SCAPS Version 2.9.03, August 2010

This is not a stand-alone manual of SCAPS. It only describes the novelties in SCAPS 2.9.03 compared to SCAPS 2.9.02 and earlier. A (kind of) manual of the SCAPS programme is contained in another document. This is complemented with (so far) two add-on manuals, for SCAPS 2.8 ('grading') and for SCAPS 2.9 ('multilevel defects'). Also, there is a short and recommendable document `Getting Started.pdf`, which does exactly what it promises.

1 Enhancements of version 2.9.03, compared to version 2.9.02

The most important new features in 2010 are in the user interface: more information can now be saved and loaded, and a script language is implemented.

1.1 Saving and loading

Until now, you could save and load the problem definition file. Now, you can also save and load the action list (all settings on the action panel), the batch description (all settings on the batch panel) and the recorder settings (all settings in the recorder panel). Also, you can save and load all settings together in one file. Doing so, you can reproduce a former simulation by just clicking `calculate`: all other settings will be as they were. In the table below the default file extensions and directories are given (the SCAPS mother directory, this is where the scaps.exe file resides, is noted as scaps\). All these files are normal text files that can be read and edited with e.g. notepad. However, editing is at own risk, it is advised to change these files from within SCAPS.

SCAPS filetypes	default extension	default directory
problem definition	.def	scaps\def
action list	.act	scaps\def
batch settings	.sbf	scaps\bdf
recorder settings	.srf	scaps\bdf
all SCAPS settings in one	.scaps	scaps\def
a SCAPS script	.script	scaps\script

1.2 Arguments on the SCAPS command line

Normally SCAPS is started by clicking an icon on the desktop. Internally in Windows, a command line attached to this icon is executed. This command line just contains the full path of the SCAPS .exe file (thus scaps2902.exe or scaps2903.exe or so). Now you can add extra arguments to this command line: a list of filenames to be loaded/executed before SCAPS starts: these can be one or more of the definition, action,…, script files listed above; also a spectrum file and a generation file can be given. The order of execution is: first .def, then .act, .scaps, .spe, .gen, .sbf, .srf, and finally .script. By doing so, you can ensure that SCAPS starts in the condition that you prefer, not in the condition fixed by the SCAPS developers: good news for

those who had to set e.g. the wavelength range appropriate for CIGS each time again and again.

Here are two ways to start SCAPS from a command line, and to edit this line:

1. Make, e.g. with Notepad, a batch file with extension .bat (this stems from the very old MS-DOS times, but is still supported in Windows). Write the command line in one line, e.g.

   ```
   scaps2903.exe all CIGS.scaps AM0.spe
   ```

   A .bat file is run by double clicking it; you can also make a shortcut to it on the desktop. To edit the .bat file, right-click the name and select 'edit', or directly open from Notepad.
2. Right-click the normal SCAPS-icon on your desktop, select 'properties', and edit the third line ('target'). This is well suited when you do this once and for all. When you would alter the command line more frequently, the previous method might feel more comfortable.

1.3 A list of parameters or filenames as a batch-parameter

In SCAPS2.9.02, file names could be passed as batch parameters: definition files (always the first batch parameter), spectrum files and generation files. These filenames could be selected in a list, but not saved and loaded. Also, an arbitrary parameter list could be used for all numerical parameters. These were read from a file with extension bdf (batch definition file). Now, in SCAPS2.9.03, all file parameters are available as batch parameters: thus also absorption files, filter files, and optical capture files. These lists of filenames can be edited in a window, and also saved and loaded. In output files, the filenames appear, not just the number of the file name, as it was before. An arbitrary list of parameters does now not need to be read from a file, it can be directly edited in a window (where you can paste e.g. from Excel); the list can be saved and loaded to a file as before.

1.4 A script language for SCAPS

There is now (on urgent users demand) a facility to write a script and let it run in SCAPS. This script is a normal text file, that contains commands that are equivalent to a mouse-click. During execution of the script, SCAPS will not respond to user interventions (mouse or keyboard). After execution of the script, SCAPS returns to it's normal interactive mode, or is switched off (to be set with a script command). When no script file was found in the command line, SCAPS remains the same interactive program as it has always been. A few SCAPS script are distributed with SCAPS2.9.03 to serve as an example.

There are several levels of sophistication to use scripts.

The basic use is that you write down all actions you would do in the interactive mode, and then can leave the lab. One advantage of doing so would be that you could split up your to-do list in several smaller batch jobs instead of one gigantic batch job (by giving several `load batchsettings` commands), and that you save the results in between (by giving `save results` or `save graphs` commands). This is safer when there is a risk that your computer (or SCAPS ☹) would hang up underway; also, all results are waiting for you when you come back to the lab.

A more sophisticated use is that you start and run an own (or another) program somewhere during the execution of the script, by giving a `runsystem` command. For example, you

could load a problem, do just the equilibrium calculation, and save the results of the generation panel to a file; this also contains the mesh. You could then open an own program, read this mesh-file and use it to do an own calculation of the optical generation, and save this in a file with the format of a SCAPS generation file. When your function returns, and the script is continued. You could then load this generation file, and do all calculations you want. This procedure is more convenient then the full-manual method that several SCAPS users have intensely used.

In this way, the communication between SCAPS and your own program is only via the file system: both SCAPS and your program read and write files, but do not communicate directly with each other. Also, reading and especially writing files in a SCAPS format might be cumbersome for a programmer. For advanced users, a more direct communication method between SCAPS and an own program has been implemented. The user program should have been compiled as a *dynamically linked library*, and should be placed in the SCAPS mother directory as a .dll and a .lib file. Also, these filenames, the name of the dll-program(up to now there is only one), and the definition (the argument list) of this function is fixed. This functionality seems to address well skilled programmers only. However, one such dll, with a flexible functionality, is distributed with SCAPS, and using it does not require high level programmer skills: see the description of the `rundll` command below, and the examples distributed with SCAPS. In contrast, we are confident that the basic use of SCAPS scripts can facilitate the simulation work of a broad class of SCAPS users.

A detailed description of the SCAPS script language is given below.

Marc Burgelman 27-8-2010

SCAPS script commands

general

The SCAPS-directory, this is where the scaps.exe file resides, is noted as `scaps\`. All command lines consist of up to three parts:

```
command argument value
```

where `command` and `argument` are reserved words, and `value` is free with some restrictions, depending on the command line. The three components of the command line are separated by whitespace (spaces, tabs,..), but should be on one line. They are not case-sensitive (upper case or lower case letters do not matter).

At this moment, the possible commands are:

SCAPS script comands			
`load`	`set`	`calculate`	`rundll`
`save`	`get`	`loop`	`runsystem`
`action`	`clear`	`show`	

Whilst processing a script, SCAPS internally maintains a few variables, as specified in the table below. The user can use these variables in `set` and `get` commands, and some are used internally in a `loop`. Also, these variables are passed to an external dll function, that can be made by the user.

name	C-type	default value	max value
xvalue	double	0	
yvalue	double	0	
xvector	array of double	0	
yvector	array of double	0	
nx	int	0	
ny	int	0	
loopcounter	int	0	
maxiteration	int	25	
looperror	double	1E30	
maxerror	double	1E-3	
status	int	0	
mode	int	0	
filename	character string	empty	max size 256 bytes

load – commands

Syntax:

`load argument value`

Where `load` is the reserved command word, `argument` can take 7 reserved values, and `value` is a filename, without path. The filename can contain spaces. The files are supposed to reside in their default directories. There is (exceptionally) some freedom allowed in the name of the argument: just writing `definition`, `action`, `batch`, `record`, `allscaps`, `spectrum` or `generation` will also do.

command	argument	value	default-directory
load	definitionfile	a filename	scaps\def
load	actionlistfile	a filename	scaps\def
load	batchsettingsfile	a filename	scaps\bdf
load	recordersettingsfile	a filename	scaps\bdf
load	allscapssettingsfile	a filename	scaps\def
load	spectrumfile	a filename	scaps\spectrum
load	generationfile	a filename	scaps\generation

save – commands

Syntax:

```
save argument value
```

Where `save` is the reserved command word, `argument` has a compound syntax; the first part can take 3 reserved values (`settings`, `results` or `graphs`). The `value` is a filename, without path. The filename can contain spaces. The files are supposed to reside in their default directories.

command	argument	value	default-directory
save	settings.definitionfile	a filename	scaps\def
save	settings.actionlistfile	a filename	scaps\def
save	settings.batchsettingsfile	a filename	scaps\bdf
save	settings.recordersettingsfile	a filename	scaps\bdf
save	settings.allscapssettingsfile	a filename	scaps\def
save	results.eb	a filename	scaps\results
save	results.genrec	a filename	scaps\results
save	results.ac	a filename	scaps\results
save	results.iv	a filename	scaps\results
save	results.cv	a filename	scaps\results
save	results.cf	a filename	scaps\results
save	results.qe	a filename	scaps\results
save	results.recorder	a filename	scaps\results
save	graph.eb.wholepanel	always .png !	scaps\results
save	graph.eb.energybands	always .png !	scaps\results
save	graph.eb.carrierdensities	…	…
save	graph.eb.currents		
save	graph.eb.ftraps		
save	graph.ac.wholepanel		
save	graph.ac.currents.amplitude		
save	graph.ac.currents.phase		
save	graph.ac.potentials.amplitude		
save	graph.ac.potentials.phase		
save	graph.genrec.wholepanel		
save	graph.genrec.genrec		
save	graph.genrec.ftraps		
save	graph.iv.wholepanel		
save	graph.iv.iv		

save	graph.iv.recombination
save	graph.cv.wholepanel
save	graph.cv.cv
save	graph.cv.gv
save	graph.cv.Mott-Schottky
save	graph.cv.dopingprofile
save	graph.cf.wholepanel
save	graph.cf.cf
save	graph.cf.gf
save	graph.cf.Nyquist
save	graph.cf.G(f)/f-f
save	graph.qe.wholepanel
save	graph.qe.qe
save	graph.recording.wholepanel
save	graph.recording.resultsgraph
save	graph.grading.wholepanel
save	graph.grading.gradinggraph

action – commands

Syntax:

`action argument value`

Where `action` is the reserved command word, `argument` can take 39 reserved values, and `value` is a numerical value or a filename, without path. The filename can contain spaces. The files are supposed to reside in their default directories. Some values can take two values only (0 or 1). There is a (very) limited degree of freedom in the exact arguments. E.g. instead of `iv.checkaction`, you can also write `iv.doiv` or `iv.iv`. Instead of `batch.checkaction`, you can also write `batch.dobatch` (as in the user interface); and alike with `recording.dorecord`. When the value of these commands is omitted, the value 1 is assumed (giving a clear meaning to the form `doiv`, `docv`,…, `dobatch`…).

command	argument	value	remark
action	workingpoint.temperature		Kelvin
action	workingpoint.voltage		Volt
action	workingpoint.frequency		Hz
action	workingpoint.numberofpoints	≥ 2	
action	dark	none	overrides light
action	light	none	overrides dark

action	generationfrominternalmodel	none	overrides generationfromfile
action	spectrumfile	filename	scaps\spectrum
action	spectrumcutoff.on	none	overrides spectrumcutoff.off
action	spectrumcutoff.off	none	overrides spectrumcutof
action	spectrumcutoff.shortlambda	filename	nm
action	spectrumcutoff.longlambda		nm
action	intensity.ND		
action	intensity.T		%
action	generationfromfile	none	overrides generationfrominternalmo
action	generationfile	filename	scaps\generation
action	generationfromfile.attenuation		%
action	iv.startV		V
action	iv.stopV		V
action	iv.points	≥ 2	
action	iv.increment		V
action	iv.checkaction	0 or 1	
action	iv.stopafterVoc	0 or 1	
action	cv.startV		
action	cv.stopV		V
action	cv.points	≥ 2	V
action	cv.increment		V
action	cv.checkaction	0 or 1	
action	cf.startf		Hz
action	cf.stopf		Hz
action	cf.total points	≥ 2	
action	cf.points per decade	≥ 2	
action	cf.checkaction	0 or 1	
action	qe.startlambda		nm
action	qe.stoplambda		nm
action	qe.points	≥ 2	
action	qe.increment		nm
action	qe.checkaction	0 or 1	
action	batch.checkaction	0 or 1	
action	recording.checkaction	0 or 1	

set – commands

Syntax:

`set argument value`

where `set` is the reserved command word, `argument` can take the reserved values from the table below. The `set` command can also be used to set the script variables. The third part of the `set` command line is `value`: this is a numerical value or a filename, without path. The filename can contain spaces. The files are supposed to reside in their default directories. Some values can take two values only (0 or 1). When the value is a numerical value, you can specify a number, e.g. 1.25E16, or one of the internal script variables xvalue, yvalue, xvector[*ix*], yvector[*ix*] (take care that *ix* < nx and *iy* < ny). How the values of the internal variables xvalue, yvalue, xvector, yvector can be set directly with a `set`-command; also, they are used and possibly changed in SCAPSUserFunction.dll (see later). The value of nx, ny is updated in a `get` command, see later. You can also use xvector[-1] or yvector[-1]: then nx or ny are incremented with one, and the value is placed as the new last element. You can also use xvector or yvector (thus without index), then nx or ny are incremented, all existing elements are shifted one up, and the value is placed at xvector[0] or yvector[0]. Before setting a script variable, you might want to re-initialise them with one of the `clear` commands, see later.

command	argument	value	remark
set the script variables			
set	scriptvariable.maxiteration	integer	
set	scriptvariable.status	integer	
set	scriptvariable.mode	integer	
set	scriptvariable.looperror		
set	scriptvariable.maxerror		
set	scriptvariable.xvalue		
set	scriptvariable.yvalue		
set	scriptvariable.xvector[*i*]		0 ≤ *i* ≤ nx - 1, or *i* = -1, or no index
set	scriptvariable.nx	integer	
set	scriptvariable.yvector[*i*]		0 ≤ *i* ≤ ny - 1, or *i* = -1, or no index
set	scriptvariable.ny	integer	
set	scriptvariable.filename	characterstring	length < 256
general set commands			
set	quitscript.interactiveSCAPS	no value	the default

set	quitscript.quitSCAPS	no value	
set	errorhandling.toscreen	no value	
set	errorhandling.appendtofile	no value	the default
set	errorhandling.overwritefile	no value	
set	errorhandling.outputlist.truncate	no value	the default
set	errorhandling.outputlist.fillzeros	no value	
set	errorhandling.outputlist.fillwhite	no value	
set	external.Rs		Ωcm^2
set	external.Rsh		Ωcm^2
set	external.Gsh		Scm^{-2}
set	internal.reflection		fraction, not %
set	internal.transmission		fraction, not %
illumination set commands			
set	illumination.fromleft	no value	
set	illumination.fromright	no value	
set	qe.photonflux		$\#.cm^{-2}s^{-1}$
set	qe.photonpower		Wcm^{-2}
contact set commands: replace contact with either leftcontact or rightcontact			
set	contact.Sn		$cm.s^{-1}$
set	contact.Sp		$cm.s^{-1}$
set	contact.opticalfilter.on	no value	
set	contact.opticalfilter.off	no value	
set	contact.opticalfilter.transmission	no value	
set	contact.opticalfilter.reflection	no value	
set	contact.opticalfilter.value		fraction, not %
set	contact.opticalfilter.file	a filename	scaps/filter
set	contact.workfunction		V or eV
set	contact.flatband.off	no value	
set	contact.flatband.once	no value	
set	contact.flatband.always	no value	
layer set commands: replace layer with layer1, layer2, … layer7			
set	layer.thickness		μm
set	layer.Eg		eV
set	layer.chi		V or eV
set	layer.epsilon		-

set	layer.NC		cm^{-3}
set	layer.NV		cm^{-3}
set	layer.vthn		$cm.s^{-1}$
set	layer.vthp		$cm.s^{-1}$
set	layer.mun		$cm^{2}V^{-1}s^{-1}$
set	layer.mup		$cm^{2}V^{-1}s^{-1}$
set	layer.NA		cm^{-3}
set	layer.ND		cm^{-3}
set	layer.radiative		$cm^{3}s^{-1}$
set	layer.Augern		$cm^{6}s^{-1}$
set	layer.Augerp		$cm^{6}s^{-1}$
set	layer.absorption.file	a filename	scaps\absorption
set	layer.absorption.A		$eV^{-1/2}cm^{-1}$
set	layer.absorption.B		$eV^{+1/2}cm^{-1}$
defect set commands: replace layer with layer1, …, and defect with defect1, defect2 or defect3			
set	layer.defect.singlelevel	no value	
set	layer.defect.uniform	no value	
set	layer.defect.gauss	no value	
set	layer.defect.CBtail	no value	
set	layer.defect.VBtail	no value	
set	layer.defect.neutral	no value	
set	layer.defect.singledonor	no value	
set	layer.defect.singleacceptor	no value	
set	layer.defect.aboveEV	no value	
set	layer.defect.belowEC	no value	
set	layer.defect.aboveEi	no value	
set	layer.defect.Et		eV
set	layer.defect.Echar		eV
set	layer.defect.Ntotal		cm^{-3}
set	layer.defect.Npeak		$cm^{-3}eV^{-1}$
interface set commands: replace interface with interface1, interface2, … interface6			
set	interface.IBtunneling.off	no value	
set	interface.IBtunneling.on	no value	
set	interface.IBtunneling.me		--

set	interface.IBtunneling.mh		--

interface defect set commands: replace interface with interface1,… and IFdefect with IFdefect1, IFdefect2, IFdefect3			
set	interface.IFdefect.singlelevel	no value	
set	interface.IFdefect.uniform	no value	
set	interface.IFdefect.gauss	no value	
set	interface.IFdefect.CBtail	no value	
set	interface.IFdefect.VBtail	no value	
set	interface.IFdefect.neutral	no value	
set	interface.IFdefect.singledonor	no value	
set	interface.IFdefect.singleacceptor	no value	
set	interface.IFdefect.abovehighestEV	no value	
set	interface.IFdefect.aboveEVleft	no value	
set	interface.IFdefect.belowlowestEC	no value	
set	interface.IFdefect.aboveEileft	no value	
set	interface.IFdefect.aboveEiright	no value	
set	interface.IFdefect.Et		eV
set	interface.IFdefect.Echar		eV
set	interface.IFdefect.Ntotal		cm^{-2}
set	interface.IFdefect.Npeak		$cm^{-2}eV^{-1}$
set	interface.IFdefect.tunneling.on	no value	
set	interface.IFdefect.tunneling.off	no value	
set	interface.IFdefect.tunneling.me		--
set	interface.IFdefect.tunneling.mh		--

calculate – command

Syntax:

`calculate`

No argument or value is required. This is equivalent with pressing “Calculate” in the interactive user interface.

get – commands

Syntax:

`get argument variable`

Here, variable is one of the internal script variables. When you ask for a scalar property, you can use xvalue, yvalue, xvector[*ix*] or yvector[*iy*]: the actual value of the variable will then

be overwritten with the result of the get action. When you use xvector[-1] or yvector[-1], the size of these vectors is incremented by one (thus nx→nx+1 or ny→ny+1), and the actual argument is in the last element of the vector. when you ask for a vectorial properties, like a full *I-V* or *QE* curve, these are placed in xvector and yvector.

The purpose is that the script file, or the program launching the script file (e.g. MatLab, another C-programme, Windows script or MS-DOS command language…) would have access to variables such as V_{oc} , J_{sc}, η, … or even arrays as *J*(*V*), … in a more convenient way then having to retrieve them from a SCAPS output file.

Also, these internal variables can be passed to and updated by the SCAPSUserFunction, that is under the control of the SCAPS user, see later.

command	argument	value and remarks
get solar cell characteristics commands		
get	characteristics.eta	a scalar script value: xvalue or yvalue or xvector[*i*] or yvector[*i*] where the index *i* should be in the range 0 ≤ *i* ≤ nx-1 or 0 ≤ *i* ≤ ny-1. Using *i* = -1 means that the value is appended at the end of xvector or yvector, and that nx or ny are incremented with one. Using xvector or yvector (thus whithout index) means that the size nx or ny is incremented with one, all elements of the vectors are shifted one position up, and the value returned by characteristics… is placed at xvector[0] or yvector[0]
get	characteristics.voc	
get	characteristics.jsc	
get	characteristics.ff	
get	characteristics.vmpp	
get	characteristics.jmpp	
get solar cell characteristics commands		
get	iv	no variables should be passed: the two vectorial script vectors xvector and yvector are always used: xvector contains the abscissa (thus *V* or *f* or λ or *x*), and yvector contains the ordinate (this *I*, *C*, *G* or *QE* or *G*eneration). The sizes nx = ny are set automatically.
get	cv	
get	gv	
get	cf	
get	gf	
get	qe	
get	gx	

loop – commands

Syntax:

`loop argument variable`

On encountering a `loop start` command line, the internal script variables are set to:

loopcounter = 0 and looperror = 1.0E30 (or the value of looperror that was set before).

The next script commands are executed until `loop stop` is met. Then, if loopcounter < maxiteration and looperror > maxerror, the internal script variable loopcounter is incremented, and the script is retaken from the preceding `loop start` command. The internal variables maxiteration and maxerror can be set with `set loop.maxiteration` and `set loop.maxerror` at any time.

There is no set command to set the internal script variable loopcounter. The variable loopcounter is internally set to zero on starting a loop, and then incremented with one each times the loop is run. The variable looperror can be set directly or be returned by the dll programme SCAPSUserFunction.dll, that should be set-up by the user (one example of such dll is distributed with the SCAPS installation). Two of the `loop` commands are equivalent with a set command:

E.g. `loop maxiteration 20` is equivalent to `set scriptvariable.maxiteration 20`

E.g. `loop maxerror 1E-6` is equivalent to `set scriptvariable.maxerror 1E-6`

command	argument	value	
`loop`	`start`	no value	
`loop`	`stop`	no value	
`loop`	`maxiteration`		`max=100; default = 25.`
`loop`	`maxerror`		`min=1E-8; max=1E25; default=1E-5`

clear – commands

Syntax:

`clear argument`

With `clear scriptvariables`, all script variables (or all but 2 or 3 elements) are set to their defaults. `clear simulations` is equivalent to pressing the 'clear all simulations' button in the SCAPS action panel.

command	argument	value	remarks

clear	scriptvariables.all	no value	see text above
clear	scriptvariables.allbutfirst3	no value	leaves xvector[*i*] and yvector[*i*] with *i* = 0, 1, 2. nx and ny are set to 3. The other script variables are not affected.
clear	scriptvariables.allbutfirst2	no value	idem, but with *i* = 0, 1
clear	scriptvariables.allbutlast3	no value	idem, but shifts elements *i* = nx-1, nx-2, nx-3 (or with ny) to *i* = 0, 1, 2 and leaves them
clear	scriptvariables.allbutlast2	no value	idem, but shifts elements *i* = nx-1, nx-2 (or with ny) to *i* = 0, 1 and leaves them
clear	simulations	no value	see text above

The application SCAPSUserFunction.dll

This function is run by

```
rundll scapsuserfunction
```

(As of now, only one user dll is recognized is SCAPS, named SCAPSUserFunction.dll. The format of this command allows possible later addition of more dll's).

This dll is the method that SCAPS is using to implement two-way communication with the user. When you do not (want to) know how to write an own program and make a dll (dynamic link library) of it, you are restricted to use only the SCAPSUserFunction.dll as delivered with SCAPS, or not to use loops in a SCAPS script. The following information is for SCAPS users with programming skills. By writing their own SCAPSUserFunction.dll, they now can realize the following (in the formulation of an external SCAPS user):

"I would need the possibility to do a simulation, evaluate the result with an external program and let it adjust the problem definition for the next simulation, do a simulation, and so on..."

… well, this external program should be named SCAPSUserFunction, and be present as a dll file in the scaps/lib directory. When implemented in C or C^{++}, this function must comply with the function definition:

int DLLIMPORT SCAPSUserFunction (int mode, double *xvalue, double *yvalue, double *xvector, int nx, double *yvector, int ny, double *looperror, char *filename);

The keyword DLLIMPORT might be dependent on the development environment; here it is for LW/CVI of National Instruments.

The meaning of the other items is:

SCAPSUserFunction: the name of the dll. The user must provide a SCAPSUserFunction.dll and SCAPSUserFunction.lib with this name, in the scaps/lib directory.

int SCAPSUserFunction: the function should return an integer value, indicating the success of the function evaluation. SCAPS interprets 0 as 'success' and a negative value as a failure. This value is stored in the script-variable status, and shown in the error output (to screen or in the SCAPS error logfile.

int mode: an integer that can be used to implement several strategies in one dll function. In the example delivered with SCAPS, mode = 1 or 2 means 'find a root' (e.g. find some N_A such that V_{oc} = 0.50 V), and mode 3 or 4 means 'find an extremum' (e.g. find some N_t such that η is maximal).

double *xvalue, double *yvalue: (pointers to) two scalar values, passed to the function by reference, such that a new value of them can be returned by the function.

double *xvector, double *yvector: (ponters to) two one dimensional arrays, one with dimension nx and one with dimension ny. These arrays can get new values in the function that is returned to SCAPS.

int nx, int ny: the dimension of the above values. These are passed by value, not by reference: their value cannot be updated and returned by the function.

double *looperror: a pointer to a scalar variable, that can be updated and returned by the function. In the SCAPS script processor, ot is treated as the internal looperror variable. Returning its value by SCAPSUserFunction.dll is the only way to change looperror in a loop. Since the script processor only checks if |looperror| < maxerror, so you can also return a negative value here.

char *filename: a pointer to a string variable of max. 256 characters. The SCAPS script processor will treat it as a filename, that can be used to set e.g. a spectrum file, a generation file, a filter file,… with the `set` command.

To set up your own dll, you can use other variable names; however, the type, size and order of the variables must be exactly as specified here. Those not using C or C^{++} should use variable types of the same size (in bits) than the C types int, double, char, pointer. Also, the header (.h

file in C) used in the dll should be the same as the corresponding .h file in SCAPS. Users planning to develop an own dll for SCAPS, should ask us for the files SCAPSdll.c and SCAPSdll.h to start from. The actual implementation of SCAPSUserFunction is explained at the bottom of this document.

Executing system commands in a script

The command line to do this is:

```
runsystem systemcommand
```

where `systemcommand` is something that is recognized by MS-Windows as a valid command. These can be .exe files, .bat files or WINDOWS commands. Here you can any of your own programmes (extension .exe; the arguments on the command line can be included), or any of your batch files (extension .bat).

Examples are:

```
runsystem myownopticalprogramme.exe inputfile1 inputfile2 outputfile
runsystem myownwindowsbatchprogramme.bat
runsystem print ivresults.iv
```

(in the last command, it is likely that Windows will need to know the full path and not only the filename…).

Show scriptvariables

The command line to do this is:

```
show scriptvariables
```

These are shown on the screen, if `errorhandling.toscreen` is set, or to the standard error file, if `errorhandling.appendtofile` or `errorhandling.overwritefile` are set. This command is very useful in debugging your script files. You can comment out the `show` commands once the script is OK.

actual implementation of SCAPSUserFunction

The actual implementation of SCAPSUserFunction implements various actions depending of the value of the scriptvariable `mode`.

1. `mode` = 0. Nothing meaningful is done for now: only `filename` = "CdS.abs" is returned. A programmer could replace this part with whatever calculations or manipulations that result in a file to pass back to SCAPS.
2. `mode` = 1 or `mode` = 2. Helps to search the root of a function $y(x)$. During the preceding script commands, the successive evaluations of $y(x)$ are stored in `xvector` and `yvector`, the most recent at `xvector`[0] and `yvector`[0]. SCAPSUserFunction finds a better approximation `xvalue` that would make `yvalue` = y(`xvalue`). During the subsequent sript commands, `xvalue` should be stored in `xvector`[0] (and all existing

elements of `xvector` should be pushed one index up). Then a new calculation should be done, and the result should be stored in `yvector`[0] (pushing the existing elements one place up). Then another call to SCAPSUserFunction can be made to obtain a next, better estimate. Use mode = 1 for a property of 'linear character' (e.g. thickness, bandgap,…) and mode = 2 for a property of 'logarithmic nature' (e.g. a doping density, a trap density).

The difference is: a variable of linear nature is incremented by adding or subtracting something; a variable of logarithmic nature is incremented by multiplying with something. You must provide at least two $y(x)$ points (as elements [0] and [1] of `xvector` and `yvector`) to start with. Of course there is no guarantee at all that such root can be found in your problem! Here is an example:

// find a value of N_t (of the first defect in the first layer) that results in V_{oc} = 0.5 V.

// for some problem and some ill;umination condition to be set first

// the variable N_t is of 'logarithmic nature', thus use `mode` = 2

```
set scriptvariable.mode 2
set scriptvariable.yvalue 0.5000     // the desired value
```

// first initial guess

```
set scriptvariable.xvector[0] 1e14
set layer1.defect1.ntotal xvector[0]
calculate
get characteristics.voc  yvector[0]
```

// second initial guess

```
set scriptvariable.xvector[1] 1e13
set layer1.defect1.ntotal xvector[1]
calculate
get characteristics.voc  yvector[1]
```

// start a loop, do not exagerate with the precision or the number of iterations

```
loop maxiteration 30
loop maxerror     1e-4
loop start
```

// Run the dll that is delivered with SCAPS

```
rundll scapsuserfunction
```

// it returns xvalue as a better guess for the variable N_t, set it to N_t,

```
set layer1.defect1.ntotal xvalue
```

// place this better guess on xvector[0] (and push the rest upward)

```
set scriptvariable.xvector xvalue
calculate
```

// places the new Voc in yvector[0] and pushes the rest up

```
get characteristics.voc  yvector
```

```
loop stop
```

// possible output afterwards

```
show scriptvariables
save results.iv findVoc=0.5V.iv
save graphs.iv.iv findVoc=0.5V.png
```

3. `mode` = 3 or `mode` = 4. Helps to search the maximum of a function $y(x)$. To start with, at least three values (the elements [0], [1] and [2]) of xvector and yvector should have been set. The function proposes a new value for the maximum in xvalue, and rearranges the elements [0], [1] and [2] so that [0] and [1] are closest to the maximum (as it thinks). The next script commands should place this xvalue on top of xvector, evaluate $y(x)$ for this new xvalue and place the result on top of the yvector, and then call SCAPSUserFunction again. Again, there is no guarantee that a maximum will be found in your problem! Here is an example:

// find a value of the thickness in layer1 of some problem that gives maximum J_{sc}.

// a thickness *d* is 'linear nature', and we are looking for a maximum

// thus we set `mode` = 3.

```
set scriptvariable.mode 3
```

// three initial guesses of the parameter *d* (in SCAPS, it is in µm)

```
set scriptvariable.xvector[0] 0.5
set scriptvariable.xvector[1] 1
set scriptvariable.xvector[2] 1.5
```

// assign the parameter to *d* and calculate the function J_{sc}; do so for the 3 guesses

```
set layer1.thickness xvector[0]
calculate
get characteristics.jsc  yvector[0]
set layer1.thickness xvector[1]
calculate
get characteristics.jsc  yvector[1]
set layer1.thickness xvector[2]
calculate
get characteristics.jsc  yvector[2]
```

// start the iteration loop; do not exaggerate with the settings!

```
loop maxiteration 15
loop maxerror     1e-2   // It should be compared to a Jsc of about 30 mA/cm2
loop start
rundll scapsuserfunction
```

// the function returns xvalue as a better guess for the variable

// set this to *d*, place it on xvector[0] (and push the rest upward)

```
set layer1.thickness xvalue
set scriptvariable.xvector xvalue
```
// calculate and place J_{sc} on top of yvector
```
calculate
get characteristics.Jsc  yvector
loop stop
```
// possible output after the end of the iteration loop
```
show scriptvariables
save results.iv findmaximumJsc.iv
save graphs.iv.iv findfindmaximumJsc.png
```

4. `mode` = 5 or `mode` = 6. Helps to search the minimum of a function $y(x)$. It works exactly as the maximum finding algorithm. `mode` = 5 is for a variable of linear nature, and `mode` = 6 for a variable of logarithmic nature.
5. `mode` > 6. Nothing is done, but the SCAPSUserFunction() waits for you to input your ideas of a meaningful user program.

Marc B. 1-9-2010