wdv-notes 36
Stand: 23.JUN.1993 (5.)
Wiss.Datenverarbeitung
FREIE UNIVERSITÄT BERLIN

Adobe PostScript – Die Operatoren.

© 1990–1998 *Karl-Heinz Dittberner*

Pro

Adobes Seitenbeschreibungs- und Programmiersprache PostScript (kurz „PS“ genannt) besteht aus einer Vielzahl von Operatoren, die vorwiegend auf Objekte auf dem Operand-Stack (Stapelspeicher) wirken. Weitere Stacks sind der Dictionary-, der Execution- und der Graphics-State-Stack.

In diesem Merkblatt wird eine Übersicht der Operatoren gegeben (Stand: Level 1). Alle Operatoren sind fett gedruckt. Vor dem Operator sind die erforderlichen Operanden in der Reihenfolge, wie sie auf dem Stack liegen müssen, angegeben. Dabei bedeutet das „–“, daß für die Operation keine Operanden verlangt werden. Nach dem Operator sind die Resultate auf dem Stack angegeben, wobei das letzte ganz oben liegt. Das Symbol „↦“ bedeutet den Boden des Stacks. Die angegebenen Nummern verweisen auf die Seiten im „Adobe: PostScript Language Reference Manual“ (Addison-Wesley 1986, 5. Auflage, ISBN: 0-201-10174-2, Preis: 22.95 $).

1. Verschiedene Operatoren:

proc	**bind**	proc	Replace operator names in *proc* by operators.	124
—	**null**	null	Push null on operand stack.	189
—	**usertime**	int	Return time in milliseconds.	237
—	**version**	string	Return interpreter version.	237

2. Manipulation des Operanden-Stacks:

any	**pop**	—	Discard top element.	193
any_1 any_2	**exch**	any_2 any_1	Exchange top two elements.	151
any	**dup**	any any	Duplicate top element.	148
$any_1 ... any_n$ n	**copy**	$any_1 ... any_n$ $any_1 ... any_n$	Duplicate top *n* elements.	131
$any_n ... any_0$ n	**index**	$any_n ... any_0$ any_n	Duplicate arbitrary element.	172
$a_{n-1} ... a_0$ n j	**roll**	$a_{(j-1) \bmod n} ... a_0$ $a_{n-1} ... a_{j \bmod n}$	Rool *n* elements up *j* times.	205
↦ $any_1 ... any_n$	**clear**	↦	Discard all elements.	127
↦ $any_1 ... any_n$	**count**	↦ $any_1 ... any_n$ n	Count elements on stack.	132
—	**mark**	mark	Push mark on stack.	184
mark $obj_1 ... obj_n$	**cleartomark**	—	Discard elements down through *mark*.	127
mark $obj_1 ... obj_n$	**counttomark**	mark $obj_1 ... obj_n$ n	Count elements down to *mark*.	133

3. Steuerung:

any	**exec**	—	Execute arbitrary object.	152
bool proc	**if**	—	Execute *proc* if *bool* is true.	169
bool $proc_1$ $proc_2$	**ifelse**	—	Execute $proc_1$ if *bool* is true, $proc_2$ if *bool* is false.	169
init incr limit proc	**for**	—	Execute *proc* with values from *init* by steps of *incr* to *limit*.	160
int proc	**repeat**	—	Execute *proc int* times.	202
proc	**loop**	—	Execute *proc* an indefinite number of times.	182
—	**exit**	—	Exit innermost active loop.	154
—	**stop**	—	Terminate **stopped** context.	226
any	**stopped**	bool	Establish context for catching **stop**.	227
—	**countexecstack**	int	Count elements on exec stack.	133
array	**execstack**	subarray	Copy exec stack into *array*.	152
—	**quit**	—	Terminate interpreter.	197
—	**start**	—	Executed at interpreter startup.	225

4. Virtueller Speicher (VM):

—	**save**	save	Create VM snapshot.	208
save	**restore**	—	Restore VM snapshot.	203
—	**vmstatus**	level used max	Report VM status.	238

5. Geräte und Ausgaben:

—	**showpage**	—	Output and reset current page.	223
—	**copypage**	—	Output current page.	132
matrix width height proc	**banddevice**	—	Install band buffer device.	123
matrix width height proc	**framedevice**	—	Install frame buffer device.	162
—	**nulldevice**	—	Install no-output device.	190
proc	**renderbands**	—	Enumerate bands for output to device.	201

6. Grafikstatus:

—	**gsave**	—	Save graphics state.	166
—	**grestore**	—	Restore graphics state.	165
—	**grestoreall**	—	Restore to bottommost graphics state.	166
—	**initgraphics**	—	Reset graphics state parameters.	173
num	**setlinewidth**	—	Set line width.	219
—	**currentlinewidth**	num	Return current line width.	137
int	**setlinecap**	—	Set shape of line ends for stroke (0=butt, 1=round, 2=square).	217
—	**currentlinecap**	int	Return current line cap.	136
int	**setlinejoin**	—	Set shape of corners foir stroke (0=miter, 1=round, 2=bevel).	218
—	**currentlinejoin**	int	Return current line join.	137
num	**setmiterlimit**	—	Set miter length limit.	220
—	**currentmiterlimit**	num	Return current miter limit.	137
array offset	**setdash**	—	Set dash pattern for stroking.	214
—	**currentdash**	array offset	Return current dash pattern.	134
num	**setflat**	—	Set flatness tolerance.	215
—	**currentflat**	num	Return current flatness.	135
num	**setgray**	—	Set color to gray value from 0 (black) to 1 (white).	216
—	**currentgray**	num	Return current gray value.	136
hue sat brt	**sethsbcolor**	—	Set color given hue, saturation, brightness.	216
—	**currenthsbcolor**	hue sat brt	Return current HSB color.	136
red green blue	**setrgbcolor**	—	Set color given RGB values.	221
—	**currentrgbcolor**	red green blue	Return current RGB color.	138
freq angle proc	**setscreen**	—	Set halftone screen.	221
—	**currentscreen**	freq angle proc	Return current halftone screen.	139
proc	**settransfer**	—	Set gray transfer function.	222
—	**currenttransfer**	proc	Return current transfer function.	139

7. Zeichenpfade erzeugen:

—	**newpath**	—	Initialize current path to be empty.	187
—	**currentpoint**	x y	Return current point coordinate.	138
x y	**moveto**	—	Set current point to (*x*, *y*).	186
dx dy	**rmoveto**	—	Relative moveto.	204
x y	**lineto**	—	Append straight line to (*x*, *y*).	180
dx dy	**rlineto**	—	Relative lineto.	204
x y r ang_1 ang_2	**arc**	—	Append counterclockwise arc.	117
x y r ang_1 ang_2	**arcn**	—	Append clockwise arc.	118
x_1 y_1 x_2 y_2 r	**arcto**	xt_1 yt_1 xt_2 yt_2	Append tangent arc.	119
x_1 y_1 x_2 y_2 x_3 y_3	**curveto**	—	Appen Bezier cubic section.	140
dx_1 dy_1 dx_2 dy_2 dx_3 dy_3	**rcurveto**	—	Relative curveto.	198
—	**closepath**	—	Connect subpath back to its starting point.	130
—	**flattenpath**	—	Convert curves to sequences of straight lines.	157
—	**reversepath**	—	Reverse direction of current path.	203
—	**strokepath**	—	Compute outline of stroked path.	229
string bool	**charpath**	—	Append character outline to the current path.	127
—	**clippath**	—	Set current path to clipping path.	129
—	**pathbbox**	ll_x ll_y ur_x ur_y	Return bounding box of current path.	191
move line curve close	**pathforall**	—	Enumerate current path.	192
—	**initclip**	—	Set clip path to device default.	172
—	**clip**	—	Establish new clipping path.	128
—	**eoclip**	—	Clip using even-odd inside rule.	149

8. Datentyp und Typumwandlungen:

any	**type**	name	Return *any*'s data type.	235
any	**cvlit**	any	Make object be literal.	141
any	**cvx**	any	Make object be executable.	144
any	**xcheck**	bool	Test executable attribute.	240
array|file|string	**executeonly**	array|file|string	Reduce access to execute-only.	153
array|dict|file|string	**noaccess**	array|dict|file|string	Disallow any access.	188
array|dict|file|string	**readonly**	array|dict|file|string	Reduce access to read-only.	200
array|dict|file|string	**rcheck**	bool	Test read access.	198
array|dict|file|string	**wcheck**	bool	Test write access.	238
num|string	**cvi**	int	Convert to integer.	141
string	**cvn**	name	Convert to name.	142
num|string	**cvr**	real	Convert to real.	142
num radix string	**cvrs**	substring	Convert to string with radix.	143
any string	**cvs**	substring	Convert to string.	144

9. Dateien (Files):

$string_1$ $string_2$	**file**	file	Open file identified by $string_1$ with access $string_2$.	155
file	**closefile**	—	Close file.	129
file	**read**	int true *or* false	Read one character from *file*.	198
file int	**write**	—	Write one character to *file*.	239
file string	**readhexstring**	substring bool	Read hex from *file* into *string*.	199
file string	**writehexstring**	—	Write *string* to *file* as hex.	240
file string	**readstring**	substring bool	Read *string* from *file*.	201
file string	**writestring**	—	Write characters of *string* to *file*.	240
file string	**readline**	substring bool	Read line from *file* into *string*.	200
file	**token**	token true *or* false	Read token from *file*.	232
file	**bytesavailable**	int	Number of bytes available.	125
—	**flush**	—	Send buffered data to standard output file.	158
file	**flushfile**	—	Send buffered data or read to EOF (end-of-file).	158
file	**resetfile**	—	Discard buffered characters.	202

file	**status**	bool	Return status of *file*.	226
string	**run**	—	Execute contents of named file.	207
—	**currentfile**	file	Return file currently executed.	135
string	**print**	—	Write characters of *string* to standard output file.	193
any	**=**	—	Write text representation of *any* to standard output file.	114
↦ any_1…any_n	**stack**	↦ any_1…any_n	Print stack using =.	224
any	**==**	—	Write syntactic representation of *any* to standard output file.	114
↦ any_1…any_n	**pstack**	↦ any_1…any_n	Print stack using ==.	194
—	**prompt**	—	Executed when ready for interactive input.	194
bool	**echo**	—	Turn on/off echoing.	149

10. Malen:

—	**erasepage**	—	Paint current page white.	151
—	**fill**	—	Fill current path with current color.	156
—	**eofill**	—	Fill using even-odd rule.	150
—	**stroke**	—	Draw line along current path.	229
width height bits/sample matrix proc	**image**	—	Render sampled image onto current page.	170
width height invert matrix proc	**imagemask**	—	Render mask onto current page.	171

11. PS-Bibliotheken (Dictionaries):

int	**dict**	dict	Create dictionary with capacity for *int* elements.	146
dict	**length**	int	Number of key-value pairs in *dict*.	179
dict	**maxlength**	int	Capacity of *dict*.	185
dict	**begin**	—	Push *dict* on dict stack.	124
—	**end**	—	Pop dict stack.	149
key value	**def**	—	Associate *key* and *value* in current dict.	145
key	**load**	value	Search dict stack for *key* and return associated *value*.	181
key value	**store**	—	Replace topmost definition *key*.	227
dict key	**get**	any	Get value associated with *key* in *dict*.	164
dict key value	**put**	—	Associate *key* with *value* in *dict*.	195
dict key	**known**	bool	Test whether *key* is in *dict*.	177
key	**where**	dict true *or* false	Find dict in which *key* is defined.	238
$dict_1$ $dict_2$	**copy**	$dict_2$	Copy contents of $dict_1$ to $dict_2$.	131
dict proc	**forall**	—	Execute *proc* for each element of *dict*.	161
—	**errordict**	dict	Push **errordict** on op stack.	151
—	**systemdict**	dict	Push **systemdict** on op stack.	231
—	**userdict**	dict	Push **userdict** on operand stack.	236
—	**currentdict**	dict	Push current dict on op stack.	134
—	**countdictstack**	int	Count elements on dict stack.	133
array	**dictstack**	subarray	Copy dict stack into *array*.	147

12. Arithmetische Operationen:

num_1 num_2	**add**	sum	num_1 + num_2.	115
num_1 num_2	**div**	quotient	num_1 / num_2.	148
int_1 int_2	**idiv**	quotient	Integer divide: num_1 / num_2.	168
int_1 int_2	**mod**	remainder	int_1 modulo int_2.	185
num_1 num_2	**mul**	product	num_1 • num_2.	186
num_1 num_2	**sub**	difference	num_1 – num_2.	230
num_1	**abs**	num_2	Absolute value of num_1.	115
num_1	**neg**	num_2	Negative of num_1.	187
num_1	**ceiling**	num_2	Ceiling of num_1.	126
num_1	**floor**	num_2	Floor of num_1.	157
num_1	**round**	num_2	Round num_1 to nearest integer.	206
num_1	**truncate**	num_2	Remove fractional part of num_1.	234
num	**sqrt**	real	Square root of *num*.	224
num den	**atan**	angle	Arctangent of *num*/*den* in degrees.	121
angle	**cos**	real	Cosine of *angle* in degrees.	132
angle	**sin**	real	Sine of *angle* in degrees.	223
base exponent	**exp**	real	Raise *base* to *exponent* power.	154
num	**ln**	real	Natural logarithm (base e).	180
num	**log**	real	Logarithm (base 10).	181
—	**rand**	int	Create pseudo-random integer.	197
int	**srand**	—	Set random number seed.	224
—	**rrand**	int	Return random number seed.	207

13. Felder (Arrays):

int	**array**	array	Create array of length *int*.	120
—	**[**	mark	Start array construction.	113
mark obj_0…obj_{n-1}	**]**	array	End array construction.	113
array	**length**	int	Number of elements in *array*.	179
array index	**get**	any	Get array element indexed by *index*.	164
array index any	**put**	—	Put *any* into *array* at *index*.	195
array index count	**getinterval**	subarray	Subarray of *array* starting at *index* for *count* elements.	165
$array_1$ index $array_2$	**putinterval**	—	Replace subarray of $array_1$ starting at *index* by $array_2$.	196
array	**aload**	a_0…a_{n-1} array	Push full *array* on stack.	115
any_0…any_{n-1} array	**astore**	array	Pop elements from stack into *array*.	121
$array_1$ $array_2$	**copy**	$subarray_2$	Copy elements of $array_1$ to initial subarray of $array_2$.	131
array proc	**forall**	—	Execute *proc* for each element of *array*.	161

14. Matrizen und Koordinatentransformationen:

—	**matrix**	matrix	Create identy matrix.	184
—	**initmatrix**	—	Set CTM (Current Transformation Matrix) to device default.	174
matrix	**identmatrix**	matrix	Fill *matrix* with identiy transform.	167
matrix	**defaultmatrix**	matrix	Fill *matrix* with device default matrix.	145
matrix	**currentmatrix**	matrix	Fill *matrix* with CTM.	137
matrix	**setmatrix**	—	Replace CTM by *matrix*.	219
t_x t_y	**translate**	—	Translate user space by (t_x, t_y).	233
t_x t_y matrix	**translate**	matrix	Define translation by (t_x, t_y).	233
s_x s_y	**scale**	—	Scale user space by s_x and s_y.	209
s_x s_y matrix	**scale**	matrix	Define scaling by s_x and s_y.	209
angle	**rotate**	—	Rotate user space by *angle* degrees.	206
angle matrix	**rotate**	matrix	Define rotation by *angle* degrees.	206
matrix	**concat**	—	Replace CTM by *matrix* × CTM.	130
$matrix_1$ $matrix_2$	**concatmatrix**	$matrix_3$	Fill $matrix_3$ with $matrix_1$ × $matrix_2$.	130
x y	**transform**	x' y'	Transform (*x*, *y*) by CTM.	233
x y matrix	**transform**	x' y'	Transform (*x*, *y*) by *matrix*.	233
dx dy	**dtransform**	dx' dy'	Transform distance (*dx*, *dy*) by CTM.	148
dx dy matrix	**dtransform**	dx' dy'	Transform distance (*dx*, *dy*) by *matrix*.	148
x' y'	**itransform**	x y	Inverse transform (*x'*, *y'*) by CTM.	176
x' y' matrix	**itransform**	x y	Inverse transform (*x'*, *y'*) by *matrix*.	176
dx' dy'	**idtransform**	dx dy	Inverse transform distance (*dx'*, *dy'*) by CTM.	168
dx' dy' matrix	**idtransform**	dx dy	Inverse transform distance (*dx'*, *dy'*) by *matrix*.	168
$matrix_1$ $matrix_2$	**invertmatrix**	$matrix_2$	Fill $matrix_2$ with inverse of $matrix_1$.	175

15. Zeichenketten (Strings):

int	**string**	string	Create string of length *int*.	228
string	**length**	int	Number of elements in *string*.	179
string index	**get**	int	Get string element indexed by *index*.	164
string index int	**put**	—	Put *int* into *string* at *index*.	195
string index count	**getinterval**	substring	Substring of *string* starting at *index* for *count* elements.	165
$string_1$ index $string_2$	**putinterval**	—	Replace substring of $string_1$ starting at *index* by $string_2$.	196
$string_1$ $string_2$	**copy**	$substring_2$	Copy elements of $string_1$ to initial substring of $string_2$.	131
string proc	**forall**	—	Execute *proc* for all element of *string*.	161
string seek	**anchorsearch**	post match true *or* string false	Determine if *seek* is initial substring of *string*.	116
string seek	**search**	post match pre true *or* string false	Search for *seek* in *string*.	211
string	**token**	post token true *or* false	Read token from start of *string*.	232

16. Logische Operationen (Booleans):

any_1 any_2	**eq**	bool	Test equal.	150
any_1 any_2	**ne**	bool	Test not equal.	186
num_1 | str_1 num_2 | str_2	**ge**	bool	Test greater or equal.	163
num_1 | str_1 num_2 | str_2	**gt**	bool	Test greater than.	167
num_1 | str_1 num_2 | str_2	**le**	bool	Test less or equal.	179
num_1 | str_1 num_2 | str_2	**lt**	bool	Test less than.	182
$bool_1$ | int_1 $bool_2$ | int_2	**and**	$bool_3$ | int_3	Logical | bitwise AND.	116
$bool_1$ | int_1	**not**	$bool_2$ | int_2	Logical | bitwise NOT.	189
$bool_1$ | int_1 $bool_2$ | int_2	**or**	$bool_3$ | int_3	Logical | bitwise Inclusive OR.	191
$bool_1$ | int_1 $bool_2$ | int_2	**xor**	$bool_3$ | int_3	Logical | bitwise Exclusive OR.	241
—	**true**	true	Push boolean value *true*.	234
—	**false**	false	Push boolean value *false*.	155
int_1 shift	**bitshift**	int_2	Bitwise shift of int_1 (positive is left).	125

17. Fonts und Zeichen:

key font	**definefont**	font	Register *font* as a font dictionary.	146
key	**findfont**	font	Return font dixt identified by *key*.	156
font scale	**scalefont**	font'	Scale *font* by *scale* to produce the new *font'*.	210
font matrix	**makefont**	font'	Transform *font* by *matrix* to produce the new *font'*.	183
font	**setfont**	—	Set font dictionary.	215
—	**currentfont**	font	Return current font dictionary.	136
string	**show**	—	Print characters of *string* on page.	222
a_x a_y string	**ashow**	—	Add (a_x, a_y) to width of each char while showing *string*.	120
c_x c_y char string	**widthshow**	—	Add (c_x, c_y) to width of char while showing string.	239
c_x c_y char a_x a_y string	**awidthshow**	—	Combined effect of ashow and widthshow.	122
proc string	**kshow**	—	Execute *proc* between characters shown from *string*.	178
string	**stringwidth**	w_x w_y	Width of *string* in current font.	228
—	**FontDirectory**	dict	Dictionary of font dictionaries.	159
—	**StandardEncoding**	array	Standard font encoding vector.	225
—	**cachestatus**	bsize bmax msize mmax csize cmax blimit	Return cache status and parameters.	126
w_x w_y ll_x ll_y ur_x ur_y	**setcachedevice**	—	Declare cached chatacter metrics.	212
w_x w_y	**setcharwidth**	—	Declare uncached character metrics.	213
num	**setcachelimit**	—	Set max bytes in cached character.	213

Notizen: