SCAPS Version 3.0.02, 17 May 2011

This manual describes the available script commands in SCAPS 3.0.02. Most of the commands present here were already present in version 2.9.3. Unfortunately, the script version of 2.9.3 is not entirely compatible with this version. The most prominent incompatibility's is the fact that the dll has changed its prototype and that extra vectors have been added. An entire overview is given in the SCAPS 3.0.00 add-on manual. The additions in SCAPS 3.0.02 are set in blue.

The SCAPS script editor

SCAPS offers a script editor to edit a new script, or load and save an existing script (Ctrl-s works to save the script with the existing filename, without confirmation), see Fig. 1. Even if this script editor is still somewhat rudimentary, it is a very useful tool to develop and adapt SCAPS scripts.

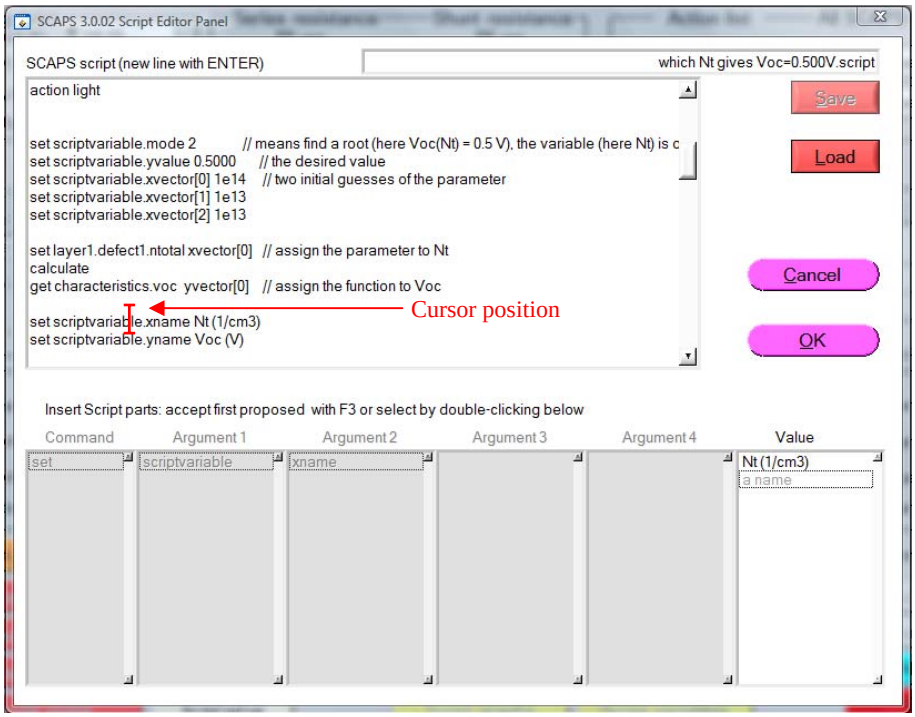

Figure 1: Screen shot of the SCAPS script editor. When placing the cursor in an existing line of a script (as shown), the components of this command line (thus: the command, the arguments and the value) are shown in the six blocks at the bottom of the panel. When typing a new line in a script, the parts of the command line available so far are proposed in these blocks, and can be selected and placed in the editor box of the script.

SCAPS script commands

general

The SCAPS-directory, this is where the scaps.exe file resides, is noted as `scaps\`.

A comment line in a script is a line that cannot be interpreted as a command line. E.g. any line starting with a punctuation character is treated as comment. You can also add comment at the

end of a command line. The Script Editor will recognize such in-line comment when it starts with a double punctuation, except ‘]]’ (thus e.g. ‘//’ or ‘!!’ or ‘>>’ are OK …).

All command lines in a script 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 commands				
`load`	`action`	`set`	`math`	`calculate`
`save`	`clear`	`get`	`loop`	`run`
	`show`			`rundll`
	`plot`			`runsystem`

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. In this list (and in this entire manual) {m} should be replaced by x, y, z, u, v, w, in order to get expressions like xvector, wvector, uvalue, ny , nv, zname, uindex…

name	C-type	default value	max value
{*m*}value	double	0	2 scalar values xvalue and yvalue are now extended with zvalue, …, wvalue in SCAPS3.0.02
{*m*}vector	array of double	0	
n{*m*}	int	0	
{*m*}name	character string	empty	max size 256 bytes
{*m*}index	int	0	6 new indices added in SCAPS3.0.02
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 8 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`
`load`	`singleshotbatch`		`scaps\bdf`

The last argument (`load singleshotbatch`) is slightly deviating from the others as it does not take a value. The purpose of this command is to work together with the command `get recorder`. When `load singleshotbatch` is called the batch settings file *singleshotbatch.sbf* is loaded. This file sets a batch calculation with one calculation at the working point temperature. So it enables you to perform a recording of a singleshot calculation. This option is very useful as a lot of properties can only be accessed in the script through performing a record calculation and taking the value via `get recorder`. In this way you can access e.g. the electrical field distribution in the structure and do calculations with it.

The temperature in this batch is set to the working point value when the command `load singleshotbatch` is called. Hence when you vary the temperature afterwards you should repeat the command again.

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	scriptvariables	a filename	scaps\results
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 the values in the list below, and `value` is a numerical value or a script variable 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 of SCAPS < 2.10); 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.kT		Volt or eV
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 spectrumcutoff.on
action	spectrumcutoff.shortlambda		nm
action	spectrumcutoff.longlambda		nm
action	intensity.ND		
action	intensity.T		%
action	generationfromfile	none	overrides generationfrominternalmodel
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	1 is the default
action	iv.doiv	none	equivalent to action iv.checkaction 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	1 is the default
action	cv.docv	none	equivalent to action cv.checkaction 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	1 is the default
action	cf.docf	none	equivalent to action cf.checkaction 1
action	qe.startlambda		nm
action	qe.stoplambda		nm
action	qe.points	≥ 2	
action	qe.increment		nm

action	qe.checkaction	0 or 1	1 is the default
action	qe.doqe	none	equivalent to action qe.checkaction 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, a script variable 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 mode, loopcounter, maxiteration, {m}index, {m}value, {m}vector and n{m}. Here {m} can be one of the letters x, y, …, w., and n{w} is the number of elements in the corresponding {w}vector.

The values of the internal variables {m}value, {m}vector, … can be set directly with a set-command; also, they are used and possibly changed in SCAPSUserFunction.dll (see later). The value of n{*m*} can be set directly with the set command; it is also updated in some commands: get, math and clear, see later. The allowed indices in SCAPS script vectors are listed in the Table below.

When you set a new value of n{m}, the length of the corresponding vector is updated. If the new value is smaller than the previous one, data gets lost, if it is larger, the vector is extended with uninitialised (random) numbers. Before setting a script variable, you might want to re-initialise them with one of the clear commands, see later.

These conventions for the use of scriptvectors in the set and get (see further) commands are summarised in the Table below.

script vector format	index	meaning; remarks
{m}vector	no index	Only as an argument of set scriptvariable… or as the value of get characteristics… The value of n{m} is incremented, all existing elements of {m}vector are shifted one up, and the value of the set scriptvariable… command, or the parameter to get, is placed at {*m*}vector[0]
{m}vector[-1]	-1	Only as an argument of set scriptvariable… or as the value of get characteristics… The value of n{m} is incremented, and the value of the set scriptvariable… command, or the parameter to get, is placed as the new last

		element of {m}vector
{m}vector[i]	a number	i is an integer number and should be 0 ≤ i ≤ n{m}-1
{m}vector[last]		For your comfort: internally, last is replaced with the appropriate n{m}-1
{m}vector[loopcounter]	a scriptvariable	
{m}vector[mode]	a scriptvariable	
{m}vector[maxiteration]	a scriptvariable	
{m}vector[{n}index]	a scriptvariable	m and n can differ: you can specify e.g. zvector[yindex]
{m}vector[{n}value]	a scriptvariable	m and n can differ: you can specify e.g. uvector[wvalue]. The value of {n}value is first rounded to the nearest integer.
{m}vector[{n}vector[…]]	a scriptvariable	Here the index … of the inner {m}vector takes one of the forms allowed in this Table. You can nest many vectors, but that should not be a reason to exaggerate

The set commands are summarised in the Table below.

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.xvalue		
set	scriptvariable.{*m*}vector[*i*]		0 ≤ *i* ≤ nx - 1, or *i* = -1, or no index
set	scriptvariable.n{*m*}	integer	
set	scriptvariable.{*m*}name	character string	length < 256
set	scriptvariable.filename	character string	length < 256
set	scriptvariable.filename.SCAPSpath	character string	length < 256

The filename is completed to (or changed to) the full default SCAPS path. E.g. the command scriptvariable.filename.SCAPSpath mycell.def will set filename to (e.g.) c:\MB\SCAPS try-outs\def\mycell.def. If no value is given, the actual filename is completed to the SCAPS default path. This command is useful to pass a filename to another programme, that might need to know the full path (e.g. the SCALSdll function).

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.opticalfilter	0 or 1	
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		$cm^2V^{-1}s^{-1}$
set	layer.mup		$cm^2V^{-1}s^{-1}$
set	layer.NA		cm^{-3}
set	layer.ND		cm^{-3}
set	layer.radiative		cm^3s^{-1}
set	layer.Augern		cm^6s^{-1}
set	layer.Augerp		cm^6s^{-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.doubledonor	no value	
set	layer.defect.singleacceptor	no value	
set	layer.defect.doubleacceptor	no value	
set	layer.defect.amphoteric	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 argument`

This is equivalent with pressing one of the "Calculate"-buttons in the interactive user interface. If no argument is present the command gets interpreted as `calculate singleshot`

command	argument	remark
calculate	singleshot	this argument can be omitted
calculate	batch	
calculate	recorder	

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 {m}value or {*m*}vector[*index*]: the actual value of the variable will then be overwritten with the result of the get action. Here *index* is one of the allowed formats for indices in the SCAPS script. Other scalar script variables that can be used are looperror and maxerror.

When you ask for a vectorial properties, like a full *I-V* or *QE* curve, these are placed in two vectors: e.g. *I* in {*m*}vector. and *V* in {*n*}vector. If no vectors are specified, xvector and yvector are assumed: thus `get cv` is identical to `get cv` xy. Also note that only the result of the last simulation is acquired: the last single shot simulation, or the last simulation in a batch run.

The `get` command updates {*m*}name as well.

The purpose of the `get` command is that the script file, or the program launching the script file (e.g. from within SCAPS, from 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 {*m*}vector[*i*] where the index *i* should be in the range $0 \le i \le$ {*m*}x-1. Using *i* = -1 means that the value is appended at the end of {*m*}vector, and that n{*m*} are incremented with one. Using {*m*}vector or (thus without index) means that the size n{*m*} is incremented with one, all elements of the vectors are shifted one position up, and the value returned by `characteristics…` is placed at {*m*}vector[0].
`get`	`characteristics.voc`	
`get`	`characteristics.jsc`	
`get`	`characteristics.ff`	
`get`	`characteristics.vmpp`	
`get`	`characteristics.jmpp`	
get general characteristics		
`get`	`iv`	Two letters should be passed

get	cv	for the value, corresponding with two vectors. The abscissa is saved in the first, the ordinate in the second. (e.g. get cf zw, saves the frequency in the vector zvector and the capacitance in the vector wvector). If the value is omitted, xy is assumed. The sizes n{*m*} are set automatically, and also the names {m}name are set. In the same way, you can now also get a measurement.
get	gv	
get	cf	
get	gf	
get	qe	
get	gx	
get	measurement.iv	
get	measurement.cv	
get	measurement.gv	
get	measurement.cf	
get	measurement.gf	
get	measurement.qe	
get	recombination.tot	
get	recombination.SRH	
get	recombination.rad	
get	recombination.aug	
get	recorder	
get mesh characteristics; layer should be substituted by layer1, layer2, … or layer7		
get	layer.leftindex	the index of the leftmost point in the specified layer
get	layer.leftx	the position *x* (in µm) of the leftmost point in the specified layer
get	layer.rightindex	the index of the rightmost point in the specified layer
get	layer.rightx	the position *x* (in µm) of the rightmost point in the specified layer
get	numberoflayers	the number of layers in the actual cell definition
get	celllength cellength	the total cell length *x* (in µm) of the actual cell definition; both celllength and cellength will work ☺

From SCAPS 3.0.02 on (may 2011), the scalar cell parameters that are available in set are made available in get. When your cell has graded properties, the parameters that you can set or get relate to the

‘pureA’ material (when grading is a function of composition) or to the left side of a layer (when grading is a function of position) (see the SCAPS2.8 add-on manual on grading). The units and remarks are as for the corresponding `set` commands
contact get commands: replace `contact` with either `leftcontact` or `rightcontact`

`get`	`contact.Sn`	`get`	`contact.opticalfilter.file`
`get`	`contact.Sp`	`get`	`contact.opticalfilter`
`get`	`contact.opticalfilter.on`	`get`	`contact.workfunction`
`get`	`contact.opticalfilter.off`	`get`	`contact.flatband.off`
`get`	`contact.opticalfilter.transmission`	`get`	`contact.flatband.once`
`get`	`contact.opticalfilter.reflection`	`get`	`contact.flatband.always`
`get`	`contact.opticalfilter.value`		

layer get commands: replace `layer` with `layer1, layer2, … layer7`

`get`	`layer.thickness`	`get`	`layer.NA`
`get`	`layer.Eg`	`get`	`layer.ND`
`get`	`layer.chi`	`get`	`layer.radiative`
`get`	`layer.epsilon`	`get`	`layer.Augern`
`get`	`layer.NC`	`get`	`layer.Augerp`
`get`	`layer.NV`	`get`	`layer.absorption.file`
`get`	`layer.vthn`	`get`	`layer.absorption.A`
`get`	`layer.vthp`	`get`	`layer.absorption.B`
`get`	`layer.mun`		
`get`	`layer.mup`		

defect get commands: replace `layer` with `layer1`, …, and `defect` with `defect1, defect2` or `defect3`

`get`	`layer.defect.energydistribution`	returns an integer that encodes for single, uniform, Gauß, …
`get`	`layer.defect.chargetype`	returns an integer that encodes for neutral, single donor, ….
`get`	`layer.defect.referencelevel`	returns an integer that encodes for above E_V, below E_C, above E_i
`get`	`layer.defect.Et`	
`get`	`layer.defect.Echar`	
`get`	`layer.defect.Ntotal`	
`get`	`layer.defect.Npeak`	

interface get commands: replace `interface` with `interface1, interface2, … interface6`

get	interface.IBtunneling.off	get	interface.IBtunneling.me
get	interface.IBtunneling.on	get	interface.IBtunneling.mh

interface defect get commands: replace interface with interface1,... and IFdefect with IFdefect1, IFdefect2, IFdefect3	
get interface.IFdefect.energydistribution	returns an integer that encodes for single, uniform, …
get interface.IFdefect.chargetype	returns an integer that encodes for neutral, single donor, …
get interface.IFdefect.referencelevel	returns an integer that encodes for above E_V left, above highest E_V, below lowest E_C, …
get interface.IFdefect.Et	
get interface.IFdefect.Echar	
get interface.IFdefect.Ntotal	
get interface.IFdefect.Npeak	
get interface.IFdefect.tunneling.on	
get interface.IFdefect.tunneling.off	
get interface.IFdefect.tunneling.me	
get interface.IFdefect.tunneling.mh	

Some action commands and some other set commands now also have their corresponding get command			
get	action.workingpoint.temperature	get	action.intensity.T
get	action.workingpoint.kT	get	action.generationfromfile.attenuation
get	action.workingpoint.voltage	get	external.Rs
get	action.workingpoint.frequency	get	external.Rsh
get	action.spectrumcutoff.shortlambda	get	external.Gsh
get	action.spectrumcutoff.longlambda	get	internal.reflection
get	action.intensity.ND	get	internal.transmission

The command get recorder gets the data from the record results, and hence allows to access almost any property in script mode.

The recorded property is selected by the value of the script variable mode (the first property in the record setting list is accessed when mode = 0, the next when mode = 1…).

If the recorded property is a scalar value as a function of the batch calculation (e.g. the open circuit voltage) the abscissa consists of the numbers of the batch calculations. If the recorded property is a vector (e.g. the conduction band profile) the abscissa value is the mesh. In this case only the recorded vector of the last batch calculation is copied to the script variable. In this view, performing a batch with only one calculation using load singleshotbatch is very recommended.

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-1 and looperror > maxerror, the internal script variable loopcounter is incremented, and the script is retaken from the preceding `loop start` command. Thus, when the error condition is never met, loopcounter will successively be set to 0 … maxiteration-1, thus maxiteration values. 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	default-directory
`loop`	`start`	no value	
`loop`	`stop`	no value	

`loop`	`maxiteration`		`min=5;`
			`max=100;`
			`default = 25.`

`loop`	`maxerror`		`min=1E-8;`
			`max=1E25;`
			`default=1E-5`

math – commands

Syntax:

`math argument value`

The math commands allows to perform mathematical operations on the script vectors. The argument is followed by a list of one to four letters form the set {x,y,z,u,v,w}. Uppercase or lower case do not matter; however, for clearness in the description below, we will use upper case letters when vectors are meant, and lower case letters otherwise.

If a variable is a vector, e.g. Y, it is interpreted as Yvector. If a variable is a scalar, e.g. z, it is interpreted as zvalue. If a variable is an index, e.g. w, it is interpreted as windex.

Some operations are on vectors. Then operations are performed element by element and can be performed 'in place' (e.g. A ← A+B) where the original content of A is lost.

command	argument	value	remark
A,B and C represent a vector variable			
a, b, c represent a scalar variable			
i represents an index variable			
math	add	ABC	A = B + C . Vector operation
math	multiply	ABC	A = B * C . Vector operation
math	subtract	ABC	A = B - C . Vector operation
math	divide	ABC	A = B / C . Vector operation
math	exp	AB	A = exp(B) . Vector operation
math	log	AB	A = ln(B) . Vector operation
math	power	ABc	A = B ^ c . Vector operation, A and B are vectors, but c is a scalar
math	vectoradd	ABC	A = B + C . Vector operation. Identical to add
math	vectormultiply	ABC	A = B * C . Vector operation Identical to multiply
math	vectorsubtract	ABC	A = B - C . Vector operation Identical to subtract
math	vectordivide	ABC	A = B / C . Vector operation Identical to divide
math	vectorexp	AB	A = exp(B) . Vector operation Identical to exp
math	vectorlog	AB	A = ln(B) . Vector operation Identical to exp
math	vectorpower	ABc	A = B ^ c . Vector operation, A and B are vectors, but c is a scalar Identical to power
math	scalaradd	abc	a = b + c . Scalar operation
math	scalarmultiply	abc	a = b * c . Scalar operation
math	scalarsubtract	abc	a = b - c . Scalar operation
math	scalardivide	abc	a = b / c . Scalar operation
math	scalarexp	ab	a = exp(b) . Scalar operation
math	scalarlog	ab	a = ln(b) . Scalar operation
math	scalarpower	abc	a = b ^ c . Scalar operation.
math	integrate	ABC	$A(B)=\int_0^B c(B')\,dB'$. Vector operation
math	differentiate	ABC	$A(B)=\frac{dC(B)}{dB}$. Vector operation
math	interpolate	aAbB	When A is considered as a function of B, thus $A_i = A(B_i)$, it returns by interpolation $a = A(b)$

`math`	`closestindex`	iaA	Returns the index i for which the element A_i is closest to a
`math`	`extract`	ABcd	c and d are indices. The elements c up to and including d of vector B are copied into vector A, that now gets dimension d-c+1; the previous contents and dimension of A is lost. The operation can be 'in place': A=B is allowed. This function is useful to pick out the information of one layer from the full x-dependence, when the indices c and d are obtained with `get layer.leftindex` and `get layer.rightindex`.
`math`	`increment`	i	The index i is incremented with one. When i is one letter from {x, y, …, w}, the index is interpreted as xindex, yindex,…, or windex. But you can also use loopcounter, maxiteration, status, mode, or one of the words xindex… written in full.
`math`	`decrement`	i	The index i is incremented with one. i is a SCAPS script index, see the above statement for the valid format.
`math`	`abs`	AB	$A_i = \|B_i\|$. Vector operation
`math`	`vectorabs`	AB	$A_i = \|B_i\|$. Vector operation. Identical to `abs`
`math`	`scalarabs`	ab	$a = \|b\|$. Scalar operation
`math`	`min`	aA	$a = \min(A_i)$
`math`	`max`	aA	$a = \max(A_i)$
`math`	`sum`	aA	$a = \sum_i A_i$
`math`	`average`	aA	$a = \frac{1}{n_A} \sum_{i=0}^{n_A-1} A_i$
`math`	`sumofsquares`	aA	$a = \sum_i A_i^2$
`math`	`averageofsquares`	aA	$a = \frac{1}{n_A} \sum_{i=0}^{n_A-1} A_i^2$
`math`	`product`	aA	$a = \prod_i A_i$
`math`	`geometricaverage`	aA	$a = \left(\prod_i A_i \right)^{1/n_A}$
`math`	`chi_square`	aBCDE	where B contains $X_{measured}$ and C contains $Y_{measured}$; and D contains $X_{calculated}$ and D contains $Y_{calculated}$. Then chi_square is calculated as:

			$\chi^2 = \frac{\sum_i (y_{meas} - y_{calc})^2}{\sum_i (y_{meas})^2}$ The sum is taken at the measurement points $x_{meas,i}$ that fall within the range of the calculated x_{calc} points. y_{calc} is linearly interpolated between two calculated points $x_{calc,j}$ and $x_{calc,j+1}$ at the measured point $x_{maes,i}$. Yhe χ^2 sum is normalised: dimensionless, and should ever become small compared to 1.
math	chi_square_log		The same as chi_square but first the logarithm of (the absolute value of) all y_{maes} and y_{calc} is taken.
math	push	ABC	A = [B , C] A is a concatenation of B and C. B is placed on top of C
math	constant	ABc	A = c; Watch out: c is a scalar, A gets the same length as B. B is only used to set the length of A. AAc is allowed as well.
math	linear	AB	A = [1, 2 ,3…]; A gets the same length as B. B is only used to set the length of A. AA is allowed as well.
math	rangeLin	A	The first point A[0] and the last point A[nA-1] of the vector A are conserved. The points in between are scaled in a linear way between those end points.
math	rangeLog	A	The first point A[0] and the last point A[nA-1] of the vector A are conserved. The points in between are scaled in a logarithmic way between those end points.

The five math commands below are special: they require a composed value. The first part is a vector a letter from {x, y, z, u, v, w} (noted here as A), that stands for the corresponding vector. The next parts of the value must be separated by whitespace (space of tab) from the first part and from each other. They can be a number, or a SCAPS script variable.

math	fillConstant	A constant n	n is the number of points
math	fillLinear	A startvalue stopvalue n	n is the number of points
math	fillLogarithmic	A startvalue stopvalue n	n is the number of points
math	fillLogarithmicPerDecade	A startvalue stopvalue n	n is the number of points per decade
math	force_in_range	a minvalue maxvalue	the value of the scalar a is forced in the range [minvalue, maxvalue]

The last for math commands (the fill-commands) provide a more comfortable way to define a vector size, and fill it.

E.g. the commands below

```
set scriptvariable.maxiteration 11
```

```
set scriptvariable.nx maxiteration
set scriptvariable.xvector[0] 0
set  scriptvariable.xvector[last] 5.0
math RangeLin X
```

now can be replaced with e.g.

```
set scriptvariable.maxiteration 11
math FillLinear X 0.0 5.0 maxiteration
```

or directly with

```
math FillLinear X 0.0 5.0 11
```

plot – commands

Syntax:

`plot argument value`

The plot command works in a similar way as the math command. Graphs which are plotted using this command are drawn on the Script results panel.

command	argument	value	remark
A,B and C represent a vector and should be chosen out of the set {x,y,z,u,v,w}			
`plot`	`draw`	AB	Plot A on the abscissa and B on the ordinate
`plot`		AB	Identical to `plot draw`
`plot`	`drawversusindex`	A	Plot the index *i* on the abscissa and A_i on the ordinate
`plot`	`clear`		Clear the plots drawn by the script. Identical to `clear plot`

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
clear	plot	no value	clears all script graphs; identical with plot clear
clear	scriptgraphs	no value	identical with plot clear or clear plot
clear	actions	no value	unchecks all 4 actions (IV, CV, Cf and QE) and restores the workpoint settings to a default (300 K, 0 V, 1 MHz, 5 pts)
clear	all	no value	clears all simulations, all scriptvariables and all plots: equivalent to clear simulations plus clear scriptvariables.all but not clear actions

---

The application SCAPSUserFunction.dll

This function is run by

rundll scapsuserfunction or

run dll scapsuserfunction or

run dll

(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 *svector[6], int sn[6], 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 use 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. Note with SCAPS 3.0.02: though there are now 6 scalar values xvalue, …, wvalue, only xvalue and yvalue are passed to the SCAPS dll. Also, the new integer variables xindex, …, windex, are not passed to the dll: thus, this dll remains compatible with earlier SCAPS versions.

double *svector[6]: array of (pointers to) one dimensional arrays, with dimensions specified in sn[]. These vectors correspond to the vectors `xvector` (=0), `yvector` (=1), `zvector` (=2), `uvector` (=3), `vvector` (=4), `wvector` (=5). These arrays can get new values in the function that is returned to SCAPS.

int sn[6]: the dimension of the above vectors. 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.

The users who want to develop their own dll, or to modify the existing dll (that is easier to start with ☺), should ask us for the source code: SCAPSdll.c and SCAPSdll.h.

Executing system commands in a script

The command line to do this is:

`runsystem systemcommand` or

`run system sytemcommand`

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…).

Executing a script from within a script

The command line to do this is:

`run script scriptfile`

where `scriptfile` is a file containing a script. You can nest script files (that is, run a script file from within another script file) as you like, but that should not be a reason for exaggeration. All the script variables are transferred from one script to the other, with the exception of some loop-variables loopcounter, looperror, maxerror, maxiterations, which are local to each script file.

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. Also, you can stop the execution of a script when the script variables are shown on the screen. When scripts are nested, you exit all scripts upon clicking 'stop script execution'. You can comment out the `show` commands once the script is OK. The `show scriptvariables` panel is also available from the action panel (the SCAPS main panel) after execution of a script.

The `show` command does not show all values of (very) long vectors, if the output is directed to the standard error file. In order to access these, you should use `save scriptvariables`.

actual implementation of SCAPSUserFunction

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

`mode` = 0. Nothing meaningful is done for now: only a file is returned as `filename`. (at this moment we are using the `mode` = 0 part of the dll for try-outs). A programmer could replace this part with whatever calculations or manipulations that result in a file to pass back to SCAPS.

`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 Nt, set it to Nt,
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
```

`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
```

`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.

`mode` > 6. Nothing is done, but the SCAPSUserFunction() waits for you to input your ideas of a meaningful user program.