyword `measure functions` is encountered. The rest of the file is considered as "legend" when the following pattern is found: `*****`.

In Xyce `*.ic` files parser, openCarac reads the file line by line and identifies keywords at the beginning of the line. Informations located after a keyword `.ic` or `.nodeset` are used for *checkop* data extraction.

Step name: If the parsed file has the same root name as the main file and has the "log file extension", the step name has no prefix; otherwise, relative path to access the parsed file in the temporary folder is used as a prefix for the step name. In a single log file, the step name is defined through an integer that is incremented each time pattern `operating point information:` is encountered, this number follows the prefix (if any) after an underscore (_).

Checkop: *Checkop* data is extracted by the `*.ic` file parser. For each line in a `*.ic` file, if the first element matches the string `.ic` or `.nodeset` (matching is not case sensitive), every data located after this pattern is split from equal signs (=), the first part represents the node voltage name and the second part represents its value. If the first parts matches the string `V(theNet)` with "theNet" matching a *checkop* pattern, openCarac extracts its value.

Extractop: *Extractop* data is extracted by the log file parser. When parsing lines in the "operating point information" category: if, in a line starting with the `name` keyword, is available a string matching an *extractop* filter pattern, if any, the element index is remembered by openCarac for a value extraction in the following lines; if, in a line starting with a keyword matching an *extractop* pattern, an element index has been remembered by openCarac for a value extraction, openCarac extracts its value. The *extractop* name is formated as `theParam(theDevice)` with "theParam" matching the *extractop* pattern and "theDevice" the remembered string matching an *extractop* filter pattern. Note that some incorrect behaviour may happen since Xyce output sometimes lacks of spaces (tested with version 6.3.).

Measures: Measures are extracted by the log file parser. When parsing lines in the "measure functions" category, for each line having an equal sign (=) in it and not having pattern `measure start time`, openCarac considers it as a measure; characters from the beginning of the line to the equal sign are used as the measure name, characters from the equal sign to the end of the line or an other equal sign are used as the value.

Running state: openCarac considers that an error occurred if one of the following patterns is found in the parsed log files:

- `xyce abort`
- `simulation aborted due to error`

5.6 Smash files parser

openCarac has three different files parsers for Smash: one of them is dedicated to parse every file that is found in the temporary folder with the "log file extension"; an other one parses every file that is found in the temporary folder with `.mes` extension; the third one parses every file that is found in the temporary folder with `.op` extension. In each case, files with the same extension are sorted using dictionary mode to define the parsing order.

In Smash log file parser, openCarac reads the file line by line looking for specific keywords to define the *running* state.
In Smash `*.mes` files parser, openCarac reads the file line by line looking for a pattern to detect measure values.
In Smash `*.op` files parser, openCarac reads the file line by line and identifies categories, this identification is not case-sensitive. The category can take different values: "nets" (used for *checkop* extractions), "sources" (where informations are ignored by openCarac) and "devices informations" (used for *extractop* extractions). openCarac considers that the category is "nets" when keyword `nets:` is encountered; it considers that the category is "sources" when keyword `sources:` is encountered and that the category is "devices informations" when one of the following patterns is found:

- `* (m) mos transistors`
- `* (c) capacitors`
- `* (d) diodes`
- `* (l) inductors`
- `* (q) bipolar transistors`
- `* (r) resistor`

Step name: If the parsed file has the same root name as the main file and has a `.op` extension or the same root name as the main file and two extensions while the second one is `.mes`, then the step name has no prefix; otherwise, relative path to access the parsed file in the temporary folder is used as a prefix for the step name. In a single `.op` file, nothing is added to the prefix; if there is no prefix, it is then set to 0. In a single `.mes` file, if the measure name contains an opening bracket (`[`) and finishes with a closing bracket (`]`), then what is located between the brackets is considered as the step value and is removed from the step name; otherwise, the step value is set to 0.

Checkop: Checkop data is extracted by the `*.op` files parser. When parsing lines in the "nets" category, if the line contains more than two elements while the second one is an equal sign (=) and the third one is a double, openCarac extracts its value according to the *checkop* patterns list.

Extractop: When parsing lines in the "device" category, devices names are defined as the first element of a line containing keyword `multiplicity`; then, the line is considered as a list of parameters definitions separated by comas (,), each parameter definition being composed by the parameter name, an equal sign (=) and the parameter value. If the parameter name matches an *extractop* pattern and the device name matches an *extractop* filter pattern, if any, openCarac extracts its value.

Measures: Measures are extracted by the `*.mes` files parser. For each line in a `*.mes` file, if it matches pattern `*? = ?*` (see TCL `string match` command matching rules for more informations), openCarac considers it as a measure; characters from the beginning of the line to the equal sign are used as the measure name, characters from the equal sign to the end of the line or an other equal sign are used as the value.

Running state: openCarac considers that an error occurred if one of the following patterns is found in the parsed log files:

- `error`
- `load failed`

Guidelines:

- By default, Smash uses mnemonics for measure values; for openCarac not to print "not a number" warnings, scientific notation must be used by calling Smash directive: `.DIGITS 4 ENG=NO`.

5.7 User defined command

If none of the natively compatible simulators is to be used, openCarac also allows to set a user defined command as the simulator. Its setting and execution remain identical with any natively compatible simulators. The main difference is that no files parser is available and no result extraction can occur by default.

On the other hand, it remains possible to customize openCarac hooks to proceed to files parsing. Alteration of hooks is shown in chapter Expert mode, openCarac also comes with an API example detailing how to make it parse files and add measures.

Chapter 6

openCarac usage

Abstract: In this chapter is described how to customize openCarac environment and its execution. Section Environment file syntax describes the environment file syntax, i.e. how to change openCarac settings and default behaviour; section Execution options lists execution options that overload openCarac default behaviour and section Configuration file syntax gives the syntax to set the *configurations* and impact on *caracs*, *simulations*, *runnings*, etc.

6.1 Environment file syntax

In the openCarac environment file is defined how openCarac execution is performed and what is the simulator files syntax so that substitutions occur correctly. This file is located in a `.openCarac` folder in the user home directory.

openCarac environment file format is based on the informal standard for *configuration* files: it is a simple text file with basic structure composed of sections, properties and values.

The basic element contained in the environment file is the *key* or *property*. Every key has a name and a value, delimited by an equals sign (=). The name appears to the left of the equals sign.

Keys are grouped into named sections. The section name appears on a line by itself, in square brackets ([and]). All keys after the section declaration are associated with that section. There is no explicit "end of section" delimiter; sections end at the next section declaration, or the end of the file. Hashes (#) indicate the beginning of a comment. Comment lines are ignored. Hashes can be used in a key value as soon as they are escaped using a backslash (\#).

6.1.1 Section application

In the application section of the environment files is defined openCarac default behaviour about various properties, some of them can be overloaded when calling openCarac main executable using Execution options.

6.1.1.1 modelMarker

For openCarac to detect the main file, this model marker must be printed in it. When creating the temporary folders, *model* and *corner* substitution is performed on the line following this model marker. Model marker must be a string that is not equal, without case sensitivity, to param marker and simu marker. It should starts with a star ($*$) to be a Spice comment; otherwise, a warning is printed by openCarac.

6.1.1.2 paramMarker

For openCarac to detect the main file, this param marker must be printed in it. When creating the temporary folders, *libparam* and *param* substitution is performed on the line following this param marker. Param marker must be a string that is not equal, without case sensitivity, to model marker and simu marker. It should starts with a star ($*$) to be a Spice comment; otherwise, a warning is printed by openCarac.

6.1.1.3 simuMarker

For openCarac to detect the main file, this simu marker must be printed in it. When creating the temporary folders, openCarac *simulation* file inclusion is substituted on the line following this simu marker. Simu marker must be a string that is not equal, without case sensitivity, to model marker and param marker. It should starts with a star (*) to be a Spice comment; otherwise, a warning is printed by openCarac.

6.1.1.4 filesExtensionFilter

When creating a new or opening an existing *configuration*, openCarac parses the found files in the *configuration* folder hierarchy. Not all the files are used by openCarac: only the ones having an extension matching one of the extensions filtered by the "files extension filter" attribute. Files extension filter must be a list of strings, each of them being a single word starting with a dot (.), openCarac automatically converts them to lower case.

6.1.1.5 simulationFileExtension

When adding a *simulation* to a *carac*, openCarac automatically verifies that the *simulation* file exists. The *simulation* file must have an extension equal to this "simulation file extension". Simulation file extension must be a string, not a list itself, of at least two characters and starting with a dot (.). If the *simulation* file extension does not appear in the "files extension filter", it is automatically added.

6.1.1.6 netlistFileExtension

When adding a *netlist* to a *carac*, openCarac automatically verifies that the *netlist* file exists. The *netlist* file must have an extension equal to this "netlist file extension". *Netlist* file extension must be a string, not a list itself, of at least two characters and starting with a dot (.). If the *netlist* file extension does not appear in the "files extension filter" attribute, it is automatically added.

6.1.1.7 archive

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When archive creation is activated, openCarac main executable proceeds to archives creation for each loaded *configuration* after results extraction.

6.1.1.8 fullsummary

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac full summary creation is activated, openCarac main executable proceeds to full summary creation after results extraction and archives creation.

6.1.1.9 filesCopy

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac simulator files copy is activated, when parsing simulator results files of a *running*, simulator files are copied into the `openCarac_Files` folder located in the *configuration* directory. The list of files to copy is filtered by extension, every file having its extension matching one the patterns defined in the "save filter" attribute of the simulator is selected for copy, see subsection saveFilter for more informations.

6.1.1.10 runBySteps

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "run by step mode" is activated, openCarac main executable deletes temporary folders for each *running* and extracts results after every simulator execution for each *carac* instead of doing it for the whole *configuration*.

6.1.1.11 custom

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "custom execution mode" is activated, custom procedure `openCarac_customRunSimulator` is called to run the simulator instead of using openCarac default behaviour. See section Custom procedure for more informations.

6.1.1.12 commentOfPossibleInclusions

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "comment of possible inclusions" is activated, when creating *runnings* temporary folders, openCarac modifies the main file to comment any *model* / *libparam* library selection or *simulation* file inclusion. A library selection is a line starting with the simulator "lib directive" and containing the tail of a possible *model* or *libparam*, i.e. a *model* or *libparam* defined in a *carac* of the same *configuration*. A *simulation* file inclusion is a line starting with the simulator "inc directive" and containing the name of a possible *simulation*, i.e. a *simulation* defined in a *carac* of the same *configuration*, followed by the *simulation* file extension. When commenting in a file, the simulator "comment syntax" is added at the beginning of the line.

6.1.1.13 creationOfHtmlFiles

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "creation of html files" is deactivated, no HTML file is generated by openCarac. This selection affects the behaviour of results extraction and archive creation.

6.1.1.14 creationOfLatexFiles

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "creation of latex files" is deactivated, no LaTeX file is generated by openCarac. This selection affects the behaviour of results extraction and archive creation.

6.1.1.15 creationOfOctaveFiles

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When openCarac "creation of octave files" is deactivated, no GNU Octave ^{TM} file is generated by openCarac. This selection affects the behaviour of results extraction and archive creation.

6.1.1.16 simulator

This attribute is a string corresponding to the name of the simulator that is selected by default by openCarac. Its value is automatically converted to lower case. When creating a *carac*, the default value for its "simulator" attribute, before being overloaded, is identical to this "default simulator" attribute.

6.1.1.17 logfile

This attribute may be a file path or set to one of the two values *stdout* and *stderr*. If "log file" attribute is set to a file path, then messages are printed into the given file. If "log file" attribute is set to *stdout*, then messages are printed in the standard output. If "log file" attribute is set to *stderr*, then messages are printed in the error output.

6.1.2 Simulators sections

In every simulator section is defined the syntax to tune the files or execute the command for the appropriate simulator. The section name can be one of the followings; definition of the properties remain correct for each of them:

- ngspice
- gnucap
- xyce
- smash
- command

6.1.2.1 command

This sets the command to execute the simulator. This command is executed by openCarac through the TCL exec command if openCarac "custom execution mode" boolean is not activated. It is concatenated with either "run options" or "check options" depending on the value of openCarac "check mode" boolean.

6.1.2.2 checkOptions

The simulator command is executed by openCarac through the TCL exec command if openCarac "custom execution mode" boolean is not activated. If openCarac "check mode" boolean is activated, the command is concatenated with the value of this "check options" attribute.

6.1.2.3 runOptions

The simulator command is executed by openCarac through the TCL exec command if openCarac "custom execution mode" boolean is not activated. If openCarac "check mode" boolean is deactivated, the command is concatenated with the value of this "run options" attribute.

6.1.2.4 saveFilter

This must be a list of strings, each of them being a single word starting with a dot (.), openCarac automatically converts them to lower case. When calling openCarac files parser, if openCarac "simulator files copy" boolean is activated, a copy of files having their extension matching a pattern in this list is performed from the temporary folder into the `openCaracFiles` directory in the *configuration* file folder. Matching follows the rules of TCL "string match" command without case-sensitivity.

6.1.2.5 toRemoveInCheckMode

This must be a list of patterns that are not lists themselves and are not empty strings. When having openCarac "check mode" boolean activated, openCarac aims to quickly verify that no error would occur when executing the simulator. To make sure that a simulator check does not take too much time, some lines from the files to copy in the temporary folders can be removed. When creating the temporary folders, files are copied and, if openCarac "check mode" boolean is activated, any line starting with a pattern from this list is substituted and the simulator comment syntax is added at the beginning of the line. Matching follows the rules of TCL "string equal" command ; case sensitivity depends on the simulator "case sensitivity" attribute

6.1.2.6 logExtension

This sets the log file extension to print what is returned by the simulator command in. The command is executed by open↩ Carac through the TCL exec command if openCarac "custom execution mode" boolean is not activated. What has been printed by the command is caught by openCarac and written in a file having the same root name as the main file in the temporary folder and this "log extension". Log file extension must be a non-empty string, not a list itself, of at least two characters and starting with a dot (.), openCarac automatically converts it to lower case. If the log file extension does not appear in the "save filter" attribute of the simulator, it is automatically added.

6.1.2.7 isCaseSensitive

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When creating a temporary folder, files are copied, substitutions occur and files are included or loaded. In order to know what to substitute, this case sensitivity is used for matching. This also affects openCarac files parser: case is not sensitive to add measures but the simulator case sensitivity is taken into account to filter devices or net names.

6.1.2.8 changeDirectory

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When executing the simulator command, depending on the simulator behaviour, files inclusion can either be relative to the file they are included in or to the directory it has been executed in. If "directory change" attribute is activated, before executing the simulator command, openCarac performs a changing of directory so that files inclusion are also relative to the directory it has been executed in. After having executed the command, openCarac changes back the previous location.

6.1.2.9 monitorErrorCode

This property is a boolean, its value must be set to "yes" for activation and "no" for deactivation. When executing the simulator command, if "custom execution mode" is not activated, an error code is returned and openCarac can monitor it.

If "monitor error code" attribute is activated, openCarac prints an error if the execution of the simulator command returns a non-zero error code. Otherwise, the returned error code is ignored by openCarac.

6.1.2.10 commentSyntax

Its value must be a non-empty string that is not a list. When creating a temporary folder, files are copied and substitutions occur. In case a line must be removed by openCarac, the "comment syntax" is added at its beginning.

6.1.2.11 incDirective

Its value must be a non-empty string that is not a list ; also, must be is different from the simulator "lib directive" and simulator "param directive". To define it, case sensitivity depends on the simulator "case sensitivity" boolean attribute. It is expecting a syntax based on the Spice `.INC` directive to define how substitutions are performed by openCarac. When creating a temporary folder, files are copied, substitutions occur and *simulation* or *netlist* files are included. To substitute, openCarac considers that a line matches a file inclusion when it starts with this "inc directive". To include a *simulation* or *netlist* file, this "inc directive" is added at the beginning of the line.

6.1.2.12 libDirective

Its value must be a non-empty string that is not a list ; also, it must be different from the simulator "inc directive" and simulator "param directive". To define it, case sensitivity depends on the simulator "case sensitivity" boolean attribute. It is expecting a syntax based on the Spice `.LIB` directive to define how substitutions are performed by openCarac. When creating a temporary folder, files are copied, substitutions occur and *model* or *libparam* files are loaded. To substitute, openCarac considers that a line matches a file loading when it starts with this "lib directive". To load a *model* or *libparam* file, this "lib directive" is added at the beginning of the line.

6.1.2.13 paramDirective

Its value must be a non-empty string that is not a list ; also, it is different from the simulator "inc directive" and simulator "lib directive". To define it, case sensitivity depends on the simulator "case sensitivity" boolean attribute. It is expecting a syntax based on the Spice `.PARAM` directive to define how substitutions are performed by openCarac. When creating a temporary folder, files are copied, substitutions occur and parameters values are tuned. To substitute, openCarac considers that a line matches a parameter setting when it starts with this "param directive" and that the "param equality" is located between its name and its value. To set a parameter that has not been found in the files, this "param directive" is added at the beginning of the line.

6.1.2.14 paramEquality

It is the string located between a parameter name and its value to match the syntax based on the Spice `.PARAM` directive and define how substitutions are performed by openCarac. When creating a temporary folder, files are copied, substitutions occur and parameters values are tuned. To substitute, openCarac considers that a line matches a parameter setting when it starts with the "param directive" and this "param equality" is located between the parameter name and its value. To set a parameter that has not been found in the files, this "param equality" is added between the parameter name and its value.

6.1.2.15 stringDelimiter

Its value must be an empty string or a single character. When creating a temporary folder, files are copied, substitutions occur and files are included or loaded. In each case of path substitution, simulator "string delimiter" is used before and after the path addition.

6.2 Execution options

When executing a TCL interpreter, it is possible to add a vector of arguments during TCL execution, these arguments can be used to alter openCarac's behaviour. For instance, it has been shown in section Run a carac that openCarac must be executed in a folder containing a *configuration* file; but, it is possible to give as an argument the path to access such a

configuration file as shown bellow:
If you are using Linux:

```
tclsh ~/openCarac/tcl/openCarac.tcl simulation/carac.conf
```

If you are using Windows:

```
C:\tcl\bin\tclsh.exe C:\openCarac\tcl\openCarac.tcl simulation\carac.conf
```

These arguments are defined in a vector named `argv` in TCL. So, in case you are using a TCL interpreter which does not allow to pass arguments during the loading of an openCarac script file, it is possible to set variable `argv` before sourcing the main file. This may look like as shown bellow:

```
set argv "--ngspice [file join {simulation} {carac.conf}]" ; source openCarac.tcl
```

The variable `argv` is in fact a list of arguments separated by spaces: it is then possible to give various arguments to openCarac at the same time when you are executing it. If more than one *configuration* file path is given, all the files will be loaded by openCarac.

More than paths to *configuration* files, you can also add execution options to change openCarac behaviour. An execution option is considered as so when starting with either two minus signs (--) and a keyword or one minus sign (-) and a letter. Note that almost every keyword has its equivalent as a letter, that arguments are case-sensitive and that the order you pass them is relevant for openCarac. Here is a list of every keyword available and their descriptions:

archive (-a): activates "archives creation mode", creates archives after results extraction, see subsection Archives creation for more informations.

batch (-b): activates "batch mode"; openCarac main executable does not try to run graphical user interface and executes the whole batch sequence: see section In batch mode for more informations.

check (-c): activates "check mode": selects "check options" instead of "run options" when executing the simulator, see subsection checkOptions for more informations; remove specific directives from the simulator files, see subsection toRemoveInCheckMode for more informations. Also, number of *runnings* is reduced: only one *running* by *simulation* name and *netlist* couple is kept. Such an option permits to anticipate the functionality of a *carac* without loading every possible *simulations* and reducing the execution time.

command (-o): selects a custom command to execute instead of the simulator selected by your openCarac environment settings, see the description of openCarac environment file in section Add openCarac to your environment for more informations.

configure (-C): makes openCarac create an environment file defining your settings in your home directory. Also creates a `customProcedures.tcl` which you can modify to change openCarac execution behaviour, see chapter Expert mode for more informations. Path to access the environment file is printed in the standard output and openCarac exits.

custom (-t): activates "custom mode": processes the *runnings* using a custom procedure defined in the `customProcedures.tcl` file located in your home directory instead of openCarac default commands. See section Custom procedure for more informations.

debug (-d): activates "debug mode", do not remove temporary folders after processing the *runnings*. As described in chapter Execution details, when executing openCarac, it creates temporary folders next to the *configuration* directory and removes them after the *runnings*; this option disables the deletion of such temporary folders.

extractres (-e): activates "extractres mode": do not execute the simulator and only creates output results files from

data available in the `openCaracFiles` folder in the *configuration* directory. Note that data has to be available by having previously run a *carac* and that extracted results depend on the *configuration*. Such an option permits to create results after having run a *carac* in various passes.

fullsummary (-f): activates "full summary creation" after archives creation, see subsection Full summary creation for more informations.

gnucap (-g): selects Gnucap ^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac environment file in section Add openCarac to your environment for more informations.

help (-h): prints a quick help in the standard output and then exits.

logfile `<arg>`: this option expects an argument, it has the same effect as the logfile property in the environment file. Argument `<arg>` represents the path to this log file; it can be absolute or relative to the location of execution.

multicarac `<arg>`: this option expects an argument. Instead of passing a list of *configuration* files as a vector of arguments, it is possible to print all these paths into a text file and use the `multicarac` option. When using this option, this file is read by openCarac and all the *configuration* files having their paths in it are loaded by openCarac. Argument `<arg>` represents the path to the file containing the locations of each *configuration* file; it can be absolute or relative to the location of execution.

ngspice (-n): selects Ngspice ^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac environment file in section Add openCarac to your environment for more informations.

nofilescopy (-N): deactivates "files copy" after executing the *runnings*, this has the same effect as switching filesCopy environment setting to "no". This might be useful to save disk space in case the simulator files are heavy.

runbysteps (-S): activates "run by step" mode, this has the same effect as switching runBySteps environment setting to "yes". In this case, when executing one *carac*, temporary folders are created, *runnings* are executed, temporary folders are removed and results are extracted; and so on with other *caracs*.

smash (-s): selects Smash ^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac environment file in section Add openCarac to your environment for more informations.

tcl `<arg>`: evaluates a TCL command after loading the openCarac environement. Note that every openCarac API function is available here. See openCarac API reference manual for more informations.

xyce (-x): selects Xyce ^{TM} simulator instead of the one selected by your openCarac environment settings, see the description of openCarac environment file in section Add openCarac to your environment for more informations.

6.3 Configuration file syntax

In order to configure the *runnings*, one has at his disposal what is called a *configuration* file ; by default, openCarac names it `carac.conf`. This file allows to tune the *caracs* by setting the alterations of *corners*, *params* loadings, *netlists* or *simulations* files inclusions. But it also has other characteristics that permit to:

- Run more than one *carac* for a circuit
- Change the path to access libraries files (for *models* or *libparams*)
- Add verifications on measures values.
- Extract devices parameters from your operating point.
- Add verifications on some voltages value at operating point.

6.3.1 Global arrangement

Here is a description of *configuration* files syntax:
Any comment can be added by starting with an hash (#), any text located between an hash and the end of the line is to be ignored. *Configuration* files syntax is not case sensitive. *Caracs* can be tuned using directives starting with a point (.) and a keyword just like with Spice syntax. Any line can be continued on the next one using character plus (+).

A *carac* is defined by the values of all directives located between two `.CARAC` or `.CHANGE` directives (see the description of these directives for more informations: .CARAC directive and .CHANGE directive), or between the beginning of the file and the first `.CARAC` or `.CHANGE` directive (in this case, your *carac* name will be set to the default value "*myCarac*", but it does not exist if no *simulation* has been stipulated), or between the last `.CARAC` or `.CHANGE` directive and the end of the file.

When having various *caracs* defined in the same file, values of directives set in previous ones remain set to the same value unless they are overloaded. You can overload the value with an empty string by setting the directive alone with no value if you want reset it.

```
# This is a comment

.CARAC TheFirstCarac

.PARAM TYP

.SIMU    op
+        openLoop    # This is a comment
+        transient

.CARAC TheSecondCarac

.PARAM SWEEP

# Value of .SIMU directive remains the same.
```

In this *configuration* example, two *caracs* are executed: in the first one, library `TYP` of the *libparam* file will be loaded; in the second one, library `SWEEP` will be loaded; in both of them, three *simulations* are going to be executed by including respectively files `op.inc`, `openLoop.inc` and `transient.inc` (where `.inc` is the selected extension for *simulation* files).

6.3.2 .CARAC directive

Description:

This directive is used to define the name of a *carac* and to delimit its description. Unless it is the first one encountered in the *configuration* file when no *simulation* has been stipulated, the previously defined *carac* is set when this directive appears. When using this directive, the user chooses the name of his *carac* using the parameter `<name>` and then can change all its settings using any available directive. The *carac* definition ends when an other `.CARAC` directive is encountered or when End Of File is reached. So, a *carac* is usually defined between two `.CARAC` directives and the name of the *carac* is the one set in the first of these two.

If a setting has been done through an other directive in a previous *carac* and the same directive is not re-used in the current *carac* definition, the previous value is also used for this *carac*. Indeed this directive does not reset the previous settings and allows only to define the parameters the user wants to alter: any directive which is re-used permits to overload the value of the previous one.

If the user wants to reset the value of a *carac* setting, he may use the same directive used as before with no value at all.

An alias to this directive is also available: `.CHANGE` directive, see subsection .CHANGE directive.

Syntax:

```
.CARAC <name>
```

Parameters:

<name>	String: defines the name of the *carac* which is configured after the .CARAC directive.

6.3.3 .CHANGE directive

This directive is an alias to .CARAC directive. See .CARAC description in subsection .CARAC directive for more informations.

6.3.4 .CHECKMEAS directive

Description:

This directive is used to verify that the measures in the *simulations* are in accordance to some specifications: it permits to define the name of the measure to check and the minimum and maximum tolerated values. So, when extracting the results, openCarac will check the conformity of the results and prints them with a format in accordance. For instance, results will be colorized in green or red into HTML tables or check matrices will be available in GNU Octave ^{TM} scripts.

To use this directive, the user must add a triplet of parameters to define respectively the name of the measure, its minimum and maximum tolerated values. More than one triplet can be defined if there are more than one measure to check.

Syntax:

```
.CHECKMEAS <name> <min> <max> [<name> <min> <max>]*
```

Parameters:

<name>	String: defines the name of the measure the user wants to check the value of.
<min>	Real: minimum tolerated value for the measure to check.
<max>	Real: maximum tolerated value for the measure to check.

Example:

In the following example is shown the definition of a *carac* where both *simulation* files op.inc and openLoop.↩ inc are used and two measures named phi_margin and gain_dc are defined in the files.
It permits to verify that the measured value of phi_margin is higher than 1.0472 and lower than 3.1416 and that measured value of gain_dc is higher than 60 and lower than 100.

```
.CARAC myCarac

.SIMU    op
+        openLoop    # AC analysis where measures to check are defined.

.CHECKMEAS  phi_margin 1.0472 3.1416   # 60/180*pi = 1.0472
+           gain_dc    60     100
```

6.3.5 .CHECKOP

directive

Description:

Since openCarac is used to sum up the values of measures and measuring a voltage at operating point is not available with every simulator, directive `.CHECKOP` can be defined. When set, this directive makes openCarac parse the simulator results files looking for voltage values at operating point. Just like when using directive `.CHECKMEAS` (see subsection .CHECKMEAS directive), it permits to define the name of the voltage to check the value of and the minimum and maximum tolerated values. Conformity of the results is checked and are printed with a format in accordance.

To use this directive, the user must add a triplet of parameters to define respectively the name of the net to get the voltage value of, its minimum and maximum tolerated values. More than one triplet can be defined if there are more than one voltage value to check.

Syntax:

```
.CHECKOP <net> <min> <max> [<net> <min> <max>]*
```

Parameters:

`<net>`	String: defines the name of the net the user wants to check the voltage value of.
`<min>`	Real: minimum tolerated value for the voltage value of the net to check.
`<max>`	Real: maximum tolerated value for the voltage value of the net to check.

Example:

In the following example is shown the definition of a *carac* where a *simulation* file `op.inc` is used and two nets named `IN1` and `IN2` are defined in the *netlist* file.
It permits to extract the voltage values of nets `IN1` and `IN2` at operating point and verify that the value of the voltage on net `IN1` is higher than 2.25 and lower than 2.75 and that measured value of the voltage on net `IN2` is higher than 0 and lower than 5.

```
.CARAC myCarac

.SIMU    op

.CHECKOP  IN1 2.25 2.75
+         IN2 0    5
```

6.3.6 .CORNER directive

Description:

When running a *carac* where library loadings from the *model* file are to be altered, a list of library names to load has to be specified (see section Organize your Spice circuit files), these library names are called *corners*. In order to define the list of *corners* to load, directive `.CORNER` has to be used. A list of strings, libraries names, separated by spaces is to be added to define the *corners* to load.

Syntax:

```
.CORNER <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the library to load.

Example:

In the following example is shown the definition of a *carac* where *simulation* file `op.inc` is used and is simulated in three *corners*.

```
.CARAC myCarac

.SIMU    op
.CORNER  TYP FAST SLOW
```

6.3.7 .CUSTOMARGS directive

Description:

The value given to this directive remains unused by openCarac in most situations but it can be accessed by the user in a hook or a custom procedure to change a behaviour function of a *carac*. See chapter Expert mode for more informations.

All the characters printed after this directive form a string to set the custom args to.

Syntax:

```
.CUSTOMARGS <args>
```

Parameters:

`<args>` String: value of the custom args.

6.3.8 .EXTRACTOP directive

Description:

Since openCarac is used to sum up the values of measures and extracting a device parameter at operating point is not available with every simulator, directive `.EXTRACTOP` can be defined. When set, this directive makes openCarac parse the simulator results files looking for devices parameters values at operating point. It permits to define the name of the *simulation* the operating point must be get from and the name of the parameters to extract.

If nothing else is done, the given parameter is going to be extracted from every devices available in the simulator output files and data is printed in the results file just like it was a measure extraction. In order to filter a list of devices to extract the parameters from, the user can add a `devices.list` file in the same directory as the *configuration* file where a list of devices is defined; if this file is present, parameters will be extracted only from the devices having the same names as given in this file.

Syntax:

```
.EXTRACTOP <simulation> <parameter> [<simulation> <parameter>]*
```

Parameters:

<simulation>	String: name of the *simulation* to extract the parameter from.
<parameter>	String: name of the parameter to extract from the devices.

Note: The *simulation* name must also be present in the list given to the .SIMU directive as defined in subsection .SIMU directive .

Example:

In the following example is shown the definition of a *carac* where three *simulations* files op.inc, openLoop.inc and tran.inc are used, parameter VDS will be extracted from devices in results files when simulating with file op.inc and parameter ID will be extracted from devices in results files when simulating with file openLoop.inc.

```
.CARAC myCarac

.SIMU      op       openLoop tran

.EXTRACTOP op       VDS
+          openLoop ID
```

If a filter must be applied on two devices, the following devices.list file can be added in the same directory:

```
M.XAMP.MDP2
M.XAMP.MDP1
```

6.3.9 .LIBPARAMLIN directive

Description:

Since openCarac allows to alter the loading of a library for simulation conditions, the user must load a file containing all the possible libraries. If various library files are available to define different conditions for a circuit, directive .LIBPARAMLIN permits to define a Linux ^{TM} format path to access an other file containing parameters libraries. In this case, it becomes possible to change the file containing the parameters libraries to load.

This directive can be completed by the usage of .LIBPARAMWIN which allows to define a Windows ^{TM} format path to access the same file (see subsection .LIBPARAMWIN directive); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux ^{TM} and Windows ^{TM}.

An alias to this directive is also available: .LINLIBPARAM directive, see subsection .LINLIBPARAM directive .

Syntax:

```
.LIBPARAMLIN <path>
```

Parameters:

<path> String: Linux ^{TM} format path to access the file containing the parameters libraries to load.

Example:

In the following example is shown the definition of a *carac* where the file containing the parameters libraries to load is altered. This works under both Linux ^{TM} and Windows ^{TM}.

```
.CARAC myCarac

.LIBPARAMLIN  /media/device/folder/conditions.lib
.LIBPARAMWIN  E:\folder\conditions.lib
```

6.3.10 .LIBPARAMWIN directive

Description:

Since openCarac allows to alter the loading of a library for simulation conditions, the user must load a file containing all the possible libraries. If various library files are available to define different conditions for a circuit, directive `.LIBPARAMWIN` permits to define a Windows ^{TM} format path to access an other file containing parameters libraries. In this case, it becomes possible to change the file containing the parameters libraries to load.

This directive can be completed by the usage of `.LIBPARAMLIN` which allows to define a Linux ^{TM} format path to access the same file (see subsection .LIBPARAMLIN directive); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux ^{TM} and Windows ^{TM}.

An alias to this directive is also available: `.WINLIBPARAM` directive, see subsection .WINLIBPARAM directive.

Syntax:

```
.LIBPARAMWIN <path>
```

Parameters:

<path> String: Windows ^{TM} format path to access the file containing the parameters libraries to load.

Example:

In the following example is shown the definition of a *carac* where the file containing the parameters libraries to load is altered. This works under both Linux ^{TM} and Windows ^{TM}.

```
.CARAC myCarac

.LIBPARAMWIN  E:\folder\conditions.lib
.LIBPARAMLIN  /media/device/folder/conditions.lib
```

6.3.11 .LINLIBPARAM directive

This directive is an alias to `.LIBPARAMLIN` directive. See `.LIBPARAMLIN` description in subsection .LIBPARAMLIN directive for more informations.

6.3.12 .LINMODEL directive

This directive is an alias to `.MODELLIN` directive. See `.MODELLIN` description in subsection .MODELLIN directive for more informations.

6.3.13 .MODELLIN directive

Description:

Since openCarac allows to alter the loading of a library for *corners* definitions, the user must load a file containing all the possible *corner* libraries. If various library files are available to define different *models* for the devices, directive `.MODELLIN` permits to define a Linux ^{TM} format path to access an other file containing *corners* libraries. In this case, it becomes possible to change the file containing the *corner* libraries to load.

This directive can be completed by the usage of `.MODELWIN` which allows to define a Windows ^{TM} format path to access the same file (see subsection .MODELWIN directive); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux ^{TM} and Windows ^{TM}.

An alias to this directive is also available: `.LINMODEL` directive, see subsection .LINMODEL directive.

Syntax:

```
.MODELLIN <path>
```

Parameters:

`<path>` String: Linux ^{TM} format path to access the file containing the *models* libraries to load.

Example:

In the following example is shown the definition of a *carac* where the file containing the *models* libraries to load is altered. This works under both Linux ^{TM} and Windows ^{TM}.

```
.CARAC myCarac

.MODELLIN  /media/device/folder/models.lib
.MODELWIN  E:\folder\models.lib
```

6.3.14 .MODELWIN directive

Description:

Since openCarac allows to alter the loading of a library for *corners* definitions, the user must load a file containing all the possible *corner* libraries. If various library files are available to define different *models* for the devices, directive `.MODELWIN` permits to define a Windows ^{TM} format path to access an other file containing *corners* libraries. In this case, it becomes possible to change the file containing the *corner* libraries to load.

This directive can be completed by the usage of `.MODELLIN` which allows to define a Linux ^{TM} format path to access the same file (see subsection .MODELLIN directive); in this case, the user can define both paths to access the same file, each of them compatible with respectively Linux ^{TM} and Windows ^{TM}.

An alias to this directive is also available: `.WINMODEL` directive, see subsection .WINMODEL directive.

Syntax:

```
.MODELWIN <path>
```

Parameters:

`<path>` String: Windows ^{TM} format path to access the file containing the *models* libraries to load.

Example:

In the following example is shown the definition of a *carac* where the file containing the *models* libraries to load is altered. This works under both Linux ^{TM} and Windows ^{TM}.

```
.CARAC myCarac

.MODELWIN  E:\folder\models.lib
.MODELLIN  /media/device/folder/models.lib
```

6.3.15 .NETLIST directive

Description:

Sometimes, it can be useful to run the same *simulations* with different *netlists* and duplicate all the files cannot be the best way to do so. A solution to re-use the *simulations* files is offered by openCarac: if the user gets the part of the circuits he wants to alter in a separated file with the *netlist* extension, it can be altered when executing the *carac* as soon as an inclusion of a *netlist* file is available in the main control file.

To do so, `.NETLIST` directive permits to define a list of *netlist* file names which will be loaded during the *carac*. Note that only one occurrence of *netlist* file inclusion must be available in the main file when executing the *carac*. A *netlist* file is identified by openCarac through its extension.

Syntax:

```
.NETLIST <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the *netlist* file to include.

Example:

In the following example is shown the definition of a *carac* where both file `nampli.nsx` and `pampli.nsx` will be alternatively included in the main control file, where `.nsx` is the *netlist* file extension.

```
.CARAC myCarac

.NETLIST nampli pampli
```

6.3.16 .PARAM directive

Description:

When running a *carac* where library loadings from the *libparam* file are to be altered, a list of library names to load has to be specified (see section Organize your Spice circuit files), these library names are called *params*. In order to define the list of *params* to load, directive `.PARAM` has to be used. A list of strings, libraries names, separated by spaces is to be added to define the *params* to load.

Syntax:

```
.PARAM <name> [<name>]*
```

Parameters:

`<name>` String: defines the name of the library to load.

Example:

In the following example is shown the definition of a *carac* where *simulation* file `op.inc` is used and is simulated with two *params* libraries.

```
.CARAC myCarac

.SIMU    op
.PARAM   TYP SWEEP
```

6.3.17 .PARAMETERS directive

This directive is an alias to `.PARAM` directive. See `.PARAM` description in subsection .PARAM directive for more informations.

6.3.18 .RESET directive

Description:

When configuring various *caracs* using `.CARAC` directive as described in subsection .CARAC directive, the user only needs to re-define what he wants to alter: the settings of every directive remain identical if not set again. If a directive must be set to its default value, it is possible to set this directive with no parameter: having an empty string is interpreted as the directive has never been set.

Sometimes, the *carac* settings may be so different as the previous one that every directives have to be reset; in such a case, it becomes possible to use `.RESET` directive. This makes every directive except `.CARAC` placed before reset to its default value.

Syntax:

```
.RESET
```

Parameters:

This directive is used with no parameter.

Example:

In the following example is shown the *configuration* of four *caracs* where:

- In the first *carac*, a *simulation* file `op.inc` is included and measure "myMeas" is checked.
- In the second *carac*, a *simulation* file `transient.inc` is included and measure "myMeas" is also checked.
- In the third *carac*, `.CHECKMEAS` directive is reset and *simulation* file `openLoop.inc` is included with no measure checked.
- In the fourth *carac*, nothing is done because `.RESET` directive also resets the `.SIMU` directive placed before.

```
.CARAC firstCarac
.SIMU      op
.CHECKMEAS myMeas 0 1

.CARAC secondCarac
.SIMU      transient

.CARAC thirdCarac
.RESET
.SIMU      openLoop

.CARAC fourthCarac
.SIMU      op
.RESET
```

6.3.19 .SIMU directive

Description:

When running a *carac* where various *simulations* have to be run, the user must separate the *simulations* definitions in various files with the appropriate extension, default value for this extension is `.inc` (see section Organize your Spice circuit files). So, to define the list of *simulations* to run, directive `.SIMU` has to be used and a list of strings, names of files with the appropriate extension, separated by spaces has to be added to define the files where the *simulations* to include are configured.

Also, since it can be useful to modify the values of parameters, the user can make each *simulation* name being followed by couples of parameters name and value defined between parentheses. What is defined in a field delimited by parentheses will determine the parameters to alter when *running* the *simulation* once, various couples can be defined to make various parameters vary. If the user wants to run various times the same *simulation* when altering the parameters with other values or other parameters, others fields delimited by parameters can be defined.

Each time, the first string is considered as the name of the *simulation* and what is defined between the parentheses is considered as a parameters setting. If various parameters settings are described after a *simulation* name, the same *simulation* name is considered and the *simulation* will be run various times.

Syntax:

```
.SIMU <name> [(<paramName> <paramValue> [<paramName> <paramValue>]*)]*
+    [<name> [(<paramName> <paramValue> [<paramName> <paramValue>]*)]*]*
```

Parameters:

<name>	String: defines the name of the file with the *simulation* extension to include.
<paramName>	String: defines the name of the parameter to alter.
<paramValue>	String: defines the value to give to the parameter.

Example:

In the following example is shown the definition of a *carac* where four *simulations* `op`, `openLoop`, `transient` and `noise` are used by including respectively files `op.inc`, `openLoop.inc`, `transient.inc` and `noise.inc`, where `.inc` is the *simulation* files extension.

Simulation `openLoop` will run twice, the first time when altering both parameters "millerValue" and "loadValue" to respectively $20f$ and $10k$; and the second time when altering parameter "millerValue" to $35f$ and by keeping originally defined value of parameter "loadValue". Other *simulations* will run once with no parameter alteration.

```
.CARAC myCarac

.SIMU op
+     openLoop  (millerValue 20f loadValue 10k) (millerValue 35f)
+     transient noise
```

6.3.20 .SIMULATIONS directive

This directive is an alias to `.SIMU` directive. See `.SIMU` description in subsection .SIMU directive for more informations.

6.3.21 .SIMULATOR directive

Description:

In case a *carac* can be run only with a specific simulator, directive `.SIMULATOR` can be used to force the selection of the simulator to run the *carac* with. Its effect is to overload the simulator selection that may have been done through openCarac environment settings or execution options. See subsection Simulators sections for the list of the compatible simulators.

If the specified simulator is not supported by openCarac, an error message is printed and the directive remains ignored.

Syntax:

```
.SIMULATOR <name>
```

Parameters:

<name>	String: defines the name of the simulator to select.

Example:

In the following example is shown the definition of a *carac* where simulator Ngspice is selected and *simulation* `op` will run in three *corners*.

```
.CARAC myCarac

.SIMULATOR NGSPICE
.SIMU      op
.CORNER    TYP FAST SLOW
```

6.3.22 .WINLIBPARAM directive

This directive is an alias to `.LIBPARAMWIN` directive. See `.LIBPARAMWIN` description in subsection .LIBPARAMWIN directive for more informations.

6.3.23 .WINMODEL directive

This directive is an alias to `.MODELWIN` directive. See `.MODELWIN` description in subsection .MODELWIN directive for more informations.

Chapter 7

Expert mode

Abstract: In this chapter are described the procedures to follow in order to extend openCarac capabilities for users that are familiar with TCL/Tk and any other language. Section Use of a custom simulator lists what the user should be aware of when using a custom command instead of the the simulator, section Custom procedure gives guidelines for editing the custom procedure and section Hooks details how to benefits the use of the hooks.

7.1 Use of a custom simulator

As introduced in chapter Simulator specificities, openCarac comes with the possibility to use a custom command instead of a native Spice simulator. Such a command can be any program from a non-native Spice simulator to a shell interpreter or a digital simulator. openCarac behaviour remains identical but, to ensure that its execution through openCarac does not bring any surprise, the following guidelines must be followed:

First, make sure that the presence of markers in the main file does not alter the command execution by making them starting with a comment syntax (see subsections modelMarker, paramMarker and simuMarker for more informations).

Then, be aware that openCarac does not include any files parser when using a custom command. As a consequence, in order to proceed to a results extraction, the user has to create his own parsers. An example to do so is given in an openCarac API example.

7.2 Custom procedure

openCarac *application* has a boolean property that defines if "custom execution mode" is activated (as described in subsection Section application), when such a mode is activated, the execution of the simulator differs from what is described in section Execution of the simulator. Indeed, execution of the simulator is done through the `openCarac_customRunSimulator` procedure that is defined in the `customProcedures.tcl` file in the openCarac environment files folder located in the user home directory.

Such a file is created when executing openCarac main executable with the `--configure` option as shown in subsection Define your simulation environment. The default file is also available in the same directory as openCarac main executable: when executing openCarac main executable, the default file is loaded and, when the environment is loaded, a source of the user `customProcedures.tcl` file occurs and any procedure defined in it overloads the previous one.

The execution of the simulator through openCarac in "custom mode" is performed as defined in the `openCarac_customRunSimulator` procedure. A documentation of the default behaviour of this procedure is given in openCarac API reference manual.

It is then possible for the user to modify the definition of this `openCarac_customRunSimulator` procedure to alter openCarac behaviour. To do so, in addition to TCL functions, openCarac provides API functions that are described in the API reference manual. These API functions can be used as soon as openCarac is running, including in the custom procedure and the hooks.

API functions to access the caracs "custom args" attribute are provided (see the documentation of function `openCarac_caracGetCustomArgs` in openCarac API reference manual for more informations) so that the custom procedure can provide a different behaviour function of the *carac*.

7.3 Hooks

In addition to the custom procedure, in the `customProcedures.tcl` file, openCarac comes with hooks: these procedures remain empty by default but can be modified in order to execute some code at a given instant in openCarac execution. The hooks are executed before or after executing the main code of an openCarac API function, see the openCarac API reference manual to know the instant when the hooks are executed.

Since openCarac main executable uses API function, what is executed in the hooks is also executed in the main executable: modifying the hooks permit to alter openCarac main executable behaviour as well.

openCarac comes with API examples where hooks are modified to fit a particular purpose, please refer to their respective documentations for a detailed description.