

**rpl**

<b>COLLABORATORS</b>
----------------------

	TITLE : rpl		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		July 25, 2024	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>rpl</b>	<b>1</b>
1.1	rpl.guide . . . . .	1
1.2	newfeatures . . . . .	1
1.3	kernel . . . . .	3
1.4	creation . . . . .	3
1.5	modification . . . . .	4
1.6	objects . . . . .	4
1.7	animation . . . . .	4
1.8	io . . . . .	4
1.9	materials . . . . .	5
1.10	miscs . . . . .	5
1.11	userinterface . . . . .	5
1.12	arexx . . . . .	5
1.13	vectors . . . . .	6
1.14	data types . . . . .	6
1.15	( . . . . .	6
1.16	) . . . . .	7
1.17	. . . . .	7
1.18	.s . . . . .	8
1.19	! . . . . .	8
1.20	& . . . . .	9
1.21	+ . . . . .	9
1.22	- . . . . .	10
1.23	* . . . . .	10
1.24	divide . . . . .	10
1.25	word . . . . .	11
1.26	; . . . . .	11
1.27	< . . . . .	12
1.28	<= . . . . .	12
1.29	<> . . . . .	12

---

1.30 = . . . . .	13
1.31 > . . . . .	13
1.32 >= . . . . .	14
1.33 >r . . . . .	14
1.34 >rad . . . . .	15
1.35 ?& . . . . .	15
1.36 ?dup . . . . .	16
1.37 ?else . . . . .	16
1.38 ?endif . . . . .	17
1.39 ?if . . . . .	17
1.40 @ . . . . .	18
1.41 again . . . . .	18
1.42 and . . . . .	19
1.43 acos . . . . .	19
1.44 asin . . . . .	20
1.45 atan . . . . .	20
1.46 b. . . . .	21
1.47 emplate . . . . .	21
1.48 b@ . . . . .	22
1.49 band . . . . .	22
1.50 begin . . . . .	23
1.51 bnot . . . . .	24
1.52 bor . . . . .	24
1.53 bxor . . . . .	25
1.54 cat . . . . .	25
1.55 constant . . . . .	26
1.56 cos . . . . .	26
1.57 cpy . . . . .	27
1.58 depth . . . . .	27
1.59 do . . . . .	28
1.60 drop . . . . .	28
1.61 dup . . . . .	29
1.62 else . . . . .	29
1.63 emit . . . . .	30
1.64 endif . . . . .	30
1.65 error . . . . .	30
1.66 eval . . . . .	31
1.67 execute . . . . .	37
1.68 exit . . . . .	37

---

1.69 exp . . . . .	38
1.70 f. . . . .	38
1.71 f! . . . . .	39
1.72 f+ . . . . .	39
1.73 f- . . . . .	39
1.74 f* . . . . .	40
1.75 fdivide . . . . .	40
1.76 f< . . . . .	40
1.77 f<= . . . . .	41
1.78 f<> . . . . .	41
1.79 f= . . . . .	41
1.80 f> . . . . .	42
1.81 f>= . . . . .	42
1.82 f>i . . . . .	42
1.83 f@ . . . . .	43
1.84 fconstant . . . . .	43
1.85 fmod . . . . .	44
1.86 forget . . . . .	44
1.87 fvariable . . . . .	44
1.88 h. . . . .	45
1.89 i . . . . .	45
1.90 i>f . . . . .	46
1.91 if . . . . .	46
1.92 j . . . . .	47
1.93 k . . . . .	48
1.94 leave . . . . .	48
1.95 load . . . . .	49
1.96 log . . . . .	49
1.97 log10 . . . . .	50
1.98 loop . . . . .	50
1.99 +loop . . . . .	50
1.100mod . . . . .	51
1.101not . . . . .	51
1.102o. . . . .	52
1.103or . . . . .	52
1.104over . . . . .	53
1.105pick . . . . .	53
1.106pow . . . . .	53
1.107puts . . . . .	54

1.108quit . . . . .	54
1.109r0 . . . . .	55
1.110r> . . . . .	55
1.111random . . . . .	55
1.112rdepth . . . . .	56
1.113repeat . . . . .	56
1.114roll . . . . .	57
1.115rot . . . . .	57
1.116s0 . . . . .	57
1.117sin . . . . .	58
1.118sqrt . . . . .	58
1.119sprintf . . . . .	59
1.120string . . . . .	60
1.121swap . . . . .	61
1.122tan . . . . .	61
1.123until . . . . .	61
1.124variable . . . . .	62
1.125vlist . . . . .	62
1.126w! . . . . .	63
1.127w@ . . . . .	63
1.128while . . . . .	64
1.129xor . . . . .	64
1.130c_aimpoint . . . . .	65
1.131c_attrib . . . . .	65
1.132c_cone . . . . .	66
1.133c_coordsys . . . . .	66
1.134c_cube . . . . .	67
1.135c_cutcone . . . . .	67
1.136c_cutpolymid . . . . .	68
1.137c_cutpyramid . . . . .	69
1.138c_cylinder . . . . .	69
1.139c_ellipse . . . . .	70
1.140c_ellipseg . . . . .	70
1.141c_ellipsoid . . . . .	70
1.142c_group . . . . .	71
1.143c_hyperbol . . . . .	72
1.144c_level . . . . .	72
1.145c_line . . . . .	73
1.146c_link . . . . .	74

---

1.147c_mesh . . . . .	74
1.148c_offset . . . . .	75
1.149c_polygon . . . . .	76
1.150c_polyhedron . . . . .	76
1.151c_polymid . . . . .	77
1.152c_pyramid . . . . .	77
1.153c_rectangle . . . . .	77
1.154c_triset . . . . .	78
1.155c_viewpoint . . . . .	78
1.156m_alpha . . . . .	79
1.157m_color . . . . .	80
1.158m_copy . . . . .	80
1.159m_cut . . . . .	80
1.160m_delete . . . . .	81
1.161m_duplicate . . . . .	81
1.162m_extend . . . . .	82
1.163m_mirror . . . . .	82
1.164m_move . . . . .	83
1.165m_movecog . . . . .	83
1.166m_name . . . . .	84
1.167m_paste . . . . .	84
1.168m_rotate . . . . .	85
1.169m_size2d . . . . .	85
1.170m_shear . . . . .	86
1.171m_size3d . . . . .	87
1.172m_stretch . . . . .	87
1.173m_swap . . . . .	88
1.174o_creatag . . . . .	88
1.175o_current . . . . .	89
1.176o_delete . . . . .	89
1.177o_deriv . . . . .	90
1.178o_eval . . . . .	90
1.179o_find . . . . .	91
1.180o_findtag . . . . .	91
1.181o_findwild . . . . .	92
1.182o_getcur . . . . .	93
1.183o_getnext . . . . .	93
1.184o_getpar . . . . .	94
1.185o_getprev . . . . .	94

---

1.186o_getsel . . . . .	95
1.187o_getsub . . . . .	95
1.188o_lock . . . . .	95
1.189o_prop . . . . .	96
1.190o_scan . . . . .	97
1.191o_select . . . . .	98
1.192aspec . . . . .	98
1.193play . . . . .	99
1.194mth_create . . . . .	100
1.195mth_delete . . . . .	101
1.196mth_find . . . . .	101
1.197fil_load . . . . .	102
1.198fil_save . . . . .	102
1.199mat_create . . . . .	103
1.200mat_delete . . . . .	104
1.201mat_find . . . . .	105
1.202mat_lock . . . . .	105
1.203err_install . . . . .	105
1.204err_remove . . . . .	107
1.205mem_alloc . . . . .	107
1.206mem_free . . . . .	108
1.207inherit . . . . .	109
1.208menu . . . . .	109
1.209refresh . . . . .	110
1.210render . . . . .	110
1.211rot_coord . . . . .	110
1.212scr_save . . . . .	111
1.213system . . . . .	111
1.214wnd_addr . . . . .	111
1.215wnd_open . . . . .	112
1.216wnd_sendmsg . . . . .	113
1.217ray_inters . . . . .	114
1.218ray_free . . . . .	115
1.219ray_prep . . . . .	116
1.220busy_cancel . . . . .	116
1.221busy_close . . . . .	117
1.222busy_open . . . . .	118
1.223busy_update . . . . .	119
1.224get_key . . . . .	119

---



1.225get_flt . . . . .	120
1.226get_str . . . . .	121
1.227get_vect . . . . .	122
1.228get_file . . . . .	122
1.229rx . . . . .	123
1.230rx_rc . . . . .	124
1.231rx_result . . . . .	125
1.232rx_setclip . . . . .	125
1.233vvariable . . . . .	126
1.234v! . . . . .	127
1.235v@ . . . . .	127
1.236vadd . . . . .	128
1.237vsub . . . . .	128
1.238vmul . . . . .	129
1.239vdot . . . . .	129
1.240vcros . . . . .	130
1.241vnorm . . . . .	130
1.242vlen . . . . .	131
1.243v. . . . .	131
1.244vconstant . . . . .	132
1.245geometry . . . . .	132
1.246[fstangle fenangle] . . . . .	132
1.247wgeomflags . . . . .	133
1.248wfreetype . . . . .	133
1.249icolor . . . . .	134
1.250attributes . . . . .	134
1.251creation tags . . . . .	135
1.252modify flags . . . . .	135
1.253getvstack . . . . .	135
1.254database . . . . .	136
1.255wnd_lock . . . . .	136
1.256inside_prep . . . . .	137
1.257inside_test . . . . .	137
1.258inside_free . . . . .	138

---

# Chapter 1

## rpl

### 1.1 rpl.guide

REAL 3D PROGRAMMING LANGUAGE - RPL

New Features of V.2.40

Data Types

Kernel Words

Object Creation

Object Modification

Object Manipulation

Animation

Real 3D File I/O

Materials

User Interface

ARexx

Vector Operations

Miscellaneous Words

### 1.2 newfeatures

New Features of V.2.4x

=====

NEW AND MODIFIED RPL WORDS

DATABASE

EVAL

GETVSTACK

INSIDE\_PREP

INSIDE\_TEST

INSIDE\_FREE

WND\_LOCK

---

! AND @ WORDS

-----

RPL words ! and @ and corresponding words for other data types have been changed to STORE and FETCH for readability reasons. The original words are defined in the new rpl-startup and vectors.rpl files. If you encounter problems in executing old RPL scripts, replace ! by FETCH and @ by STORE or include the appropriate word definitions.

#### CHANGES TO RPL LOCKING SYSTEM

-----

One change has been made to the locking system: macro execution no longer locks the object data structure. O\_LOCK calls are now added to the macro files automatically by the macro recorder. This makes the macro execution identical with executing RPL programs from RPL windows.

The rule from now on is that locking MUST be used whenever the object data structure is manipulated.

The words with which the object data structure must be locked, are:

C\_XXX, O\_XXX, M\_XXXX, RAY\_XXX, INSIDE\_XXX.

For example:

```
"r3d2:rpl/sys/locks.rpl" LOAD    ( iLOCK_EXCL/SHARED/REMOVE defined here

iLOCK_EXCL O_LOCK                ( lock object data structure

O_GETSEL                        ( access object
0.1 0 0 0 M_MOVE

iLOCK_REMOVE O_LOCK             ( release object data structure

REFRESH                         ( refresh all windows
```

#### DEAD LOCKS

Note that you must NOT use locking carelessly, because it can lead to dead lock situations. In other words, it is possible to call a word which attempts to lock the object data structure but cannot obtain the lock because it is already locked. For example, the following example can lead to a dead lock situation:

```
iLOCK_EXCL O_LOCK                ( lock
REFRESH                          ( send refresh message to all windows
iLOCK_REMOVE O_LOCK             ( unlock
```

The reason for this is that the word REFRESH makes windows refresh their contents which involves that each window which displays object related information must use locking. Because the object data is already locked by the RPL script which caused the refresh, windows never obtain the lock and the call 'REFRESH' never returns.

Therefore, use O\_LOCK only with the words listed above.

## 1.3 kernel

### RPL KERNEL WORDS

(	)	.	.S	!
&	+	-	*	/
:	;	<	<=	=
>	>=	<>	>R	>RAD
?&	?DUP	?ELSE	?ENDIF	?IF
@	AGAIN	AND	ACOS	ASIN
ATAN	B.	B@	BAND	BEGIN
BNOT	BOR	BXOR	CAT	CONSTANT
COS	CPY	DEPTH	DO	DROP
DUP	ELSE	EMIT	ENDIF	ERROR
EVAL	EXECUTE	EXIT	EXP	F.
F!	F+	F-	F*	F/
F<	F<=	F<>	F=	F>
F>=	F>I	F@	FCONSTANT	FMOD
FORGET	FVARIABLE	H.	I	I>F
IF	J	K	LEAVE	LOAD
LOG	LOG10	LOOP	+LOOP	MOD
NOT	O.	OR	OVER	PICK
POW	PUTS	QUIT	R0	R>
RANDOM	RDEPTH	REPEAT	ROLL	ROT
S0	SIN	SQRT	SPRINTF	STRING
SWAP	TAN	UNTIL	VARIABLE	VLIST
W!	W@	WHILE	XOR	

## 1.4 creation

### OBJECT CREATION WORDS

C_AIMPOINT	C_ATTRIB
C_CONE	C_COORDSYS
C_CUBE	C_CUTCONE
C_CUTPOLYMID	C_CUTPYRAMID
C_CYLINDER	C_ELLIPSE
C_ELLIPSEG	C_ELLIPSOID
C_GROUP	C_HYPERBOL
C_LEVEL	C_LINE
C_LINK	C_MESH
C_OFFSET	C_POLYGON
C_POLYHEDRON	C_POLYMID
C_PYRAMID	C_RECTANGLE
C_TRISET	C_VIEWPOINT

## 1.5 modification

### MODIFICATION WORDS

M_ALPHA	M_COLOR
M_COPY	M_CUT
M_DELETE	M_DUPLICATE
M_EXTEND	M_MIRROR
M_MOVE	M_MOVECOG
M_NAME	M_PASTE
M_ROTATE	M_SIZE2D
M_SHEAR	M_SIZE3D
M_STRETCH	M_SWAP

## 1.6 objects

### OBJECT SPECIFIC WORDS

O_CREATAG	O_CURRENT
O_DELETE	O_DERIV
O_EVAL	O_FIND
O_FINDTAG	O_FINDWILD
O_GETCUR	O_GETNEXT
O_GETPAR	O_GETPREV
O_GETSEL	O_GETSUB
O_LOCK	O_PROP
O_SCAN	O_SELECT

## 1.7 animation

### ANIMATION SPECIFIC WORDS

ASPEC  
PLAY  
MTH\_CREATE  
MTH\_DELETE

## 1.8 io

### REAL 3D BINARY FILE FORMAT I/O

FIL\_LOAD  
FIL\_SAVE

## 1.9 materials

### MATERIAL SPECIFIC WORDS

MAT\_CREATE  
MAT\_DELETE  
MAT\_FIND  
MAT\_LOCK

## 1.10 miscs

### MISCELLANEOUS WORDS

BUSY_CANCEL	RAY_INTERS
BUSY_CLOSE	RAY_FREE
BUSY_OPEN	RAY_PREP
BUSY_UPDATE	REFRESH
DATABASE	RENDER
ERR_INSTALL	ROT_COORD
ERR_REMOVE	SCR_SAVE
GETVSTACK	SYSTEM
INHERIT	WND_ADDR
MEM_ALLOC	WND_LOCK
MEM_FREE	WND_OPEN
MENU	WND_SENDMSG

## 1.11 userinterface

### USER INTERFACE WORDS

GET\_KEY  
GET\_STR  
GET\_FLT  
GET\_VECT  
GET\_FILE

## 1.12 arexx

### AREXX WORDS

RX  
RX\_RC  
RX\_RESULT  
RX\_SETCLIP

---

## 1.13 vectors

### VECTOR OPERATIONS

VVARIABLE  
V!  
V@  
VADD  
VSUB  
VMUL  
VDOT  
VCROS  
VNORM  
VLEN  
V.  
VCONSTANT

## 1.14 data types

### RPL DATA TYPES

a - generic address

b - byte (8 bits)

e - either integer or floating-point value

f - floating-point value

i - long integer value (32 bits)

l - boolean flag. 0 denotes FALSE, any other value denotes TRUE.

s - address of a string

v - vector (consist of three floating-point values)

w - short integer value (16 bits)

CFA - Code Field Address. The address of a defined word on the Vocabulary Stack.

name - The text for an RPL token (word or variable).

NULL - The integer value zero.

## 1.15 (

### WORD

(

---

## TEMPLATE

```
(
```

## DESCRIPTION

This defines the start of an RPL comment. All the text after it until either `' ) '` or EOL is ignored.

## NOTE

The two comment control words `' ( ' and ' ) '` are defined internally and do not appear as part of the vocabulary.

## EXAMPLE

```
( this is a comment )  
VLIST ( list vocabulary
```

## 1.16 )

## WORD

```
)
```

## TEMPLATE

```
)
```

## DESCRIPTION

Terminates a comment before EOL reached. Other words can then follow.

## EXAMPLE

```
0 1 . ( top stack item, then second item ) .
```

## 1.17 .

## WORD

```
.
```

## TEMPLATE

```
i .
```

## DESCRIPTION

Takes a parameter off the stack and prints it as an integer value.

---



## EXAMPLE

```
10 .  
15.5 .  
10 20 30 1.2 . . . .
```

## ALSO SEE

F.                    H.                    O.                    B.

**1.18 .s**

## WORD

.S

## TEMPLATE

.S

## DESCRIPTION

Prints the whole contents of the Parameter Stack without removing any values.

**1.19 !**

## WORD

!

## TEMPLATE

iValue aVariable !

## DESCRIPTION

Assigns an integer value iValue to an integer variable aVariable.

Integer and floating point variables must be referenced only with words that are for the variable type in question. For example, an integer variable MUST NOT be referenced using F! or F@, but only with ! and @. This is because the internal representations for similar integer and floating-point values are different.

## EXAMPLE

```
VARIABLE MyVar ( define an integer variable )
```

## ALSO SEE

@ VARIABLE

## 1.20 &

WORD

&

TEMPLATE

& Word aWordAddr

DESCRIPTION

Retrieves the address of the specified word and places it on the stack.  
The word can then be stored in a variable and executed by EXECUTE.

EXAMPLE

```
: MyWord
  "Hello!" PUTS
;

& MyWord EXECUTE
```

ALSO SEE

?& EXECUTE

## 1.21 +

WORD

+

TEMPLATE

i2 i1 + iResult

DESCRIPTION

Takes two integers off the stack, adds them, and puts the sum on the stack.

EXAMPLE

```
VARIABLE MyVar

10 20 + MyVar ! ( MyVar = 10 + 20 )
```

## 1.22 -

WORD

-

TEMPLATE

```
i2 i1 - iResult
```

DESCRIPTION

Takes two integers off the stack, subtracts the stack top value from the second one, and puts the difference on the stack.

EXAMPLE

```
20 10 - .
```

## 1.23 \*

WORD

\*

TEMPLATE

```
i2 i1 * iResult
```

DESCRIPTION

Takes two integers off the stack, multiplies them, and puts the product on the stack.

EXAMPLE

```
3 4 * .
```

## 1.24 divide

WORD

/

TEMPLATE

```
i2 i1 / iResult
```

DESCRIPTION

Takes two integers off the stack, divides the second value on the stack by the stack top item, and puts the integer part of the

quotient on the stack.

EXAMPLE

```
10 5 / .
```

## 1.25 word

WORD

:

TEMPLATE

:

DESCRIPTION

Begins a word definition. The ':' is followed by a space and the name, which can be up to 15 characters, of the RPL word to be defined.

The definition ends with a ';'.

EXAMPLE

```
( define RPL word )
: MyFunction
  "this is RPL word" PUTS
;

( call it )
MyFunction
```

ALSO SEE

;

## 1.26 ;

WORD

;

TEMPLATE

;

DESCRIPTION

Ends a word definition.

---

ALSO SEE

:

## 1.27 <

WORD

<

TEMPLATE

```
i2 i1 < lResult
```

DESCRIPTION

Takes two integer values off the stack and compares them. If the second value is less than the first value, < puts TRUE on the stack, otherwise FALSE is put on the stack.

## 1.28 <=

WORD

<=

TEMPLATE

```
i2 i1 <= lResult
```

DESCRIPTION

Takes two integer values off the stack and compares them. If the second value is less than or equal to the first value, <= puts TRUE on the stack, otherwise FALSE is put on the stack.

## 1.29 <>

WORD

<>

TEMPLATE

```
i2 i1 <> lResult
```

DESCRIPTION

Takes two integer values off the stack and compares them. If the second value is not equal to the first value, <> puts TRUE on the stack, otherwise FALSE is put on the stack.

---

## EXAMPLE

```
: MyTest ( i1 i2 )
<>
IF
  "not equal" PUTS
ELSE
  "equal values" PUTS
ENDIF
;

10 20 MyTest ( not equal
5 5 MyTest ( equal values
```

**1.30 =**

## WORD

=

## TEMPLATE

```
i2 i1 = lResult
```

## DESCRIPTION

Takes two integer values off the stack and compares them. If the values are equal, = puts TRUE on the stack, otherwise FALSE is put on the stack.

## EXAMPLE

```
: IsEqual ( i1 i2 )
=
IF
  "Yes" PUTS
ELSE
  "No" PUTS
ENDIF
;

10 20 IsEqual ( no
10 10 IsEqual ( yes
```

**1.31 >**

## WORD

>

## TEMPLATE

```
i2 i1 > lResult
```

#### DESCRIPTION

Takes two integer values off the stack and compares them. If the second value is greater than the first value, > puts TRUE on the stack, otherwise FALSE is put on the stack.

#### EXAMPLE

```
10 20 > . ( 0
10 5 > . ( 1
```

## 1.32 >=

#### WORD

```
>=
```

#### TEMPLATE

```
i2 i1 >= lResult
```

#### DESCRIPTION

Takes two integer values off the stack and compares them. If the second value is greater than or equal to the first value, >= puts TRUE on the stack, otherwise FALSE is put on the stack.

#### EXAMPLE

```
: IsNegative ( val )
  0 >=
  IF
    "yes" PUTS
  ELSE
    "no" PUTS
  ENDIF
;

10 IsNegative ( no
-1 IsNegative ( yes
 0 IsNegative ( no
```

## 1.33 >r

#### WORD

```
>R
```

#### TEMPLATE

---

```
e >R
```

#### DESCRIPTION

Takes the stack top value and stores it on the Return Stack as an integer value.

ALSO SEE

```
R>
```

## 1.34 >rad

#### WORD

```
>RAD
```

#### TEMPLATE

```
fDeg >RAD fRad
```

#### DESCRIPTION

Converts a value given in degrees to radians.

#### EXAMPLE

```
180.0 >RAD F. ( 3.141593
```

## 1.35 ?&

#### WORD

```
?&
```

#### TEMPLATE

```
?& name a
```

#### DESCRIPTION

Retrieves the address of a given word. If the word is not found pushes a 0 onto the stack.

#### EXAMPLE

```
: DO_IF_FOUND ( executes word DOIT
               ( if it has been defined
DOIT ?& ?DUP
IF
```



```
        EXECUTE
      ENDIF
    ;

ALSO SEE

&          EXECUTE
```

## 1.36 ?dup

WORD

?DUP

TEMPLATE

e ?DUP e e

DESCRIPTION

Duplicates the stack top value if it is not zero.

ALSO SEE

?DUP

## 1.37 ?else

WORD

?ELSE

TEMPLATE

?ELSE

DESCRIPTION

In an interactive conditional structure marks the beginning of the block that is to be executed when the condition fails (i.e the flag tested is FALSE).

EXAMPLE

```
( define constant if not yet defined )
?& MyVar NOT
?IF
  1 CONSTANT MyVar
?ENDIF
```

ALSO SEE

---

?ENDIF      ?IF      ELSE      ENDIF      IF

## 1.38 ?endif

WORD

?ENDIF

TEMPLATE

?ENDIF

DESCRIPTION

Ends an interactive conditional structure, either ?IF..?ENDIF or ?IF..?ELSE..?ENDIF.

ALSO SEE

?ELSE      ?IF      ELSE      ENDIF      IF

## 1.39 ?if

WORD

?IF

TEMPLATE

1 ?IF

DESCRIPTION

Begins an interactive conditional structure, either ?IF..?ENDIF or ?IF..?ELSE..?ENDIF.

If the flag is TRUE then the words entered after ?IF will be executed immediately until either ?ELSE or ?ENDIF is encountered.

If the flag is FALSE, the words between ?IF and ?ELSE/?ENDIF are ignored. Execution then resumes after ?ELSE/?ENDIF.

The interactive conditional structure remains active until ?ENDIF is encountered.

These interactive conditional structures may be nested.

NOTE

Interactive conditional structures can be used to control the execution of parts of an RPL file as it is loaded.

EXAMPLE

```
( check if the word VADD is already defined )
?& VADD
?IF
  "vectors.rpl already installed" PUTS
?ELSE
  "vectors.rpl" LOAD
?ENDIF
```

ALSO SEE

?ELSE            ?ENDIF            ELSE            ENDIF            IF

## 1.40 @

WORD

@

TEMPLATE

aInteger @ iValue

DESCRIPTION

Fetches the value of an integer variable and puts the value on the stack top. The address of the integer variable must be on the stack top before calling this word.

EXAMPLE

```
( MyVar = MyVar + 1 )
MyVar @ 1 + MyVar @ !
```

ALSO SEE

!            VARIABLE

## 1.41 again

WORD

AGAIN

TEMPLATE

AGAIN

DESCRIPTION

Marks the end of an BEGIN..AGAIN loop. The BEGIN..AGAIN loop

---

executes forever unless a QUIT or EXIT is executed.

#### NOTE

Can only be used inside a word definition.

#### EXAMPLE

```
: MyLoop
  BEGIN
    "YES|NO" "Cancel Loop ?" GET_KEY
    IF
      EXIT
    ENDIF
  AGAIN
;
```

#### ALSO SEE

BEGIN            EXIT            QUIT

## 1.42 and

#### WORD

AND

#### TEMPLATE

12 11 AND 1

#### DESCRIPTION

Takes two boolean flags off the stack and, if they both are TRUE, puts TRUE on the stack, otherwise puts FALSE on the stack. The value is TRUE if it is not zero.

#### EXAMPLE

```
1 1 AND . ( 1
0 1 AND . ( 0
0 0 AND . ( 0
10 20 AND . ( 1
```

#### ALSO SEE

IF            OR            XOR

## 1.43 acos

#### WORD

---

ACOS

TEMPLATE

fRad ACOS fAng

RETURNS

fAng - value from 0 to PI

DESCRIPTION

Arccosine function.

EXAMPLE

0.5 ACOS F.

ALSO SEE

ASIN            ATAN            COS

## 1.44 asin

WORD

ASIN

TEMPLATE

fRad ASIN fAng

RETURNS

fAng - value from  $-\pi/2$  to  $\pi/2$

DESCRIPTION

Arcsine function.

EXAMPLE

0.5 ASIN F.

ALSO SEE

ACOS            ATAN            COS

## 1.45 atan

WORD

---



b aByte B!

#### DESCRIPTION

Stores a value in an byte variable. Takes the address of the variable and the value to be stored off the stack.

#### NOTE

There are no byte variables in RPL. This word is needed only when accessing 8 bit data (like R,G,B) from data structures.

#### ALSO SEE

W!                      W@                      B@

## 1.48 b@

#### WORD

B@

#### TEMPLATE

aByte B@ iValue

#### DESCRIPTION

Fetches the value of a byte. The address of the byte must be on the stack top before calling this word.

#### NOTE

See note for B!.

#### ALSO SEE

W!                      W@                      B!

## 1.49 band

#### WORD

BAND

#### TEMPLATE

i2 i1 BAND i

#### DESCRIPTION

Makes a binary AND operation on the two operands and puts the result

---

on the stack.

In a binary AND operation, the result has only those bits set whose corresponding bits are set in both the operands.

#### EXAMPLE

```
1 1 BAND ( 1
2 1 BAND ( 0
2 3 BAND ( 3
```

#### ALSO SEE

BNOT            BOR            BXOR

## 1.50 begin

#### WORD

BEGIN

#### TEMPLATE

BEGIN

#### DESCRIPTION

BEGIN marks the beginning of an indefinite loop. An indefinite loop can be any of the following:

```
BEGIN..UNTIL
BEGIN..AGAIN
BEGIN..WHILE..REPEAT
```

In the BEGIN..UNTIL loop a flag is tested at the end of each repetition of the loop. If the flag is TRUE, the loop terminates. Otherwise the loop repeats. Since the test is made at the end of the loop, the loop will always be executed at least once.

The BEGIN..AGAIN loop executes forever unless a QUIT or EXIT is executed.

In the BEGIN..WHILE..REPEAT loop the words between BEGIN and WHILE are first executed, and then a flag is tested. If the flag is TRUE, the words between WHILE and REPEAT are executed and the loop starts over. If the flag is FALSE, then execution skips to the word that comes after REPEAT.

Indefinite loops can be nested.

#### NOTE

Can only be used inside a word definition.

#### EXAMPLE

---



```

:BeginUntil
  BEGIN
    "Yes|No" "Cancel Loop ?" GET_KEY
  UNTIL
;

: BeginAgain
  BEGIN
    "Yes|No" "Cancel Loop ?" GET_KEY
    IF
      QUIT
    ENDIF
  AGAIN
;

: BegWhlRpt
  BEGIN
    "Yes|No" "Continue ?" GET_KEY
  WHILE
    "hello" PUTS
  REPEAT
;

ALSO SEE

  UNTIL      AGAIN      WHILE      REPEAT      QUIT      EXIT

```

## 1.51 bnot

WORD

BNOT

TEMPLATE

```
i1 BNOT i
```

DESCRIPTION

Inverts the bits of a integer value on the stack. Any bits in the integer value that are set (1) are reset, and any bits that are reset (0) are set.

ALSO SEE

BAND          BOR          BXOR

## 1.52 bor

WORD

---

BOR

TEMPLATE

```
i2 i1 BOR i
```

DESCRIPTION

Makes a binary OR operation on the two operands and puts the result on the stack.

In a binary OR operation the result has all those bits set that have a corresponding bit set in either or both of the operands.

ALSO SEE

BNOT            BAND            BXOR

## 1.53 bxor

WORD

BXOR

TEMPLATE

```
i2 i1 BXOR i
```

DESCRIPTION

Makes a binary XOR (exclusive or) operation on the two operands and puts the result on the stack.

In a binary XOR operation the result has those bits set that have a corresponding bit set in only one of the operands.

ALSO SEE

BNOT            BAND            BOR

## 1.54 cat

WORD

CAT

TEMPLATE

```
s2 s1 CAT
```

DESCRIPTION

---

Concatenates two strings. Takes two pointers s1 and s2, and joins the string pointed to by s2 to the end of string pointed to by s1.

The result is stored in s1.

#### EXAMPLE

```
30 STRING NAME

"Mary" NAME CPY
" Smith" NAME CAT
NAME PUTS
```

#### ALSO SEE

CPY                  PUTS                  SPRINTF                  STRING

## 1.55 constant

#### WORD

CONSTANT

#### TEMPLATE

i CONSTANT name

#### DESCRIPTION

Defines a named integer constant and initializes it to the value popped off the stack.

When the constant is later referenced by entering it's name, the value of the constant is pushed onto the stack.

#### NOTE

This word is usually used outside word definitions.

#### ALSO SEE

VARIABLE                  FVARIABLE                  FCONSTANT

## 1.56 cos

#### WORD

COS

#### TEMPLATE

f1 COS f

---

## DESCRIPTION

Calculates the cosine of the stack top item. The operand must be in radians.

## EXAMPLE

```
3.16 SIN F.
```

## ALSO SEE

SIN

## 1.57 cpy

## WORD

CPY

## TEMPLATE

```
s2 s1 CPY
```

## DESCRIPTION

Copies a string. Takes two pointers s1 and s2, and copies the string pointed to by s2 to the string pointed to by s1.

## EXAMPLE

```
100 STRING sBuf
"Hello world" sBuf CPY
sBuf PUTS
```

## ALSO SEE

CAT                PUTS                SPRINTF                STRING

## 1.58 depth

## WORD

DEPTH

## TEMPLATE

```
DEPTH i
```

## DESCRIPTION

Puts the count of the stack items onto the stack.

---

ALSO SEE

RDEPTH

## 1.59 do

WORD

DO

TEMPLATE

i2 i1 DO

DESCRIPTION

Begins a definite loop, either DO..LOOP or DO..+LOOP. A definite loop executes the words inside the loop a specified number of times. The beginning (i1) and ending (i2) values for the loop variable are put on the stack before the word DO. The loop variable can be referenced using the words I, J or K depending on the nesting level.

NOTE

Can only be used inside a word definition.

EXAMPLE

```
: 5Times
  5 0 DO
    I .
  LOOP
;
```

ALSO SEE

LOOP            +LOOP            I            J            K

## 1.60 drop

WORD

DROP

TEMPLATE

e DROP

DESCRIPTION

Removes the top value from the stack.

---

## 1.61 dup

WORD

DUP

TEMPLATE

e DUP e e

DESCRIPTION

Duplicates the stack top value retaining its type (i.e. integer or floating-point).

ALSO SEE

?DUP

## 1.62 else

WORD

ELSE

TEMPLATE

ELSE

DESCRIPTION

In a conditional structure, marks the beginning of the block that is to be executed when the condition fails (i.e the flag tested is FALSE).

NOTE

Can only be used inside a word definition.

EXAMPLE

```
: ABS  ( i ABS i )
  DUP
  0 >=
  IF
  ELSE
    -1 *
  ENDIF
;
```

ALSO SEE

IF                   ENDIF

---

## 1.63 emit

WORD

EMIT

TEMPLATE

i EMIT

DESCRIPTION

Prints the ASCII character corresponding to the first byte of the top stack value.

EXAMPLE

```
: CR
  13 EMIT
  10 EMIT
; ( carriage return and line feed )

"Hello" PUTS CR "World" PUTS CR
```

## 1.64 endif

WORD

ENDIF

TEMPLATE

ENDIF

DESCRIPTION

Ends a conditional structure, either IF..ENDIF or IF..ELSE..ENDIF.

NOTE

Can only be used inside a word definition.

ALSO SEE

IF                    ELSE

## 1.65 error

WORD

ERROR

TEMPLATE

---

sErrorMsg ERROR

#### PARAMETERS

sErrorMsg - error message to be printed

#### DESCRIPTION

Terminates the program as if an error had occurred and prints out the given error message.

RPL programs can use this word for terminating code execution in situations which are not interpreted as errors by RPL. For example, if a procedural texture handler realizes that it cannot generate required color information for the renderer, it can call this word in order to cancel rendering.

If the sErrorMsg is 0, no error message is printed.

#### EXAMPLE

```
RX_RC @ IF
  "Return value is not zero" ERROR
ENDIF
```

## 1.66 eval

#### WORD

EVAL

#### TEMPLATE

sExpr EVAL f

#### DESCRIPTION

Evaluates an algebraic expression contained in the string pointed to by sExpr, and pushes the result on the stack. If the string contains multiple expressions then the last one evaluated determines the return value.

The following operators are supported by the EVAL word:

,	- separator for multiple expressions
(	- parentheses for controlling evaluation precedence
)	-
+	- add
-	- subtract
*	- multiply
/	- divide
-	- negate
^	- power
%	- modulo



```

+=      - x+=0.1 is same as x = x + 0.1
-=      -
*=      -
/=      -
&&      - logical AND
||      - logical OR
=       - assignment
==      - comparison is equal
>       - comparison greater than
<       -
>=      - comparison greater or equal
<=      -
!=      - comparison not equal

=if()   - =if(a>b,t,f) if a > b then return t, otherwise f

abs()   - absolute value, for example abs(-10) produces 10
ceil()  - returns smallest integer value which is >= x
data()  - interface to object and material data structures.
exp()   - exp(x) returns e^x
floor() - greatest integer value which is <= x
lg()    - log to base 10
ln()    - natural log
max()   - returns the greatest from the list
min()   - min(a,b,c,1,2) returns the smallest from the list a,b,c etc.
sgn()   - sign, sgn(x) is -1 if x < 0, 0 if x = 0 and 1 if x > 0
sqrt()  - square root
sum()   - sum(1,2,3) = 6

cos()   - trigonometric functions
sin()   -
tan()   -

acos()  - inverse trigonometric functions
asin()  -
atan()  -

```

#### NOTE

```

ceil() & floor()
=====

```

#### examples:

```

    ceil(1.4) returns 2, floor(1.4) returns 1 and ceil(-1.4) returns -1

```

```

data()
=====

```

By using the data() operator of EVAL, the properties of objects and materials and some global system values can be accessed.

THE APPROPRIATE DATA STRUCTURE MUST BE LOCKED USING O\_LOCK OR MAT\_LOCK BEFORE USING THE data() OPERATOR TO ACCESS THEM.

## Objects

-----

The syntax for accessing object attributes through EVAL is:

```
"data(/path/name->prop)" EVAL f
```

EVAL again returns the address of the property if used without any expression assignment.

Where:

'/path/name' - Objects name including full absolute path. If the first character in data() is a slash, the name is assumed to refer to an object. Otherwise it is considered to be a material reference.

'prop' - One of the following:

R, G, B - Red, Green, or Blue color signal (0 .. 255)

A - Alpha information

reg - A register color for wire frame representation

ptrn - A pattern for the line. Value must be between 0 and 65535 and each bit in pattern represents one pixel in the line to be drawn. If the first bit is set, then the first pixel in the line will be set. Thus a value of 65536 produces a solid line.

## Materials

-----

The syntax is as follows:

```
"data(name->prop)" EVAL f
```

If used without any expression assignment then EVAL will return the address of the material property.

Where:

'name' - The name of the material in the Material Library to be accessed.

'prop' - Any of the material properties listed below:

name - name (string field)

imag - image (string field)

splu - spline mapping u/v/width/height (0.0 .. 1.0)

splv

splw

splh

```
frex - texture frequency x (positive integer)
frey - texture frequency y (positive integer)

tr.r - transparency color R, G, B (0 .. 255)
tr.g
tr.b

spec - specularity
sbri - specular brightness
bril - brilliance
tran - transparency
turb - turbidity
tbsa - turbidity saturation
refr - refraction
roug - roughness
dith - dithering scale
bump - bump height (-100 .. 100)
effc - effect

maph - mapping handler
mapa - constants for the mapping handler (floating-point)
mapb

scoh - scope handler
scoa
scob

bmph - bump handler
bmpa
bmpb

colh - color handler
cola
colb

indh - index handler
inda
indb
```

Values for all properties can vary between 0 .. 100, unless otherwise specified above.

Handlers can contain values 0, 1, 2 etc. depending on the number of choices in the corresponding cycle gadget in the Material Editor window (Default = 0, etc.).

```
attr()
=====
```

RPL/EVAL word now supports a new operator `attr()`. The operator can be used for accessing internal data structures of Real and is intended to replace the old `data()` operator.

The `attr()` operator takes one parameter which describes the attribute to be accessed. The syntax of the parameter is as follows:

Database:object->attribute

where Database can be one of the following:

O: Objects  
 M: Materials  
 W: Windows  
 A: Animation Settings

Any object can be accessed by defining full path name for it. For example:

```
attr(O:/level/ellipse->r) = 0;
```

Materials can be accessed accordingly except that the object path is replaced by the material name. For example:

```
attr(M:wood->bril) = 10;
```

Just like materials, windows can be accessed through their names. Currently only View window attributes can be accessed. If the user refers to a window which is not a View window, error message will be given. For example, animating the brightness attribute could be done as follows:

```
attr(W:View->brightness) = ...
```

The O: and M: database attributes are listed in the manual RPL/EVAL/data().

The possible Window attributes are listed below:

```
type      - projection type: parallel/perspective
flags     - View window specific flags
mouse.r   - current abs.space mouse position
mouse.s
mouse.t
distance  - distance between aimpoint and viewpoint
scale     - current scale
aspectratio - aspect ratio
mesh_subdiv - See View/Drawing Settings for these two
curve_subdiv
gridsize.x - grid size
gridsize.y
gridsize.z
gridpos.x - grid position
gridpos.y
gridpos.z
grid.color - register color for grid
grid.patter - 0 ... 65535 value specifying line pattern
backgr_img - background image for rendering
env_map   - environment mapping for rendering
mode      - rendering mode. See RPL/RSPEC
output    - output for rendering RPL/RSPEC
dither    - dithering method, see RPL/RSPEC
ditherscale - dithering scale
aadept    - antialiasing depth
dx        - screen pixels/rendering pixel
dy
lsamples  - for smooth shadows
```

```

recursions - recursion depth
brightness - brightness
ambient.r - ambient color
ambient.g
ambient.b
bgr_from.r - background color
bgr_from.g
bgr_from.b
bgr_to.r - background gradient color
bgr_to.g
bgr_to.b
env_from.r - environment color
env_from.g
env_from.b
env_to.r - environment gradient color
env_to.g
env_to.b
overlight - overlight
subdiv - B-Spline quality (from -2 to 2, 0=default)
dof - depth of field
dofw - depth of field
msamples - material samples
filename - output file name

```

Animation settings can be accessed as follows:

```
"attr(A:->attribute)" EVAL
```

where 'attribute' is one of the following:

```

time - current time (0.0 ... 1.0)
frames - total number of frames in the animation
scrfilename - name of the file name for screen animations
flags - animation system specific flags
format - for formatting actual file name (filename+frame)
screen - screen name to be saved
framecmd - frame cmd
currentfrm - current frame (0 ... frames - 1)
endtime - end time for animation (0.0 ... 1.0)
samples - samples for motion blur/particle animations
seconds - seconds for particle animations (0.0 ... )

```

#### Global Data

```
-----
```

Global system data can also be accessed through EVAL. The following variables are defined for this purpose:

```

T - Current Time (floating-point value)
Res - Animation Resolution for animation (integer)
Frm - Current Animation Frame (integer)

```

#### EXAMPLE

```
FVARIABLE X
```

```
0.02 X F!  
"0.5 * sin(2 * X) + cos(X)" EVAL  
F.  
  
"Real:Textures/wood1"  
"data(wood->image)" EVAL CPY  
  
"data(/Root/rectangle->R)=255" EVAL  
  
( print out the number of frames )  
"Res" EVAL F.  
  
( reset the frame counter and discard )  
"Frm=0" EVAL DROP ( the return value )
```

## 1.67 execute

WORD

EXECUTE

TEMPLATE

aCFA EXECUTE

DESCRIPTION

Executes a word given it's CFA .

ALSO SEE

&                      ?&

## 1.68 exit

WORD

EXIT

TEMPLATE

EXIT

DESCRIPTION

Terminates the execution of the current word, and returns control to the word that executed the current word.

NOTE

---

Can only be used inside a word definition.

#### EXAMPLE

```
: TEST
  DUP
  5 < IF
    DROP
  EXIT
ENDIF
.
;

: LUP
  10 0 DO
    I TEST
  LOOP
;

```

ALSO SEE

QUIT

## 1.69 exp

WORD

EXP

TEMPLATE

fPar EXP fRet

DESCRIPTION

Raises the natural logarithm base E to the fPar power.

## 1.70 f.

WORD

F.

TEMPLATE

f F.

DESCRIPTION

Takes a floating-point value off the stack and prints it.

---

ALSO SEE

. H. O. B.

## 1.71 f!

WORD

F!

TEMPLATE

f aFloat F!

DESCRIPTION

Stores a value in a floating-point variable. Takes the address of the variable and the value to be stored off the stack.

NOTE

See the note on !

ALSO SEE

F@ FVARIABLE

## 1.72 f+

WORD

F+

TEMPLATE

f2 f1 F+ f

DESCRIPTION

Takes two floating-point values off the stack, adds them, and puts the sum on the stack.

## 1.73 f-

WORD

F-

TEMPLATE

---



```
f2 f1 F- f
```

#### DESCRIPTION

Takes two integers off the stack, subtracts the stack top value from the second one, and puts the difference on the stack.

## 1.74 f\*

#### WORD

```
F*
```

#### TEMPLATE

```
f2 f1 F* f
```

#### DESCRIPTION

Takes two floating-point values off the stack, multiplies them, and puts the product on the stack.

## 1.75 fdivide

#### WORD

```
F/
```

#### TEMPLATE

```
f2 f1 F/ f
```

#### DESCRIPTION

Takes two floating point values off the stack, divides the second value on the stack by the stack top item, and puts the quotient on the stack.

## 1.76 f<

#### WORD

```
F<
```

#### TEMPLATE

```
f2 f1 F< l
```

#### DESCRIPTION

---

Takes two floating-point values off the stack and compares them. If the second value is less than the first value, `F<` puts `TRUE` on the stack, otherwise `FALSE` is put on the stack.

## 1.77 `f<=`

WORD

`F<=`

TEMPLATE

`f2 f1 F<= 1`

DESCRIPTION

Takes two floating-point values off the stack and compares them. If the second value is less than or equal to the first value, `F<=` puts `TRUE` on the stack, otherwise `FALSE` is put on the stack.

## 1.78 `f<>`

WORD

`F<>`

TEMPLATE

`f2 f1 F<> 1`

DESCRIPTION

Takes two floating-point values off the stack and compares them. If the second value is not equal to the first value, `F<>` puts `TRUE` on the stack, otherwise `FALSE` is put on the stack.

## 1.79 `f=`

WORD

`F=`

TEMPLATE

`f2 f1 F= 1`

DESCRIPTION

Takes two floating-point values off the stack and compares them. If

---

the values are equal, F= puts TRUE on the stack, otherwise FALSE is put on the stack.

## 1.80 f>

WORD

F>

TEMPLATE

f2 f1 F> 1

DESCRIPTION

Takes two floating-point values off the stack and compares them. If the second value is greater than the first value, F> puts TRUE on the stack, otherwise FALSE is put on the stack.

## 1.81 f>=

WORD

F>=

TEMPLATE

f2 f1 F>= 1

DESCRIPTION

Takes two floating-point values off the stack and compares them. If the second value is greater than or equal to the first value, F>= puts TRUE on the stack, otherwise FALSE is put on the stack.

## 1.82 f>i

WORD

F>I

TEMPLATE

f F>I i

DESCRIPTION

Takes a floating point value off the stack and pushes a corresponding integer value on the stack.

---

ALSO SEE

I>F

## 1.83 f@

WORD

F@

TEMPLATE

aFloat F@ f

DESCRIPTION

Fetches the value of a floating-point variable and puts the value on the stack top. The address of the floating-point variable must be on the stack top before calling this word.

NOTE

See the note on !

ALSO SEE

F!                    FVARIABLE

## 1.84 fconstant

WORD

FCONSTANT

TEMPLATE

f FCONSTANT name

DESCRIPTION

Defines a named floating-point constant and initializes it to the value popped off the stack.

When the constant is later referenced by entering it's name, the value of the constant is pushed onto the stack.

NOTE

This word is usually used outside word definitions.

ALSO SEE

---

FVARIABLE      VARIABLE      CONSTANT

## 1.85 fmod

WORD

FMOD

TEMPLATE

f2 f1 FMOD f

DESCRIPTION

Takes two floating point values off the stack, divides the second value on the stack by the stack top item, and puts the remainder of the division on the stack.

## 1.86 forget

WORD

FORGET

TEMPLATE

FORGET name

DESCRIPTION

Removes definitions from the vocabulary. All words that have been defined after the given word, as well as the word itself, are removed from the Vocabulary Stack.

## 1.87 fvariable

WORD

FVARIABLE

TEMPLATE

FVARIABLE name

DESCRIPTION

Defines a named floating-point variable. The name of the variable must follow the word FVARIABLE. The RPL interpreter allocates memory from the Vocabulary Stack for storing the variable. The

---

variable is initialized as 0.0.

When the variable is later referenced by entering it's name, the address of the memory location that stores the variable's value is pushed onto the stack.

#### NOTE

This word is usually used outside word definitions.

#### ALSO SEE

F!                      F@                      FCONSTANT              VARIABLE              CONSTANT

## 1.88 h.

#### WORD

H.

#### TEMPLATE

i H.

#### DESCRIPTION

Takes an integer off the stack and prints it as a hexadecimal number.

#### ALSO SEE

.                      O.                      B.

## 1.89 i

#### WORD

I

#### TEMPLATE

I i

#### DESCRIPTION

Inside a definite loop copies the value of the loop variable of the innermost loop onto the stack.

#### NOTE

Can only be used inside a word definition.

---

ALSO SEE

J                      K

## 1.90 i>f

WORD

I>F

TEMPLATE

i I>F f

DESCRIPTION

Takes an integer value off the stack and pushes a corresponding floating-point value onto the stack.

ALSO SEE

F>I

## 1.91 if

WORD

IF

TEMPLATE

1 IF

DESCRIPTION

Begins a conditional structure, either IF..ENDIF or IF..ELSE..ENDIF.

In the former case, if the flag is TRUE then the words between IF and ENDIF are executed, and then the words after ENDIF. If the flag is FALSE, execution skips the words between IF and ENDIF.

In the latter case, if the flag is TRUE then the words between IF and ENDIF are executed and then the words after ENDIF. If the flag is FALSE, the words between ELSE and ENDIF are executed and then the words after ENDIF.

Conditional structures may be nested.

NOTE

Can only be used inside a word definition.

---

## EXAMPLE

```

: PrintSign
  DUP 0 < IF
    DROP
    "-" PUTS
  ELSE
    0 > IF
      "+" PUTS
    ENDIF
  ENDIF
;

-2 PrintSign
3 PrintSign
0 PrintSign

```

## ALSO SEE

```
ELSE      ENDIF
```

**1.92 j**

## WORD

```
J
```

## TEMPLATE

```
J i
```

## DESCRIPTION

Inside a definite loop copies the value of the loop variable of the second innermost loop onto the stack.

## NOTE

Can only be used inside a word definition.

## EXAMPLE

```

( line feed and carriage return )
: CR 10 EMIT 13 EMIT ;

: MULT_TABLE
  11 1 DO
    11 1 DO
      I J * .
    LOOP
  CR
  LOOP
;

```

## ALSO SEE



I                    K

## 1.93 k

WORD

K

TEMPLATE

K i

DESCRIPTION

Inside a definite loop copies the value of the loop variable of the third innermost loop onto the stack.

NOTE

Can only be used inside a word definition.

ALSO SEE

I                    J

## 1.94 leave

WORD

LEAVE

TEMPLATE

LEAVE

DESCRIPTION

Terminates a definite loop prematurely. The word LEAVE causes the definite loop to terminate at the next LOOP or +LOOP.

NOTE

Can only be used inside a word definition.

EXAMPLE

```
( terminates when I squared exceeds 50 )
: LUP
  10 0 DO
    I DUP * DUP
    50 > IF
```

```
        LEAVE
      ELSE
        DROP
      ENDIF
    .
  LOOP
;
ALSO SEE

EXIT          QUIT
```

## 1.95 load

WORD

LOAD

TEMPLATE

sFile LOAD

DESCRIPTION

Loads a file containing RPL code. The effect is the same as if the text had been entered in the RPL window. The address of the string containing the file name is taken off the stack top.

## 1.96 log

WORD

LOG

TEMPLATE

fPar LOG fRet

PARAMETER

fPar - positive parameter value

DESCRIPTION

Takes the base E logarithm. The parameter fPar must be a positive value.

ALSO SEE

LOG10

---

## 1.97 log10

WORD

LOG10

TEMPLATE

fPar LOG10 fRet

PARAMETER

fPar - positive parameter value

DESCRIPTION

Takes the base 10 logarithm. The parameter fPar must be a positive value.

ALSO SEE

LOG

## 1.98 loop

WORD

LOOP

TEMPLATE

LOOP

DESCRIPTION

Ends a definite DO..LOOP loop.

NOTE

Can only be used inside a word definition.

ALSO SEE

DO                   +LOOP           I                   J                   K

## 1.99 +loop

WORD

+LOOP

TEMPLATE

---

i +LOOP

#### DESCRIPTION

Ends a definite DO..+LOOP loop. This word is used when the loop variable must be incremented by values other than 1. +LOOP takes the stack top value and adds it to the loop variable.

If the ending value is greater than the beginning value, then the loop is exited if the loop variable becomes greater than or equal to the ending value.

If the ending value is less than the beginning value, then the DO..+LOOP is exited when the loop variable becomes less than or equal to the ending value.

#### NOTE

Can only be used inside a word definition.

#### EXAMPLE

```
( use negative increment )
: Down DO I . -1 +LOOP ;
0 5 Down
```

#### ALSO SEE

DO                    LOOP                    I                    J                    K

## 1.100 mod

#### WORD

MOD

#### TEMPLATE

i2 i1 MOD i

#### DESCRIPTION

Takes two integers off the stack, divides the second value on the stack by the stack top item, and puts the remainder of the division on the stack.

## 1.101 not

#### WORD

NOT

---

TEMPLATE

11 NOT 12

DESCRIPTION

Inverts a boolean value on the stack. If the flag is TRUE, it is replaced by FALSE and vice versa. Any value that is not equal to zero is regarded as TRUE. A value of zero is regarded as FALSE.

ALSO SEE

IF

## 1.102 o.

WORD

O.

TEMPLATE

i O.

DESCRIPTION

Takes an integer off the stack and prints it as an octal number.

ALSO SEE

.                    H.                    B.

## 1.103 or

WORD

OR

TEMPLATE

12 11 OR 1

DESCRIPTION

Takes two boolean flags off the stack and, if either one (or both) is TRUE, puts TRUE on the stack, otherwise puts FALSE on the stack.

ALSO SEE

IF                    AND                    XOR

---

## 1.104 over

WORD

OVER

TEMPLATE

e2 e1 OVER e2 e1 e2

DESCRIPTION

Copies the second stack item to the stack top.

ALSO SEE

ROT                  ROLL                  PICK                  SWAP

## 1.105 pick

WORD

PICK

TEMPLATE

i PICK e

DESCRIPTION

Gets a copy of a value on the stack. The copied value will be the i'th value BEFORE the operand for PICK was entered.

EXAMPLE

```
10 9 8 7 .S
4 PICK
.S
```

ALSO SEE

ROLL                  ROT                  SWAP                  OVER

## 1.106 pow

WORD

POW

TEMPLATE

fPow fVal POW fRet

---

## DESCRIPTION

Raises fVal to the fPow power.

## EXAMPLE

```
2 3 POW . ( 9
3 4 POW . ( 64
```

## 1.107 puts

## WORD

PUTS

## TEMPLATE

s PUTS

## DESCRIPTION

Prints a (formatted) string pointed to by s.

## EXAMPLE

```
"Hi there!" PUTS
prints
Hi there!
and
2 3 + "high %d" PUTS
prints
high 5
```

## ALSO SEE

CAT            CPY            SPRINTF        STRING

## 1.108 quit

## WORD

QUIT

## TEMPLATE

QUIT

## DESCRIPTION

QUIT terminates execution of the current word, empties all stacks and returns control to the interpreter.

NOTE

Can only be used inside a word definition.

ALSO SEE

EXIT

## 1.109 r0

WORD

R0

TEMPLATE

R0 i

DESCRIPTION

Puts on the stack the starting address of the return stack. This is the address where the first item (usually a return address) on return stack is stored.

ALSO SEE

RDEPTH

## 1.110 r>

WORD

R>

TEMPLATE

R> i

DESCRIPTION

Takes an integer value off the return stack and pushes it onto the Operand Stack.

ALSO SEE

>R

## 1.111 random



WORD

RANDOM

TEMPLATE

RANDOM f

DESCRIPTION

Returns a floating-point value between 0.0 and 1.0 inclusive and pushes it onto the stack.

## 1.112 rdepth

WORD

RDEPTH

TEMPLATE

RDEPTH i

DESCRIPTION

Puts on the stack the count of the items on the return stack.

ALSO SEE

DEPTH

## 1.113 repeat

WORD

REPEAT

TEMPLATE

REPEAT

DESCRIPTION

Marks the end of a BEGIN..WHILE..REPEAT loop.

NOTE

Can only be used inside a word definition.

ALSO SEE

BEGIN            WHILE

---

## 1.114 roll

WORD

ROLL

TEMPLATE

described below

DESCRIPTION

The word ROLL is used to rotate a given number (i) of stack items so that the i'th stack item becomes the stack top item and all the items between the first and the i'th item are moved one position deeper in the stack. The count i is on the stack top before executing ROLL.

EXAMPLE

```
10 9 8 7 .S
3 ROLL
.S
```

ALSO SEE

PICK            ROT            SWAP            OVER

## 1.115 rot

WORD

ROT

TEMPLATE

e3 e2 e1 ROT e1 e3 e2

DESCRIPTION

ROT rotates the three topmost stack values so that the third item becomes the top item, and the first and the second item (counting from the stack top) will be the second and the third stack item, respectively.

ALSO SEE

ROLL            SWAP            OVER            PICK

## 1.116 s0

---

WORD

S0

TEMPLATE

S0 i

DESCRIPTION

Puts on the stack the starting address of the operand stack. This is the address where the first item on operand stack is stored.

ALSO SEE

DEPTH

## 1.117 sin

WORD

SIN

TEMPLATE

f1 SIN f2

DESCRIPTION

Calculates the sine of the stack top item. The operand must be in radians.

ALSO SEE

COS

## 1.118 sqrt

WORD

SQRT

TEMPLATE

fVal SQRT fRet

PARAMETER

fVal - Positive value

DESCRIPTION

---

Squareroot of fVal.

#### EXAMPLE

```
9 SQRT . ( 3
16 SQRT . ( 4
```

## 1.119 sprintf

#### WORD

SPRINTF

#### TEMPLATE

```
e .. sFormat sResult SPRINTF
```

#### DESCRIPTION

This formats the operand list `e ..` using the format string specified by `sFormat` and stores the result in the string whose address is `sResult`. The operand list is in NORMAL order unlike for most other RPL word operands.

The format string `s1` controls the output format as follows:

`%` start conversion specification.

Then any of the following:

- left adjust operand in its field.
- `m` digit string specifying minimum field width.
- `.` separator from field width and next digit string.
- `n` digit string specifying maximum number of characters or floating-point precision.

Then a conversion character:

- `c` Operand is taken as a single character.
  - `d` Operand is converted to decimal notation.
  - `e` Operand is taken as floating-point and converted to decimal.
  - `f` Operand is taken a floating-point and output as `[-]mm.nnn` where the number of digits for `nnn` is specified by the precision.
  - `g` Use `%e` or `%f` whichever is shorter.
-

- o Operand is converted to unsigned octal notation.
- s Operand must be a string, and is stored until precision specification or the end of the string is reached.
- u Operand is converted to unsigned decimal notation.
- x Operand is converted to unsigned hexadecimal notation.

#### NOTE

For more information about this word refer to a 'C' Language reference manual.

#### EXAMPLE

```
80 STRING Buffer
"Bill" 31 "Hello %s, I am %d years old" Buffer SPRINTF
Buffer PUTS
```

#### ALSO SEE

CPY                  CAT                  PUTS                  STRING

## 1.120 string

#### WORD

STRING

#### TEMPLATE

i STRING name

#### DESCRIPTION

Allocates memory for a named string variable. The size of the memory to be allocated is popped off the stack.

When the string variable is later referenced by entering it's name, the address of the first character of the string is pushed onto the stack.

#### NOTE

This word is usually used outside word definitions.

#### ALSO SEE

CPY                  CAT                  PUTS                  SPRINTF

---

## 1.121 swap

WORD

SWAP

TEMPLATE

e2 e1 SWAP e1 e2

DESCRIPTION

Changes the order of the two stack top values.

ALSO SEE

ROT                  ROLL                  PICK                  OVER

## 1.122 tan

WORD

TAN

TEMPLATE

f1 TAN f2

DESCRIPTION

Tangent function

ALSO SEE

SIN                  COS

## 1.123 until

WORD

UNTIL

TEMPLATE

1 UNTIL

DESCRIPTION

Marks the end of a BEGIN..UNTIL loop. In the BEGIN..UNTIL loop a flag is tested at the end of each repetition of the loop. If the flag is TRUE, the loop terminates. Otherwise the loop repeats. Since the test is made at the end of the loop, the loop will always

be executed at least once.

#### NOTE

Can only be used inside a word definition.

#### EXAMPLE

```
( scan through a list terminated by zero )
: ListScan
  BEGIN
    DUP
    DUP
    = IF
      .H
    ENDIF
  NOT UNTIL
;

```

#### ALSO SEE

BEGIN

## 1.124 variable

#### WORD

VARIABLE

#### TEMPLATE

VARIABLE name

#### DESCRIPTION

Defines a named integer variable. The name of the variable must follow the word VARIABLE. RPL interpreter allocates memory from the Vocabulary Stack for storing the variable. The variable is initialized as 0.

When the variable is later referenced by entering it's name, the address of the memory location that stores the variable's value is pushed onto the stack.

#### NOTE

This word is usually used outside word definitions.

#### ALSO SEE

!                    @                    CONSTANT            FVARIABLE            FCONSTANT

## 1.125 vlist

---

WORD

VLIST

TEMPLATE

VLIST

DESCRIPTION

Lists the names of all words on Vocabulary Stack.

## 1.126 w!

WORD

W!

TEMPLATE

w aWord W!

DESCRIPTION

Stores a value in an short integer variable. Takes the address of the variable and the value to be stored off the stack.

This word can be used for accessing 16 bit integer fields, such as found from Real 3D's material and object data structures.

NOTE

There are no word variables in RPL. This word is needed only when accessing 16 bit data from external data structures.

ALSO SEE

W@                    B@                    B!

## 1.127 w@

WORD

W@

TEMPLATE

aWord W@ i

DESCRIPTION

---



Fetches the value of a short integer. The address of the integer must be on the stack top before calling this word.

#### NOTE

See note for W! .

#### ALSO SEE

W!            B@            B!

## 1.128 while

#### WORD

WHILE

#### TEMPLATE

WHILE

#### DESCRIPTION

In a BEGIN..WHILE..REPEAT loop a flag is tested at the WHILE word. If the flag is TRUE, the words between WHILE and REPEAT are executed and the loop starts over. If the flag is FALSE, then execution skips to the word that comes after REPEAT.

#### NOTE

Can only be used inside a word definition.

#### EXAMPLE

```
: TILL_TEN    ( i TIL_TEN )
  BEGIN
    DUP 1 +
    10 <= WHILE
    .
  REPEAT
  DROP
;
```

#### ALSO SEE

BEGIN            REPEAT

## 1.129 xor

#### WORD

XOR

---

## TEMPLATE

```
12 11 XOR 1
```

## DESCRIPTION

Takes two boolean flags off the stack and, if one and only one of them is TRUE, puts TRUE on the stack, otherwise puts FALSE on the stack.

## ALSO SEE

IF                    AND                    OR

## 1.130 c\_aimpoint

## WORD

C\_AIMPOINT

## TEMPLATE

```
VPosition  
bR bG bB bA  
sName  
iAttr  
TagList  
C_AIMPOINT aObject
```

## 1.131 c\_attrib

## WORD

C\_ATTRIB

## TEMPLATE

```
iR, iG, iB, iA  
sName  
iAttr  
TagList  
C_ATTRIB aObject
```

## EXAMPLE

```
( create a texture with mapping type 'Default' )  
255 255 255 0 ( RGBA )  
"wood_texture" ( Name )  
1OF_TEXTURE ( Attrib )  
"CEND" ( Tags )  
"wood" "SMAT"
```

C\_ATTRIB DROP

## 1.132 c\_cone

WORD

C\_CONE

TEMPLATE

Vapex  
Va  
Vb  
Vaxis  
Vp  
Vm  
Vn  
[fStAngle fEnAngle]  
iR iB iG iA  
sName  
iAttr  
TagList  
C\_CONE aObject

EXAMPLE

```
( surface )  
1.50 1.50 2.00 ( center )  
0.50 0.00 0.00 ( a )  
0.00 0.50 0.00 ( b )  
0.00 0.00 -1.00 ( c )  
( bounding plane )  
1.50 1.50 1.00 ( p )  
0.00 0.50 0.00 ( m )  
0.50 0.00 0.00 ( n )  
255 255 255 0 ( RGBA )  
"cone" ( name )  
0 ( attr )  
"CEND" ( tags )  
C_CONE DROP
```

## 1.133 c\_coordsys

WORD

C\_COORDSYS

TEMPLATE

Vorigin  
Vx  
Vy

---

```

Vz
bR bG bB bA
sName
iAttr
TagList
C_COORDSYS aObject

```

## 1.134 c\_cube

WORD

```
C_CUBE
```

TEMPLATE

```

Vvertex0
Vvertex1
Vvertex2
Vdvect
bR bG bB bA
sName
iAttr
TagList
C_CUBE aObject

```

EXAMPLE

```

( create a cube animated by )
( a RPL method MethodWord )
2.00    3.01    0.00    ( first vertex )
3.01    3.01    0.00    ( second vertex )
2.00    2.00    0.00    ( third vertex )
0.00    0.00    1.00    ( dvect )
255 255 255 0    ( RGBA )
"cube"      ( name )
0           ( attr )
"CEND"
"RPL" "SMTH"
"MethodWord" "SRPL" ( See MTH_CREATE example )
C_CUBE DROP

```

## 1.135 c\_cutcone

WORD

```
C_CUTCONE
```

TEMPLATE

```

Vcentre
Va
Vb

```

```

Vaxis
Vp1
Vm1
Vn1
Vp2
Vm2
Vn2
[fStAngle fEnAngle]
iR iB iG iA
sName
iAttr
TagList
C_CUTCONE aObject

```

#### EXAMPLE

```

( create a sectored cut-cone )
( surface )
-2.51  -0.50   2.50   ( center )
0.50   0.00   0.00   ( a )
0.00   0.50   0.00   ( b )
0.00   0.00  -1.50   ( axis )
( first bounding plane )
-2.51  -0.50   1.00   ( p1 )
0.00   1.00   0.00   ( m1 )
1.00   0.00   0.00   ( n1 )
( second bounding plane )
-2.51  -0.50   2.00   ( p2 )
0.00   1.00   0.00   ( m2 )
1.00   0.00   0.00   ( n2 )
0.00   4.71   ( angles )
255 255 255 0   ( RGBA )
"cutcone"       ( name )
1OF_SECTOR      ( attr: sector )
"CEND"
C_CUTCONE DROP

```

## 1.136 c\_cutpolymid

#### WORD

```
C_CUTPOLYMID
```

#### TEMPLATE

```

Vvertex0
Vvertex1
Vvertex2
Vvertex3
Vvertex4
Vvertex5
[Vvertexn ...]
iCount
bR bG bB bA
sName

```

---

```
iAttr
TagList
C_CUTPOLYMID aObject
```

#### NOTE

iCount specifies the number of vertices on each polygonal face of the polymid, NOT the total number of vertices. There must be an equal number of vertices for both faces, or the geometry is illegal.

## 1.137 c\_cutpyramid

#### WORD

```
C_CUTPYRAMID
```

#### TEMPLATE

```
Vvertex0
Vvertex1
Vvertex2
Vvertex3
Vvertex4
Vvertex5
bR bG bB bA
sName
iAttr
TagList
C_CUTPYRAMID aObject
```

## 1.138 c\_cylinder

#### WORD

```
C_CYLINDER
```

#### TEMPLATE

```
Vcentre
Va
Vb
Vaxis
Vp1
Vm1
Vn1
Vp2
Vm2
Vn2
[fStAngle fEnAngle]
iR iB iG iA
sName
iAttr
```

---

```
TagList
C_CYLINDER aObject
```

## 1.139 c\_ellipse

WORD

```
C_ELLIPSE
```

TEMPLATE

```
Vcentre
Va
Vb
Vdvect
[fStAngle fEnAngle]
iR iB iG iA
sName
iAttr
TagList
C_ELLIPSE aObject
```

## 1.140 c\_ellipseg

WORD

```
C_ELLIPSEG
```

TEMPLATE

```
Vcentre
Va
Vb
Vaxis
Vp1
Vm1
Vn1
Vp2
Vm2
Vn2
[fStAngle fEnAngle]
iR iB iG iA
sName
iAttr
TagList
C_ELLIPSEG aObject
```

## 1.141 c\_ellipsoid

WORD

C\_ELLIPSOID

TEMPLATE

Vcentre  
Va  
Vb  
Vaxis  
bR bG bB bA  
sName  
iAttr  
TagList  
C\_ELLIPSOID aObject

## 1.142 c\_group

WORD

C\_GROUP

TEMPLATE

iIndx0 [iIndxn ..]  
iCount  
bR bG bB bA  
sName  
iAttr  
TagList  
C\_GROUP aObject

OPERANDS

iIndxn - Indexes of the points of the freeform.

DESCRIPTION

Creates a sub-group primitive referring to the points of a freeform. One tag in must be SOBJ and its value must contain the path and name of the freeform.

EXAMPLE

```
0.0 0.0 0.0
0.0 0.1 0.0
1.0 1.0 0.2
3          ( count )
3          ( type )
0          ( geometry flags )
255 255 255 0 ( RGBA )
"line"      ( name )
1OF_RTINVISIBLE ( attr: )
"CEND"      ( tags )
```



```

C_LINE DROP

( point references )
0 2
2      ( count )
255 255 255 0 ( RGBA )
"line_group"  ( name )
0      ( attr: )
"CEND"      ( tags )
"/Root/line" "SOBJ"
C_GROUP DROP

```

ALSO SEE

C\_LINE                      C\_MESH

## 1.143 c\_hyperbol

WORD

C\_HYPERBOL

TEMPLATE

```

Vcentre
Va
Vb
Vaxis
Vp1
Vm1
Vn1
Vp2
Vm2
Vn2
[fStAngle fEnAngle]
iR iB iG iA
sName
iAttr
TagList
C_HYPERBOL aObject

```

## 1.144 c\_level

WORD

C\_LEVEL

TEMPLATE

```

wBoolean
sName
iAttr

```

---

```
TagList
C_LEVEL aObject
```

#### DESCRIPTION

Creates a level with the attributes and tags supplied. The Boolean Operator type is specified by wBoolean as follows:

```
wOT_AND
wOT_OR
```

#### NOTE

The level does not become the Current Level.

#### EXAMPLE

```
wOT_OR "mylevel" 0 "CEND" C_LEVEL DROP
```

#### ALSO SEE:

```
O_CURRENT
```

## 1.145 c\_line

#### WORD

```
C_LINE
```

#### TEMPLATE

```
Vpt0
Vpt1
[Vptn ..]
iCount
wFreeType
wGeomFlags
bR bG bB bA
sName
iAttr
TagList
C_LINE aObject
```

#### EXAMPLE

```
( points )
1.00   -1.00   0.00
2.00   -1.00   0.00
2.00   -2.00   0.00
1.00   -2.00   0.00
4      ( count )
3      ( type )
wGF_CLOSEU ( geom. flags )
255 255 255 0 ( RGBA )
"mycurve"   ( name )
```

```
0          ( flags )
"CEND"
C_LINE DROP
```

NOTE

For B-spline lines there must be a minimum of four points.

## 1.146 c\_link

WORD

C\_LINK

TEMPLATE

```
sName
iAttr
TagList
C_LINK aObject
```

DESCRIPTION

Create a symbolic link. One of the tags must be "SOBJ" and its value is the path and name for the object the link refers to.

EXAMPLE

```
"link" 0 "CEND" "/Root/rectangle"
"SOBJ" C_LINK
```

## 1.147 c\_mesh

WORD

C\_MESH

TEMPLATE

```
Vpt[0,0]
Vpt[0,1]
[Vpt[u,v] ..]
Vpt[1,0]
Vpt[1,1]
[Vpt[u,v] ..]
iU
iV
wFreeType
wGeomFlags
bR bG bB bA
sName
iAttr
```

```
TagList
C_MESH aObject
```

#### OPERANDS

iU, iV - number of points for each dimension of the mesh.

#### NOTE

For a B-spline mesh there must be at least four points for each dimension. The total number of points for the mesh must be the product of iU and iV.

#### EXAMPLE

```
( line 0 )
-0.50  0.50  0.00
-0.17  0.50  0.00
0.17   0.50  0.00
0.50   0.50  0.00
( line 1 )
-0.50  0.17  0.00
-0.17  0.17  0.00
0.17   0.17  0.00
0.50   0.17  0.00
( line 2 )
-0.50 -0.17  0.00
-0.17 -0.17  0.00
0.17  -0.17  0.00
0.50  -0.17  0.00
( line 3 )
-0.50 -0.50  0.00
-0.17 -0.50  0.00
0.17  -0.50  0.00
0.50  -0.50  0.00
4 ( width )
4 ( height )
3 ( type )
0 ( geom. flags )
255 255 255 0 ( RGBA )
"mesh"      ( name )
0           ( attr: )
"CEND"
C_MESH DROP
```

## 1.148 c\_offset

#### WORD

```
C_OFFSET
```

#### TEMPLATE

```
Vposition
bR bG bB bA
```

```
sName
iAttr
TagList
C_OFFSET aObject
```

#### EXAMPLE

```
( create blue light-point that doesn't cast shadows )
0.0 0.0 0.0 ( position )
35 35 255 0 ( RGBA )
"BlueLight" ( name )
1OF_LIGHTSOURCE 1OF_SHADOWLESS BOR
"CEND"
C_OFFSET DROP
```

## 1.149 c\_polygon

#### WORD

```
C_POLYGON
```

#### TEMPLATE

```
Vvertex0
Vvertex1
Vvertex2
[Vpt ..]
Vdvect
iCount
bR bG bB bA
sName
iAttr
TagList
C_POLYGON aObject
```

## 1.150 c\_polyhedron

#### WORD

```
C_POLYHEDRON
```

#### TEMPLATE

```
Vvertex0
Vvertex1
Vvertex2
[Vvertexn ..]
Vdvect
iCount
bR bG bB bA
sName
iAttr
```

---

```
TagList
C_POLYHEDRON aObject
```

## 1.151 c\_polymid

WORD

```
C_POLYMID
```

TEMPLATE

```
Vvertex0
Vvertex1
Vvertex2
[Vvertexn ..]
Vapex
iCount
bR bG bB bA
sName
iAttr
TagList
C_POLYMID aObject
```

## 1.152 c\_pyramid

WORD

```
C_PYRAMID
```

TEMPLATE

```
Vvertex0
Vvertex1
Vvertex2
Vapex
bR bG bB bA
sName
iAttr
TagList
C_PYRAMID aObject
```

## 1.153 c\_rectangle

WORD

```
C_RECTANGLE
```

TEMPLATE

```
Vvertex0
```

---

```

Vvertex1
Vvertex2
Vdvect
bR bG bB bA
sName
iAttr
TagList
C_RECTANGLE aObject

```

## 1.154 c\_triset

WORD

```
C_TRISSET
```

TEMPLATE

```

Vpt0
Vpt1
Vpt2
[Vptn ..]
iCount
iIndx0 iIndx1 iIndx2
[iIndxn iIndxo iIndxp ..]
iFaceCount
wFreeType
bR bG bB bA
sName
iAttr
TagList
C_TRISSET aObject

```

OPERANDS

```

iIndxn      - Indexes of the points forming the faces
iFaceCount  - Number of index triplets

```

DESCRIPTION

This word is used internally for creating a mesh with a triangular face topology. There are no menu functions for this.

It is possible to create RPL programs to convert the data structures of other 3D graphics programs that use triangular mesh topologies into Real 3D meshes using this word.

NOTE

wFreeType must specify either polygonal or phong type

## 1.155 c\_viewpoint

WORD

C\_VIEWPOINT

TEMPLATE

Vleft  
Vright  
Vdvect  
bR bG bB bA  
sName  
iAttr  
TagList  
C\_VIEWPOINT aObject

OPERANDS

Vleft     - position of left view for stereo vision pair  
Vright    - right view  
Vdvect     - direction of stereo vision pair

DESCRIPTION

Create a viewpoint consisting of a stereo vision pair.

## 1.156 m\_alpha

WORD

M\_ALPHA

TEMPLATE

0 aObject1 [...aObject]  
iA  
iFlags  
M\_ALPHA

OPERANDS

iA - New alpha channel value for objects.

DESCRIPTION

Modifies so called 'alpha channel' attribute of given objects. The value of iA must be between 0 and 255.

EXAMPLE

O\_GETSEL   ( objects )  
100         ( alpha )  
0  
M\_ALPHA



## 1.157 m\_color

WORD

M\_COLOR

TEMPLATE

```
0 aObject1 [...aObject]
iR iB iG iA
iRegister
iFlags
M_COLOR
```

OPERANDS

iR iB iG iA - New color for objects  
iRegister - Register color for wire-frame

DESCRIPTION

Changes the color of given objects.

EXAMPLE

```
0 "/Root/rectangle" O_FIND ( objects )
255 255 255 0          ( RGBA )
2                      ( register )
0
M_COLOR
```

## 1.158 m\_copy

WORD

M\_COPY

TEMPLATE

```
0 aObject1 [aObjectn ...] M_COPY
```

DESCRIPTION

Copies the targets to the Clip Buffer.

EXAMPLE

```
O_GETSEL ( objects )
M_COPY
```

## 1.159 m\_cut

---

WORD

M\_CUT

TEMPLATE

0 aObject1 [aObjectn ..] M\_CUT

DESCRIPTION

Cuts targets from the hierarchy and places them in the Clip Buffer.

EXAMPLE

O\_GETSEL ( objects )  
M\_CUT

## 1.160 m\_delete

WORD

M\_DELETE

TEMPLATE

0 aObject1 [aObjectn ..] M\_DELETE

DESCRIPTION

Deletes targets from hierarchy.

EXAMPLE

O\_GETSEL ( objects )  
M\_DELETE

## 1.161 m\_duplicate

WORD

M\_DUPLICATE

TEMPLATE

0 aObject1 [aObjectn ..]  
aLevel  
iFlags  
M\_DUPLICATE

## DESCRIPTION

Duplicates given objects and inserts them in to the given object aLevel.

## EXAMPLE

```
O_GETSEL    ( objects )
O_GETCURR   ( to )
0           ( flags )
M_DUPLICATE
```

## 1.162 m\_extend

## WORD

M\_EXTEND

## TEMPLATE

```
0
aObject1
[aObjectn ..]
vCentre
vDirect
fCoeff
iFlags
M_EXTEND
```

## DESCRIPTION

The targets are extended about Vcentre along the direction specified by Vdirect by the amount specified by fCoeff.

## EXAMPLE

```
O_GETSEL          ( objects )
0.21    0.22    0  ( center )
0.97    -0.25   0  ( direction )
-0.6                ( coefficient )
0                  ( flags )
M_EXTEND
```

## 1.163 m\_mirror

## WORD

M\_MIRROR

## TEMPLATE

```
0 aObject1 [...aObjectn]
vCentre
```

---

```
iFlags
M_MIRROR
```

#### DESCRIPTION

The targets are mirrored about a plane that passes through Vcentre, and in the direction specified by Vdirect.

#### EXAMPLE

```
O_GETSEL      ( objects )
-0.03  0.78    0  ( center )
-0.58  -0.81   0  ( axis )
0          ( flags )
M_MIRROR
```

## 1.164 m\_move

#### WORD

```
M_MOVE
```

#### TEMPLATE

```
0 aObject1 [...aObjectn]
Vdelta
iFlags
M_MOVE
```

#### OPERANDS

Vdelta - vector defining direction and amount to move targets.

#### EXAMPLE

```
O_GETSEL      ( objects )
0.2 0.22 0    ( delta )
0          ( flags )
M_MOVE
```

## 1.165 m\_movecog

#### WORD

```
M_MOVECOG
```

#### TEMPLATE

```
0 aObject1 [...aObjectn]
vPosition
iFlags
M_MOVECOG
```

---

## OPERANDS

vPosition - new position for targets' COGs.

## EXAMPLE

```
O_GETSEL      ( objects )
-0.23 0.52 0   ( position )
0              ( flags )
M_MOVECOG
```

## 1.166 m\_name

## WORD

M\_NAME

## TEMPLATE

```
0 aObject1 [...aObjectn]
sName
iFlags
M_NAME
```

## DESCRIPTION

Renames ALL targets to the name specified by sName.

## EXAMPLE

```
O_GETSEL      ( objects )
"newname"     ( name )
0              ( flags )
M_NAME
```

## 1.167 m\_paste

## WORD

M\_PASTE

## TEMPLATE

```
aLevel iFlags M_PASTE
```

## DESCRIPTION

Pastes copies of the Clip Buffer into the object pointed by aLevel.

## EXAMPLE

```
"/Root/rect*" O_FINDWLD M_CUT
"/Root/Level" O_FIND 0 M_PASTE
```

## 1.168 m\_rotate

WORD

M\_ROTATE

TEMPLATE

```
0 aObject1 [aObjectn...]
vCenter
vHor
vVert
vNorm
iFlags
M_ROTATE
```

DESCRIPTION

The targets are rotated and scaled to the coordinate system defined by vCenter, vHor, vVert and vNorm parameters. If the lengths of vHor, vNorm and vVert vectors are 1, then the target objects are only rotated. For example, if the length of the vector vHor is 2, the the size of the objects is doubled in the direction defined by vHor.

The syntax of this word reflects the functions Modify/Linear/Rotate, Rot&Ext and Deform.

EXAMPLE

```
O_GETSEL          ( objects )
-0.32   0.4       0   ( center )
0.83    0.56      0   ( hor )
-0.56   0.83      0   ( vert )
0       0         1   ( norm )
0                               ( flags )
M_ROTATE
```

## 1.169 m\_size2d

WORD

M\_SIZE2D

TEMPLATE

```
0 aObject1 [aObjectn ..]
vCenter
vHor
vVert
```

---

```
fHCoeff fVCoef
iFlags
M_SIZE3D
```

#### DESCRIPTION

Changes the size of the targets in two dimensions about the center defined by vCenter.

The directions in which the object is stretched are defined by the parameters vHor and vVert. This word reflects the internal implementation of the Modify/Linear/Size 2D function.

#### EXAMPLE

```
O_GETSEL      ( objects )
-0.68  0.82    0 ( center )
1      0      0 ( hor  )
0      1      0 ( vert )
0.44    0.44   ( hf, vf )
0                          ( flags )
M_SIZE2D
```

## 1.170 m\_shear

#### WORD

```
M_SHEAR
```

#### TEMPLATE

```
0 aObject1 [aObjectn...]
vCenter
vNorm
vDir
fCoeff
iFlags
M_SHEAR
```

#### DESCRIPTION

The targets are sheared in the direction defined by vDir parameter. The longer the distance between a point and vCenter in the direction defined by vNorm, the more it is moved along the vDir axis. The coefficient defines how much the targets are sheared.

#### EXAMPLE

```
O_GETSEL      ( objects )
0.28  0.36    0 ( center )
-0.92 -0.38   0 ( normal )
0.38  -0.92   0 ( dir  )
-0.83          ( coeff )
0              ( flags )
```

M\_SHEAR

## 1.171 m\_size3d

WORD

M\_SIZE3D

TEMPLATE

```
0
aObject1
[aObjectn ...]
Vcentre
fCoeff
iFlags
M_SIZE3D
```

DESCRIPTION

Alter the size of the targets about the centre specified by Vcentre in all dimensions by the amount specified by fCoeff.

EXAMPLE

```
O_GETSEL      ( objects )
-0.67  0.78    0 ( center )
1.4          ( coefficient )
0           ( flags )
M_SIZE3D
```

## 1.172 m\_stretch

WORD

M\_STRETCH

TEMPLATE

```
0
aObject1
[aObjectn ...]
vCentre
vHor
vVert
vNorm
vCoeff
iFlags
M_STRETCH
```

DESCRIPTION

---



Stretches objects in three dimensions. How much the object is stretched is defined independently in all three dimensions by the parameter vCoeff. For example, if the value of vCoeff is 0 1 0, then the size of object is doubled in the direction defined by vVert.

#### EXAMPLE

```
O_GETSEL ( objects )
0      0.3    0      ( center )
1      0      0      ( hor  )
0      1      0      ( vert )
0      0      1      ( norm )
-0.02   -0.23   0    ( coeff )
0 ( flags )
M_STRETCH
```

## 1.173 m\_swap

#### WORD

M\_SWAP

#### TEMPLATE

```
iFlags
M_SWAP
```

#### DESCRIPTION

Swaps the order of selected objects. This function corresponds the function Modify/Structure/Swap.

#### EXAMPLE

```
0 ( flags )
M_SWAP
```

## 1.174 o\_creatag

#### WORD

O\_CREATAG - Create Object Tag

#### TEMPLATE

```
aObject TagList O_CREATAG aTagAddr
```

#### DESCRIPTION

Creates and adds to the given object the given tags.  
The tags are defined as pairs of a Tag Value and a Tag ID, and the

list must be terminated with "CEND". The word returns the address of the last Tag ID created.

#### NOTE

The address of the Tag Field is the address of the Tag ID + 4. If the tag is an integer (Ixxx) then the Tag Field is the Tag Value, otherwise it is the address of the Tag Values.

#### EXAMPLE

```
( create vector tag to the object '/Root/myobj' )
"/Root/myobj" O_FIND
"CEND"
12.0 5.0 0.0 "VMYT"
O_CREATAG
DROP
```

## 1.175 o\_current

#### WORD

O\_CURRENT

#### TEMPLATE

aObject O\_CURRENT aPrevCurr

#### DESCRIPTION

This makes the object specified by aObject the current level. It returns the address of the previous current level unless the object is a primitive with no sub-structure ( e.g. an ellipsoid ).

#### EXAMPLE

```
"/Car/Engine" O_FIND O_CURRENT
```

## 1.176 o\_delete

#### WORD

O\_DELETE

#### TEMPLATE

aObject O\_DELETE

#### DESCRIPTION

Deletes the object whose address is aObject. Note that NULL is a valid address for O\_DELETE, in which case the word does nothing.

---

## EXAMPLE

```
"/Root/myobj" O_FIND O_DELETE
```

## 1.177 o\_deriv

## WORD

O\_DERIV

## TEMPLATE

```
aParam VPSpace O_DERIV VDir
```

## DESCRIPTION

This returns the three floating-point values of the direction evaluated from the parameter aParam with the parameter values specified by VPSpace.

The vector components of VPSpace correspond the r, s and t dimensions.

## NOTE

The operand aParam must point to an evaluable object or an error will follow.

## EXAMPLE

```
( get the address of a valid parameter )
"/Root/line" O_FIND

( specify the Parameter Space coordinates )
0.5 0.0 0.0
O_DERIV
F. F. F      ( print out the direction )
```

## 1.178 o\_eval

## WORD

O\_EVAL

## TEMPLATE

```
aParam VPSpace O_EVAL Vpoint
```

## DESCRIPTION

This returns the three floating-point values of the point evaluated from the parameter aParam with the parameter values specified by VPSpace.

---

The vector components of VPSpace correspond to the r (Time), s and t dimensions.

#### NOTE

The operand aParam must point to a valid parameter or an error will result.

#### EXAMPLE

```
( get the address of a valid parameter )
"/Root/ellipse" O_FIND

( specify the Parameter Space coordinates )
0.5 0.0 0.0
O_EVAL
```

## 1.179 o\_find

#### WORD

O\_FIND

#### TEMPLATE

sName O\_FIND aObject

#### DESCRIPTION

Attempts to find an object by its name. If an object is found, its address is pushed onto the stack. If not found, NULL is returned.

#### EXAMPLE

```
: MyRename ( sNewName sOldName )
  O_FIND DUP
  IF
    boFO_NAME + CPY
  ELSE
    DROP
  ENDIF
;

"myrect" "/Root/Level/rectangle" MyRename
```

## 1.180 o\_findtag

#### WORD

O\_FINDTAG

---

## TEMPLATE

```
aObject sTagID O_FNDTAG aTagAddr
```

## DESCRIPTION

Attempts to find the tag specified by sTagID from the object specified by the address aObject. It returns the address of the Tag ID if the tag is found.

## NOTE

See O\_CRETAG for notes about Tag ID, Tag Field and Tag Values.

## EXAMPLE

```
: TagTest
  "/Root/Level" O_FIND
  "FMAS" O_FNDTAG
  IF
    "Yes" PUTS
  ELSE
    "FMAS not found" PUTS
  ENDIF
;
```

## 1.181 o\_findwild

## WORD

```
O_FINDWILD
```

## TEMPLATE

```
sWild O_FINDWILD 0 aObject1 [aObjectn ..]
```

## DESCRIPTION

Attempts to find objects whose name matches the given wild-card, and pushes the address for each match onto the stack. Zero is used for terminating this address list.

The wild-card characters available are:

- \* - any number of any characters
- ? - any single character
- . - current level
- .. - parent level
- / - level separator

## NOTE

The list of object addresses can be used directly by the M\_words.

---

EXAMPLE

```
( move all objects at the current level )
( whose name ends with 'le' )
"*le" O_FINDWILD
0.1 0 0 0 M_MOVE
REFRESH
```

## 1.182 o\_getcur

WORD

O\_GETCUR

TEMPLATE

O\_GETCUR aCurrent

DESCRIPTION

Returns the address of the current level.

EXAMPLE

```
( pop up one level )

O_GETCUR O_GETPAR O_CURRENT
```

## 1.183 o\_getnext

WORD

O\_GETNEXT

TEMPLATE

aObject O\_GETNEXT aNextObj

DESCRIPTION

Returns the address of the next object in the hierarchy at the same level as aObject, unless the object was the last in which case NULL is returned.

EXAMPLE

```
: PrintLevel ( aLevel )
  O_GETSUB
  BEGIN
    DUP
  WHILE
    DUP boFO_NAME + PUTS
```

```
        O_GETNEXT
    REPEAT
        DROP
;

( print objects inside the current level )
O_GETCUR PrintLevel
```

## 1.184 o\_getpar

WORD

O\_GETPAR

TEMPLATE

aObject O\_GETPAR aParent

DESCRIPTION

This returns the address of the of the parent of the object pointed by aObject unless there is no parent i.e. aObject is the Root level.

NOTE

This works only if the object is linked to the object data structure.

## 1.185 o\_getprev

WORD

O\_GETPREV

TEMPLATE

aObject O\_GETPREV aPrevObj

DESCRIPTION

This takes a pointer to an object as its operand, and returns a pointer to the previous object at the same hierarchy level. If there is no previous object, NULL is returned.

NOTE

Due to the internal implementation, it is faster to fetch the address of the next object than the previous one. This word works only if the object is linked to the object data structure.

---

## 1.186 o\_getsel

WORD

O\_GETSEL

TEMPLATE

O\_GETSEL 0 aObject1 [aObjectn ..]

DESCRIPTION

Returns a list of object addresses to the selected objects.  
The list is terminated with zero.

EXAMPLE

```
( move selected objects )
```

```
O_GETSEL  
0.1 1.0 0 0 M_MOVE  
REFRESH
```

## 1.187 o\_getsub

WORD

O\_GETSUB

TEMPLATE

aObject O\_GETSUB aFirstSub

DESCRIPTION

This returns the address of the first sub-object of aObject unless  
it contains no sub-objects, in which case 0 is returned.

## 1.188 o\_lock

WORD

O\_LOCK

TEMPLATE

iAccess O\_LOCK

DESCRIPTION

Locks or unlocks the object data structure (hierarchy) according to  
the value of iAccess as specified below:

---



LOCK\_REMOVE - Unlocks the object data

LOCK\_EXCLUSIVE - Tries to lock the data exclusively so that no other task can access it. If the data is already locked, the task in question goes to sleep.

LOCK\_SHARED - Tries to get shared access to the data structure. Several tasks can access the data structure if they all use shared access.

For more details about locking and task access See: "Amiga ROM Kernel Reference Manual: Exec"

#### EXAMPLE

```
LOCK_EXCLUSIVE O_LOCK ( lock object data
"/Root/re*" O_FIND O_DELETE ( delete object
LOCK_REMOVE O_LOCK      ( unlock )
```

ALSO SEE

MAT\_LOCK

## 1.189 o\_prop

WORD

O\_PROP

TEMPLATE

```
aObject lProperty O_PROP ....
```

PARAMETERS

```
aObject - pointer to object
lProperty - flags defining what properties the word should fetch:
    iOP_COG - Center of gravity
    iOP_SIZE - Size of the object
    iOP_DIR - direction
    iOP_MASS - the mass of the object.
```

RETURNS

The number of return values depends on the lProperty flags as follows:

Property	Return value
iOP_COG	A vector
iOP_DIR	Three vectors (coordinate system ) defining the object space
iOP_SIZE	A float value defining the radius of the bounding sphere
iOP_MASS	A float value defining the mass of the object

Parameters are pushed on the stack in this order.

---

## DESCRIPTION

Takes the object and property flags and returns corresponding properties of the object on the stack.

## EXAMPLE

```
( Print out the size of the current level
O_GETCURR iOP_SIZE O_PROP F.

( Print out the size and the mass
( of the current level
O_GETCURR iOP_MASS iOP_SIZE
BOR O_PROP F. F.

( word printing the distance between
( given objects
: PrintDist ( aObj1 aObj2 )
  iOP_COG O_PROP ( COG of the 1st object
  4 ROLL
  iOP_COG O_PROP ( COG of the 2nd object
  VSUB          ( subtract COGs
  VLEN          ( length of the vector
  F.
;
```

**1.190 o\_scan**

## WORD

O\_SCAN

## TEMPLATE

aObject aCFA O\_SCAN e

## DESCRIPTION

Object scan and execute. This parses the hierarchical structure of the object pointed to by aObject and executes the RPL word pointed by aCFA for each sub-object.

The RPL word executed receives the address of the current object as a parameter on the stack and must return a non-zero value to continue the scan or zero to stop.

O\_SCAN returns the value from the executed word.

## EXAMPLE

```
: PrintName
  4 + PUTS ( print the name )
  1 ( return 1 to continue scanning )
;
```

```
"/Root" O_FIND & PrintName O_SCAN DROP
```

## 1.191 o\_select

WORD

O\_SELECT

TEMPLATE

```
0 aObject1 [aObjectn ..] O_SELECT
```

DESCRIPTION

Objects whose address is supplied as an operand become selected.

NOTE

This word is useful for creating macros which select targets for menu function modifications.

EXAMPLE

```
( find and select all rectangles )  
( at the current level )  
"rectangle*" O_FINDWILD  
O_SELECT  
REFRESH
```

## 1.192 aspec

WORD

ASPEC

TEMPLATE

```
iFrameResol  
fStartTime  
sFileName  
iFlags  
sFormatString  
sScrName  
sFrameComm  
iCurrFrame  
iSamples  
fSeconds  
ASPEC
```

DESCRIPTION

---

Animation settings. The parameters of this word correspond the contents of the Animation window.

iFlags parameter can contain the following bits:

iAF\_SAVE - save rendered frames  
iAF\_WIRE - preview  
iAF\_GOTO - go directly to given EndTime

#### EXAMPLE

```
40      ( frame resolution )
0       ( current time )
"ram:test" ( filename )
iAF_SAVE  ( flags )
"%s%d"    ( format )
"Real.1"  ( screen name )
""        ( frame command )
0         ( current frame )
0         ( samples )
1.0       ( seconds )
ASPEC

0 0 1.0 PLAY ( play the animation )
```

#### ALSO SEE

PLAY

## 1.193 play

#### WORD

PLAY

#### TEMPLATE

aObject aMethod fEndTime PLAY

#### PARAMETERS

aObject, aMethod - Set these to NULL  
fEndTime - time value between 0 ... 1

#### DESCRIPTION

Plays the animation to the given time using the current animation settings.

If the given time is less than the current time, the animation is played backwards.

#### NOTE

The first two parameters MUST be set to 0 for future compatibility.

---

## EXAMPLE

```

: Forwards
  0 0 1.0 PLAY
;

: Backwards
  0 0 0.0 PLAY
;

: DoTwice
  Forwards
  Backwards
;

DoTwice

```

**1.194 mth\_create**

## WORD

MTH\_CREATE

## TEMPLATE

aWord sName MTH\_CREATE aMthAddr

## DESCRIPTION

Creates a custom method to Real 3D's Method list. The new method can then be assigned to objects as if it was one of the built-in methods.

The aWord operand contains the address of the RPL word to be used by the method and sName specifies the name of the custom method as it will appear on the list. This can be up to 15 characters long; if a longer string is supplied, it will be truncated.

The word returns the address of the method data.

When an animation is played, this word is called with the following syntax:

```
aNewTime aMthTime aMthObj aParentObj XXXXXXXX
```

where

aNewTime - the address of the vector defining the new time.  
 aMthTime - the address of the vector describing the current method time. If aNewTime is different than aMthTime, the method should 'do something'.  
 aMthObj - the address of the object to which the method procedure was associated.  
 aParObj - the address of the parent object of aMthObj.

Also the following variables are defined during the animation:

```
o1 - method object
o2 - parent object
t, u, v - new time
fx, fy, fz - method's current time
dt - time interval
```

#### EXAMPLE

```
: ColorProc
  0 o2 @ ( address of object to be modified
  t 255 *      ( R
  RANDOM 255 * ( G
  t 6.28 * SIN 127 * 128 + ( B
  0      ( A
  2      ( register color
  0      ( iFlags
  M_COLOR
;

& ColorProc "Color" MTH_CREATE DROP
```

## 1.195 mth\_delete

WORD

MTH\_DELETE

TEMPLATE

aMthAddr MTH\_DELETE

DESCRIPTION

Deletes the custom method specified by aMthAddr from the Method list.

## 1.196 mth\_find

WORK

MTH\_FIND

TEMPLATE

sName MTH\_FIND aMthAddr

DESCRIPTION

Returns the address of the method given its name.

## 1.197 fil\_load

WORD

FIL\_LOAD

TEMPLATE

sFile iSections iReplace FIL\_LOAD

DESCRIPTION

Loads the data sections specified by the bits of iSections from a Real 3D Binary Format IFF file defined by the name sFile. The variable iReplace defines which sections replace the existing ones and which sections are inserted.

The bits of iSections and iReplace select the Data Sections (Real 3D IFF HUNKS) in the following way:

HUNK	DESCRIPTION
RSCR	Screens
RWIN	Windows
RINF	Measuring System
RSTT	Global Settings
RGRI	Grids
RREN	Render Settings
RANI	Animation Settings
RATT	Default Primitive Attributes
ROBJ	Objects
RMTR	Materials
RCOL	Named Colors
RRPL	RPL Text

NOTE

The Version Hunk RVRS is always loaded.

EXAMPLE

```
"io.rpl" LOAD

( insert all sections found )
"MyProject" lIO_RALL 0 FIL_LOAD

( insert all except objects )
"MyProject" lIO_RALL lIO_ROBJ FIL_LOAD
```

## 1.198 fil\_save

WORD

FIL\_SAVE

---

## TEMPLATE

```
sFile iSections iFlags FIL_SAVE
```

## DESCRIPTION

Saves the data sections specified by the bits of iSections to the file specified in sFile using Real 3D binary format.

## NOTE

See FIL\_LOAD for the data sections bits and notes about iFlags. FIL\_SAVE always saves the Version Hunk.

## 1.199 mat\_create

## WORD

```
MAT_CREATE
```

## TEMPLATE

```
sName
wSpecularity
wSpecBright
wBrilliance
wTransparency
wTurbidity
wRefraction
wCurIndex
wEffect
wRESERVED
wRoughness
wFlags
wTurbSatur
wMapMethods
wRESERVED
wFreqX
wFreqY
fSplineU
fSplineV
fSplineW
fSplineH
wRESERVED
sImage
bR bG bB bA
wBumpHeight
wDither
wScopeHandle  sScopeExpr fScope_a  fScope_b
wMapHandle    sMapExpr   fMap_a    fMap_b
wBumpHandle   sBumpExpr  fBump_a   fBump_b
wIndexHandle  sIndexExpr fIndex_a  fIndex_b
TagList
MAT_CREATE aMaterial
```



## OPERANDS

wFlags

The bits of this integer specify various material properties.

wMTF\_TRANCOL  
wMTF\_UNSHADED  
wMTF\_TILEX  
wMTF\_TILEY  
wMTF\_FLIPX  
wMTF\_FLIPY  
wMTF\_GRADEX  
wMTF\_GRADEY  
wMTF\_SPLINEMAP  
wMTF\_SCOPEMASK  
wMTF\_NOREFL  
wMTF\_EXCL  
wMTF\_SMOOTH

## wMapMethods

The bits of this integer specify the mapping types:

wMM\_COLOR  
wMM\_BUMP  
wMM\_BRILL  
wMM TRANSP  
wMM\_CLIP  
wMM\_SHADOW  
wMM\_ENVIRONM

## DESCRIPTION

Creates a material in the material library using the name specified with sName. It returns the address of the material created.

## 1.200 mat\_delete

## WORD

MAT\_DELETE

## TEMPLATE

aMaterial MAT\_DELETE

## DESCRIPTION

Deletes the material pointed by aMaterial from the material library.

## 1.201 mat\_find

WORD

MAT\_FIND

TEMPLATE

sName MAT\_FIND aMaterial

DESCRIPTION

Searches the material specified by sName from the material library and returns its address, or zero if the search failed.

## 1.202 mat\_lock

WORD

MAT\_LOCK

TEMPLATE

iAccess MAT\_LOCK

DESCRIPTION

Locks or unlocks the material data structure (material library) so that tasks can access it safely.

NOTE

THE MATERIAL DATA STRUCTURE MUST BE LOCKED BEFORE ANY ACCESS FROM RPL CODE. Failure to do so may cause RPL code to crash the system. The material data structure should be unlocked as soon as possible, because locking prevents other tasks from accessing it.

If the RPL code only reads materials then shared access can be used. If it changes materials, (MAT\_CREATE, MAT\_DELETE etc.), then exclusive access MUST be used.

See: 3.4 O\_LOCK for details of locking and iAccess.

## 1.203 err\_install

WORD

ERR\_INSTALL

TEMPLATE

---

aErrHook ERR\_INSTALL

#### PARAMETERS

aErrHook - address of the word to be called when an error occurs

#### DESCRIPTION

Installs a new error hook word to the RPL environment. When an error occurs the defined word is called. The number of possible hooks is not restricted in any way and they are called so that the hook installed first is called last. The called error hook word is removed by the system.

This word is usually needed for error handling. If a RPL program allocates any system resources, it should also deallocate them when the program is terminated.

#### NOTE

Error hooks cannot be nested. In other words, you cannot install error hook word for another error hook word.

#### ALSO SEE

ERR\_REMOVE

#### EXAMPLE

VARIABLE aMem

```
: MyErrHook
  "All Right"
  "This is my own error handler"
  GET_KEY DROP
    aMem @ 512 MEM_FREE
;

: DoSomething

  ( allocate some memory )
  512 0 MEM_ALLOC aMem !

  ( install error hook )
  & MyErrHook ERR_INSTALL

  ( then do something which causes error )
  #@!)=?

  ( this is the normal exit procedure )
  ( when everything went OK )
  & MyErrHook ERR_REMOVE
  aMem @ 512 MEM_FREE
;
```

DoSomething

## 1.204 err\_remove

WORD

ERR\_REMOVE

TEMPLATE

aErrHook ERR\_REMOVE

PARAMETERS

aErrHook - address of the RPL word installed with ERR\_INSTALL

DESCRIPTION

Removes an error hook word from the RPL environment. If the given RPL word is not installed, an error message is produced.

ALSO SEE

ERR\_INSTALL

EXAMPLE

```
: MyErrorHandler
  ( .... )
;
```

```
& MyErrorHandler ERR_INSTALL
( .... )
& MyErrorHandler ERR_REMOVE
```

## 1.205 mem\_alloc

WORD

MEM\_ALLOC

TEMPLATE

iSize iFlags ALLOC aAddr

PARAMETERS

iSize - the amount of memory to be allocated

---

iFlags - must be 0

#### RETURNS

aAddr - the address of the allocated memory

#### DESCRIPTION

Allocates given amount of memory from the system and returns the address of the allocated hunk. If allocation fails, 0 is returned.

#### ALSO SEE

MEM\_FREE

#### EXAMPLE

```
VARIABLE aMyStr

: AllocExample ( allocate 64 bytes of memory
  64 0 MEM_ALLOC aMyStr !
  aMyStr @ 63 "Enter any string" GET_STR
  IF
    aMyStr @ PUTS
  ENDIF
  aMyStr @ 64 MEM_FREE
;
```

## 1.206 mem\_free

#### WORD

MEM\_FREE

#### TEMPLATE

aAddr iSize MEM\_FREE

#### PARAMETERS

aAddr - the address of the memory to be freed  
iSize - the size of the memory to be freed

#### DESCRIPTION

Frees the memory allocated by MEM\_ALLOC.

#### ALSO SEE

---

MEM\_ALLOC

## 1.207 inherit

WORD

INHERIT

TEMPLATE

sName INHERIT

DESCRIPTION

The operand sName must be a valid RPL Environment (e.g. an existing RPL Window). The current RPL environment then inherits the vocabulary from this environment. When RPL looks for a word, the local vocabulary will be searched first, then any inherited definitions will be examined.

INHERIT does not enable closed inheritance 'loops'. If an environment attempts to INHERIT from an environment that has already inherited the first one, then INHERIT will be ignored.

There is one special RPL Environment called "Master" which is used for processing Macros and AREXX commands. This environment is created automatically by Real 3D and does not have a window associated with it.

EXAMPLE

"RPL.1" INHERIT

## 1.208 menu

WORD

MENU

TEMPLATE

iMenu iItem iSubItem MENU

DESCRIPTION

This word executes the menu function specified by the operands exactly as if it was selected using the mouse.

Menus are numbered from zero starting from the top left. Menu separator lines are counted as menu items. All three operands must be supplied even if there is no Sub-Menu. See appendix B of the manual for details of menu selection operands.

---

## EXAMPLE

```
2 2 1 MENU ( MODIFY/Properties/Name )
```

ALSO SEE List of Menu Numbers

## 1.209 refresh

## WORD

REFRESH

## TEMPLATE

REFRESH

## DESCRIPTION

Refreshes all windows using their individual refresh settings.

## 1.210 render

## WORD

RENDER

## TEMPLATE

RENDER

## DESCRIPTION

Renders all Views using their individual render settings

## 1.211 rot\_coord

## WORD

ROT\_COORD

## TEMPLATE

fxAngle fyAngle fzAngle aCoord ROT\_COORD

## DESCRIPTION

This rotates three vectors Vx, Vy, Vz in an array pointed by aCoord, around each other by the angles (in radians) specified by fxAngle, fyAngle & fzAngle. First Vy and Vz are rotated around Vx by fxAngle, then in the

resulting system Vz and Vx are rotated around Vy by fyAngle etc.

#### NOTE

Used extensively by the RPL Libraries.

## 1.212 scr\_save

#### WORD

SCR\_SAVE

#### TEMPLATE

sScreen sFile SCR\_SAVE l

#### DESCRIPTION

Saves the screen having the name sScreen to the file whose name is defined by sFile. It returns TRUE if the screen was found and saved, otherwise it returns FALSE.

#### EXAMPLE

"Real3D" "RAM:Real3D.IFF" SCR\_SAVE

## 1.213 system

#### WORD

SYSTEM

#### TEMPLATE

sCLI SYSTEM

#### DESCRIPTION

This passes the string sCLI to the OS CLI for execution.

#### EXAMPLE

"SYS:Tools/IconEdit" SYSTEM

## 1.214 wnd\_addr

#### WORD

WND\_ADDR

---



## TEMPLATE

sName WND\_ADDR aWnd

## PARAMETERS

sName - the name of any window

## RETURNS

aWnd - the address of the window

## DESCRIPTION

Returns the address of the window whose name is aName. If the window cannot be found, returns NULL.

## NOTE

Don't use this word if you don't know the internal structure of the window in question.

## 1.215 wnd\_open

## WORD

WND\_OPEN

## TEMPLATE

iType sName iLeft iTop iWidth iHeight WND\_OPEN

## PARAMETERS

iType - the type of the window to be opened. Can be one of the following:

iWT\_SELECT  
iWT\_VIEW  
iWT\_VIEWSB  
iWT\_VIEWBL  
iWT\_SHELL  
iWT\_TOOL  
iWT\_ANIM  
iWT\_PALETTE  
iWT\_VIEWDB  
iWT\_MATERIAL  
iWT\_SCREEN  
iWT\_MEASURE

sName - Name for the window.

iLeft, iTop - Top left edge of the window

wWidth, iHeight - Size of the window

## DESCRIPTION

Opens a Real 3D window.

#### EXAMPLE

```
( open the select window )

"editor.rpl" LOAD

iWT_SELECT "MyWindow" 100 10 150 100 WND_OPEN
```

## 1.216 wnd\_sendmsg

#### WORD

WND\_SENDMSG

#### TEMPLATE

```
0 aPn ... aP1 aWndName iMsgIde WND_SENDMSG iMsgNum
```

#### PARAMETERS

0 aPn ... aP1 - the addresses of parameters to be passed with the message  
aWndName - a string defining the target windows  
iMsgIde - a message identifier. Can be one of the following:

```
iWM_ACTIVATE
iWM_GETDATA
iWM_INTUIMSG
iWM_DIE
```

#### RETURNS

iMsgNum - a number of messages sent.

#### DESCRIPTION

Sends a message to given windows.  
The parameter list maximally consists of 5 addresses of parameters.  
An address 0 means that there are no more parameters to be passed.  
The purpose and amount of parameters depends on the message in question.

The window name can include wildcards so that it is possible to send the message to more than one window. The return value indicates the number of messages sent.

#### EXAMPLES

Bring the palette window to the front and activate it:

```
0 "Color" iWM_ACTIVATE WND_SENDMSG DROP
```

Kill the 'Color' window:

---

```
0 "Color" iWM_DIE WND_SENDSMSG DROP

MyView window to front if exists. If not, create it:

0 "MyView" iWM_ACTIVATE WND_SENDSMSG
NOT IF
    iWT_VIEW "MyView" 10 10 300 200 WND_OPEN
ENDIF
```

## 1.217 ray\_inters

WORD

RAY\_INTERS - Ray/surface intersection

TEMPLATE

aHandle avPos avDir avInters avNorm RAY\_PREP iResult

PARAMETERS

aHandle - a handle from the RAY\_PREP word  
avPos - the address of a vector defining the position of the ray  
avDir - the address of a vector defining the direction of the ray  
avInters - the address of a vector to contain the intersection point  
avNorm - the address of a vector to contain the surface normal in  
the intersection point

RETURNS

iResult - TRUE if intersection was detected, FALSE if no intersection found.

DESCRIPTION

Executes an intersection test between a given object (aHandle) and a given ray (avPos and avDir). If no intersection was found, FALSE is pushed on the stack. Otherwise the variables avPos and avDir contain the position and the normal vector of the surface where the intersection between object and ray was detected.

NOTE

This word can be used for creating 'behavioral' animations where objects observe their living environment making decisions and conclusions depending on the information they receive. For example, a flying object can try to avoid hitting other objects a RPL method using this word.

EXAMPLE

```
( check if there is an object )
( somewhere in front of us )
```

```

VVARIABLE vPos
VVARIABLE vDir
VVARIABLE vHit
VVARIABLE vNrm
VARIABLE iHnd
100 STRING sBuff

: MyCollTest

    ( prepare intersection )
    "/Root/Enemy" RAY_PREP iHnd !

    ( shoot some rays )
    0 0 0 vPos V! ( start point of the ray )
    1 0 0 vDir V! ( direction of the 1st ray )
    iHnd vPos vDir vHit vNrm RAY_INTERS
    IF
        vHit V@
        "Hit found in position %g %g %g"
        sBuff SPRINTF
        sBuff PUTS
    ENDIF

    0 1 0 vDir V! ( direction of the 2nd ray )
    iHnd vPos vDir vHit vNrm RAY_INTERS
    IF
        vHit V@
        "Hit found in position %g %g %g"
        sBuff SPRINTF
        sBuff PUTS
    ENDIF

    ( free intersection handle )
    iHnd @ RAY_FREE
;

```

ALSO SEE

RAY\_PREP      RAY\_FREE

## 1.218 ray\_free

WORD

RAY\_FREE - Free a ray intersection handle

TEMPLATE

aHandle RAY\_FREE

PARAMETERS

aHandle - pointer to a ray intersection handle

## DESCRIPTION

Deallocates the data structures needed for ray intersection testing.

## EXAMPLE

```
VARIABLE RayHandle

"/Root/Tube" O_FIND RAY_PREP RayHandle !

....

RayHandle @ RAY_FREE
```

## ALSO SEE

RAY\_INTERS      RAY\_PREP

## 1.219 ray\_prep

## WORD

RAY\_PREP - prepare a ray/surface intersection handle

## TEMPLATE

```
aObject RAY_PREP aHandle
```

## PARAMETERS

aObject - a pointer to an object

## RETURNS

aHandle - A ray intersection handle

## DESCRIPTION

Prepares data structures needed for ray intersection testing

## ALSO SEE

RAY\_INTERS      RAY\_FREE

## 1.220 busy\_cancel

## WORD

```
BUSY_CANCEL
```

## TEMPLATE

```
aHnd BUSY_CANCEL iBool
```

## PARAMETERS

aHnd - a return value from the BUSY\_OPEN word

## RETURNS

iBool - TRUE or FALSE (1/0)

## DESCRIPTION

Checks whether or not the user has pressed the CANCEL gadget of a given busy requester. If the gadget is pressed, TRUE is pushed on the stack, otherwise FALSE.

## ALSO SEE

BUSY\_OPEN      BUSY\_CLOSE      BUSY\_UPDATE

## EXAMPLE

```
VARIABLE aBusyHnd

: BusyTest
  100 0 DO
    aBusyHnd @ "Rendering..."
    I BUSY_UPDATE ( Rendering )
    aBusyHnd @ BUSY_CANCEL
    IF
      LEAVE
    ENDIF
  LOOP
  aBusyHnd @ BUSY_CLOSE
;
```

## 1.221 busy\_close

## WORD

BUSY\_CLOSE

## TEMPLATE

```
aHnd BUSY_CLOSE
```

## PARAMETERS

aHnd - a return value from the BUSY\_OPEN word

## DESCRIPTION

Closes a given busy requester.

ALSO SEE

BUSY\_OPEN      BUSY\_UPDATE      BUSY\_CANCEL

EXAMPLE

```
aBusyHnd @ BUSY_CLOSE
```

## 1.222 busy\_open

WORD

BUSY\_OPEN

TEMPLATE

```
sHeader BUSY_OPEN aHnd
```

PARAMETERS

sHeader - title string

RETURN

aHnd - address of the busy requester.

DESCRIPTION

Opens a busy requester with the given header text and returns a handle which can be used for updating the busy requester and checking whether the user has requested cancelling.

ALSO SEE

BUSY\_CANCEL      BUSY\_CLOSE      BUSY\_UPDATE

EXAMPLE

VARIABLE aHnd

```
: MyAnimation
  "Rendering Animation..." BUSY_OPEN aHnd !
  100 0 DO
    O_GETSEL
    0.1 0 0 0 M_MOVE
    REFRESH
    aHnd @ 0 I BUSY_UPDATE
  LOOP
  aHnd @ BUSY_CLOSE
;
```

## 1.223 busy\_update

WORD

BUSY\_UPDATE

TEMPLATE

aHnd aNewHdr iPer BUSY\_UPDATE

PARAMETERS

aHnd - a return value from the BUSY\_OPEN word  
aNewHdr - a new header text for the requester  
iPer - value between 0 and 100.

DESCRIPTION

Updates the contents of the busy requester. If the aNewHdr parameter is not 0, then it is assumed to be the address of the new header text for the requester. If it is NULL, the header text of the requester is not changed. The iPer parameter must be between 0 and 100.

ALSO SEE

BUSY\_OPEN      BUSY\_CLOSE      BUSY\_CANCEL

EXAMPLE

```
VARIABLE aBusyHnd

: BusyTest
  "Optimizing ..." BUSY_OPEN aBusyHnd !
  100 0
    aBusyHnd @ 0 I
    BUSY_UPDATE ( Optimizing )
    LOOP
    100 0 DO
    aBusyHnd @ "Rendering..." I
    BUSY_UPDATE ( Rendering )
    LOOP
  aBusyHnd @ BUSY_CLOSE
;
```

## 1.224 get\_key



WORD

GET\_KEY

TEMPLATE

aGadTxts aHdrTxt GET\_KEY iRetVal

PARAMETERS

aGadTxts - string defining gadgets to be created. Gadget texts are separated by the character '|'.  
aHdrTxt - Headline string for the requester.

RETURN

iRetVal - integer corresponding the selected gadget.  
The value corresponding the first (leftmost) gadget is 1 and the return value corresponding the rightmost gadget is 0 (cancel/negative choice should always be the rightmost one as suggested by the Amiga Style Guide). Intermediate gadgets/return values are incremented by one from left to right.

DESCRIPTION

Opens a requester with a given header text and gadgets, waiting the user to select one of them. The function returns a value corresponding the selected gadget.

EXAMPLE

```
"FIRST|SECOND|THIRD|CANCEL"  
"Select One of These" GET_KEY .
```

## 1.225 get\_flt

WORD

GET\_FLT

TEMPLATE

aFlt aHdr GET\_FLT iRetVal

PARAMETERS

aFlt - pointer to a floating point  
aHdr - headline text string for the requester

RETURN

iRetVal - TRUE of FALSE depending on the user's action

EXAMPLE

---

Opens a requester allowing the user to define a floating point value. Formula evaluation is supported.

#### EXAMPLE

```
( define float variable )
FVARIABLE MyFlt

: MyTest
  3.14 MyFlt F!
  MyFlt "Define Angle" GET_FLT
  IF
    MyFlt F@ F.
  ENDIF
;

MyTest
```

## 1.226 get\_str

#### WORD

GET\_STR

#### TEMPLATE

```
aStr iLen aHdr GET_STR iRet
```

#### PARAMETERS

aStr - a pointer to a buffer  
iLen - the maximum length of the string (length of the buffer - 1)  
aHdr - a header text for the requester

#### RETURN

iRet - 1 or 0 depending on the user's choice.

#### DESCRIPTION

Opens a requester allowing the user to define a string.

#### EXAMPLE

```
( create an object with custom name )

"creation.rpl" LOAD

16 STRING ObjNam

: MyTest
  "Noname" ObjNam CPY
  ObjNam 16 "Create Object" GET_STR
  IF
    wOT_OR ObjNam 0 "CEND" C_LEVEL DROP
  ENDIF
```

;

## 1.227 get\_vect

WORD

GET\_VECT

TEMPLATE

aVct aHdr GET\_VECT iRet

PARAMETER

aVct - pointer to a vector (an array of 3 floating points)  
 aHdr - a header text for the requester

RETURN

iRet - TRUE or FALSE depending on the user's choice.

DESCRIPTION

Opens a requester allowing the user to define three floating point values, in other words a 3D vector. Formula evaluation is supported.

EXAMPLE

```
( move selected objects )

"vectors.rpl" LOAD

VARIABLE vDelta

: MyTest
  0.0 1.0 3.1 vDelta V!
  vDelta "Move Selected Objects" GET_VECT
  IF
    O_GETSEL
    vDelta F@ 0 M_MOVE
  ENDIF
;
```

## 1.228 get\_file

WORD

GET\_FILE

TEMPLATE

iType aNam aDir aHdr GET\_FILE iRet

## PARAMETER

iType - One of the following action qualifiers:  
 iIO\_READ - Selection is used for reading  
 iIO\_WRITE - Selection is used for writing  
 iIO\_DIR - Selection is used for defining a path

aNam - pointer to an initial file name/result string  
 aDir - pointer to an initial directory path  
 aHdr - a header text for the requester

## RETURN

iRet - TRUE or FALSE depending on the user's choice.

## DESCRIPTION

Opens a file requester on a given DOS drawer (directory) allowing the user to select a file name. If the user moves in the hierarchy, the contents of the buffer pointed by 'iDir' is updated accordingly. If the user selects OK, the buffer pointed by 'iNam' will contain the complete file name (including the path). Note that both buffers should be large enough to hold the result strings.

## EXAMPLE

```
( load a Real 3D IFF file )

"io.rpl" LOAD

256 STRING Name
256 STRING Path

: MyLoad
  "myfile"      Name CPY
  "r3d2:projects" Path CPY

  iIO_READ Name Path "Load something" GET_FILE
  IF ( replace all sections: )
    Name lIO_RALL lIO_RALL FIL_LOAD
  ENDIF
;

MyLoad
```

**1.229 rx**

## WORD

RX

## TEMPLATE

sPrg RX

---

## PARAMETER

sPrg - string defining the arexx program to be executed

## DESCRIPTION

This word sends a given ARexx program to the ARexx manager process as if it was typed in from an Amiga DOS shell using the rx command.

Note that if the RX word does not return 0, it is not automatically interpreted as an error situation. The reason for this is that some applications use the return value for indicating something else than an error.

## EXAMPLE

```
"ADDRESS COMMAND dir" RX
"RENDER" RX
```

## ALSO SEE

RX\_RC

## 1.230 rx\_rc

## WORD

RX\_RC

## TEMPLATE

RX\_RC - aRetVal

## RETURNS

aRetVal - The address of the return value

## DESCRIPTION

This word can be used for fetching the return value from a previously executed RX word. RX\_RC returns the address of an integer variable containing the result.

## EXAMPLE

```
( send ARexx command )
"ADDRESS STANDALONE RENDER" RX
RX_RC @    ( fetch return value )

IF ( terminate the RPL program )
  "ERROR:Cannot Execute" ERROR
```

ENDIF

#### NOTE

Although the word returns the address of the return value, currently it does not make sense to assign a return value to ARexx commands arriving to Real 3D's ARexx port.

## 1.231 rx\_result

#### WORD

RX\_RESULT

#### TEMPLATE

RX\_RESULT - aResStr

#### RETURNS

aResStr - Address of the result string

#### DESCRIPTION

This word can be used for accessing the result string variable of Real 3D. The word returns the address of the result string. RPL programs can read this variable after each sent ARexx message in order to detect the possible result strings returned by external applications. The word can also be used for passing a result string to an external application who have sent an ARexx message to the ARexx port of Real 3D.

Note that a result string is returned only if it is requested by the command sender application. Furthermore, all commands do not return a result string even if it is requested.

If the ARexx command sent to Real 3D fails, result string is NOT returned. In other words, the result string is valid only if the return code indicates that no error occurred during command processing.

#### EXAMPLE

```
.... "RX  ( send ARexx command )
RX_RC @  ( fetch return value )
IF
    "ERROR:Cannot Execute" ERROR
ENDIF
RX_RESULT PUTS ( process result string )
```

## 1.232 rx\_setclip

WORD

RX\_SETCLIP

TEMPLATE

sName sValue RX\_SETCLIP

PARAMETERS

sName - Address of the name string  
sValue - Address of the value string

DESCRIPTION

This word can be used for putting new items to the Clip List or updating existing ones. Each item in the Clip List consists of a pair of strings defining a name and a value for the item. The Clip List can be used for passing information between Real 3D and ARexx and is typically used whenever two or more result strings are needed.

EXAMPLE

"rad" "0.5" RX\_SETCLIP  
"position", "0.5, 0.8, 0.0" RX\_SETCLIP

### 1.233 vvariable

WORD

VVARIABLE

TEMPLATE

VVARIABLE name

DESCRIPTION

Defines a vector variable. The name of the variable must follow the word VVARIABLE. The variable is initialized as 0 0 0.

A vector variable is an array of three floating point variables. When the variable is later referenced by entering its name, the address of the first floating point is pushed onto the stack.

EXAMPLE

VVARIABLE myVector

ALSO SEE

V@                    V!

## 1.234 v!

WORD

V!

TEMPLATE

fX fY fZ aVariable V!

PARAMETERS

fX fY fZ - three values defining a vector  
aVariable - an address of the vector variable

DESCRIPTION

Store a vector in a variable.

EXAMPLE

VVARIABLE myVar  
0 0 1.5 myVar V!

## 1.235 v@

WORD

V@

TEMPLATE

aVariable V@ fX fY fZ

PARAMETERS

aVariable - the address of a vector variable

RETURNS

fX fY fZ - contents of the variable

DESCRIPTION

Fetch a vector from a given address.

EXAMPLE

( print the value of vMyVar )  
vMyVar F@ V.

---



## 1.236 vadd

WORD

VADD

TEMPLATE

v1 v2 VADD vRes

PARAMETERS

v1 v2 - two vectors to be added

RETURNS

vRes - result vector

DESCRIPTION

Vector addition. Pulls two vectors off the stack and pushes the result vector back.

EXAMPLE

```
VARIABLE v1 VARIABLE v2 VARIABLE vRes  
  
1 0 0 v1 V!      ( v1 = 1 0 0 )  
0 1 0 v2 V!      ( v2 = 0 1 0 )  
v1 V@ v2 V@ VADD vRes V! ( vRes = v1 + v2 )
```

## 1.237 vsub

WORD

VSUB

TEMPLATE

v1 v2 VSUB vRes

PARAMETERS

v1, v2 - vectors to be subtracted

RESULT

vRes - result

DESCRIPTION

Pulls two vectors off the stack, subtracts them and pushes the result back onto the stack.

---

## EXAMPLE

```
10 5.0 0.5 ( v1 )
 5 5.0 0.5 ( v2 )
VSUB
V.
```

## 1.238 vmul

## WORD

VMUL

## TEMPLATE

```
v f VMUL vRes
```

## PARAMETERS

```
v - a vector to be multiplied
f - a scalar value
```

## RESULT

```
vRes = v1 * f
```

## DESCRIPTION

Multiplies a vector by a scalar. In other words, each component of the given vector 'v' is multiplied by the given float 'f' and the result is pushed onto the stack.

## EXAMPLE

```
( duplicate the length of a vector )
1.5 3.1 8.2 2.0 VMUL V.
```

## 1.239 vdot

## WORD

VDOT

## TEMPLATE

```
v1 v2 VDOT fRes
```

## PARAMETERS

```
v1, v2 - two vectors to be operated
```

## RESULT

fRes - Dot product of given vectors

#### DESCRIPTION

Pulls two vectors off the stack, executes dot product and pushes the result (a floating point value) back onto the stack.

#### EXAMPLE

```
( dot product of perpendicular vectors = 0 )  
1 0 0 0 1 0 VDOT F.
```

## 1.240 vcros

#### WORD

VCROS

#### TEMPLATE

v1 v2 VCROS vRes

#### PARAMETERS

v1, v2 - two vectors

#### RESULT

vRes - result vector

#### DESCRIPTION

Cross product. The result vector is always perpendicular to the operands v1 and v2.

#### EXAMPLE

```
1 0 0 0 1 0 VCROS V. ( result = 0 0 1 )
```

## 1.241 vnorm

#### WORD

VNORM

#### TEMPLATE

v VNORM vRes

#### PARAMETERS

v - vector to be normalized

RESULT

vRes - unit vector

DESCRIPTION

Vector normalization. The given vector is divided by its length and the result is pushed back on to the stack. The length of the result vector is always 1.

EXAMPLE

```
120.2 10.2 -2.1 VNORM
```

## 1.242 vlen

WORD

VLEN

TEMPLATE

v VCROS fLen

PARAMETERS

v - any vector

RESULT

fLen - the length of 'v'

DESCRIPTION

Pulls the given vector off the stack and puts the length of it back to the stack.

EXAMPLE

```
10 20 30 VLEN F.
```

## 1.243 v.

WORD

V.

## TEMPLATE

```
v V.
```

## PARAMETERS

```
v - vector
```

## DESCRIPTION

```
Pulls a vector ( three float values ) off the stack and
prints them out.
```

## 1.244 vconstant

## WORD

```
VCONSTANT
```

## TEMPLATE

```
VCONSTANT name
```

## DESCRIPTION

```
Defines a vector constant.
```

## EXAMPLE

```
( some useful constants )
1 0 0 VCONSTANT vX
0 1 0 VCONSTANT vY
0 0 1 VCONSTANT vZ

vX V.
```

## 1.245 geometry

## GEOMETRY DATA FOR CREATION WORDS

The structure of this section depends on the object in question and some objects like 'links' and 'levels' don't have this section at all. The purpose of this section is to describe the geometry for the object to be created. Although the structure is this section is different for all objects, they include some common data.

## 1.246 [fstangle fenangle]

## SECTOR ANGLES

These parameters describe the angle of the sector in terms of the object space of the primitive starting from `fStAngle` around anti-clockwise to `fEnAngle`. These two floating-point operands are optional. If a sector primitive is required then bit 12 of `iAttr` must be set and these two operands must be supplied.

Only the following quadric primitives can be sectorized:

- cone
- cut-cone
- cylinder
- ellipse
- ellipse-segment
- hyperboloid

## 1.247 wgeomflags

### GEOMETRY FLAGS

This parameter describes additional geometry information for freeforms.

- `wGF_CLOSEU` - u dimension periodic: 0 - open, 1 - closed
- `wGF_CLOSEV` - v dimension periodic: 0 - open, 1 - closed
- `wGF_SECTOR` - set if sector primitive
- `wGF_PERIODIC` - open/closed evaluation for animations

`wGF_CLOSEU` and `wGF_CLOSEV` flags can be used for closing freeform objects. If the flag is set, the curve/surface is closed in corresponding direction. Note that curves are only sensitive to the flag `wGF_CLOSEU`.

The `wGF_SECTOR` flag is set whenever the object is sector primitive. This flag tells to the evaluation system whether or not to treat the primitive as a sector.

If the flag `wGF_PERIODIC` is set, closed curves are evaluated so that their end point is the same as their beginning point. In order to use closed loops for generating continuous motions for loop-animations, set this flag.

## 1.248 wfreetype

### FREEFORM TYPES

This specifies how the point data of a freeform is evaluated:

- `wFT_POLYGON` - polygonal line or surface
- `wFT_PHONG` - phong shaded surface (treated as polygonal for lines)
- `wFT_BSPLINE` - Cubic B-Spline curve/surface

---

## 1.249 icolor

### COLOR

Color section consists of four integer values defining a color for the object to be created. Whether or not this section should exists, depends on the object in question. The RPL format of this section is the following:

```
bR bG bB bA ( Red Green Blue Alpha )
```

## 1.250 attributes

### ATTRIBUTES

This section is required by all creation words. The attributes data section consists of two different parameters: name and object flags.

#### Name

The name of the object to be created can be up to 16 characters long. If a longer string is supplied, it will be truncated.

#### Object Flags

The flags field is an integer value containing 'yes/no' (on/off) kind of information for the object to be created.

Flag	Description
1OF_INVERTED	Volume inverted in Boolean Operations
1OF_PAINTED	Surface properties affects in Booleans
1OF_WFINVISIBLE	Wire frame is invisible
1OF_LIGHTSOURCE	Object is a light source
1OF_HOLLOW	Represented as a surface instead of solid object
1OF_INFINITE	Infinite object
1OF_SCENE	Invisible in primary ray tracing
1OF_RTINVISIBLE	Ray tracing invisible
1OF_NOBP1	No 1st. bounding plane
1OF_NOBP2	No 2nd bounding plane
1OF_TEXTURE	Primitive used for mapping textures to objects
1OF_SECTOR	Sector Primitive
1OF_PROTECTED	Primitive cannot be modified
1OF_SEGMENT	Segment instead of sector
1OF_NOTREFL	Not reflected
1OF_MOTION	Motion Blurred object
1OF_SHADOWLESS	Does not Cast Shadows
1OF_MATTE	Matte Object
1OF_SECTOR	Sector primitive
1OF_SEGMENT	Segment primitive

## 1.251 creation tags

### TAGS FOR CREATION WORDS

The tags section consists of a list of pairs of tag values and tag ID's in that order. The list must terminate with "CEND", which because of the RPN nature of the RPL language must be entered first.

## 1.252 modify flags

### MODIFY FLAGS

FLAG	DESCRIPTION
-----	
LMF_ROTTEXT	Rotate & Extend if set
LMF_NOSUB	0 - Modify sub-objects, 1 - Only modify targets
LMF_NOCOG	About COGs if set
LMF_COGONLY	With COGs if set
LMF_BNDSize	0 - M_BEND Move, 1 - M_BEND Size
LMF_PARABOL	
LMF_LINEAR	
LMF_SPHERE	
LMF_CURVE	
LMF_SIN	

## 1.253 getvstack

### WORD

GETVSTACK

### SYNTAX

iCnt GETVSTACK iActual ... vPl

### PARAMETERS

iCnt - Number of items to be fetched or zero for fetch all.

### RESULT

iActual - Actual number of items fetched.  
 ... vPl - Vectors fetched from the vector stack.

### DESCRIPTION

Attempts to fetch 'iCnt' vectors from the Vector Stack to the RPL stack. If the 'iCnt' is zero, all items are fetched. The 'iActual' holds the actual number of fetched vectors which may be less than asked number.



## EXAMPLE

```
( Move selected object to given position )

: MoveTo
  O_GETSEL      ( fetch selected objects
  1 GETVSTACK   ( attempt fetch one item from the vector stack
  IF            ( make sure we really got what we asked
    0 M_MOVECOG ( move objects to fetched position
  ENDIF
;

```

## 1.254 database

## WORD

DATABASE

## SYNTAX

sDataName DATABASE aAddr

## PARAMETERS

sDataName - name of the data structure to be fetched

## RESULT

aAddr - address of the data structure

## DESCRIPTION

Attempts to fetch the address of given data structure on stack.  
The following data names are supported:

"real" - Real 3D root data structure  
"editor" - Editor  
"anim" - Animation System

## NOTE

This word is intended for applications which need to deal with  
the details of the internal data structures of Real 3D.

## 1.255 wnd\_lock

## WORD

WND\_LOCK

## SYNTAX

iAccess WND\_LOCK

#### PARAMETERS

iAccess - iLOCK\_EXCL, iLOCK\_SHARED, iLOCK\_REMOVE

#### DESCRIPTION

Attempts to lock the window list of Real 3D. This word must be called prior to using the word WND\_ADDR, to prevent closing the window while the internal data of it is accessed.

#### NOTE

Do not use this word if you don't know the internal structure of the window in question.

## 1.256 inside\_prep

#### WORD

INSIDE\_PREP

#### SYNTAX

aObj INSIDE\_PREP aObjHnd

#### PARAMETERS

aObj - an object to be prepared for inside/outside test

#### RESULT

aObjHnd - Handle to the prepared object

#### DESCRIPTION

Prepares a given object for Inside/Outside testing. This makes sense only to solid objects.

#### ALSO SEE

INSIDE\_TEST    INSIDE\_FREE

## 1.257 inside\_test

#### WORD

INSIDE\_TEST

#### SYNTAX

---

```
aObjHnd vPoint  INSIDE_TEST iBool
```

#### PARAMETERS

aObjHnd - Handle to object prepared with the word INSIDE\_PREP  
vPoint - address of vector to be tested

#### RESULT

iBool - 1 if the vPoint was inside the object aObjHnd, otherwise 0

#### DESCRIPTION

Tests whether the given point 'vPoint' is inside the object 'aObjHnd'.  
If not, returns zero, otherwise 1.

#### ALSO SEE

INSIDE\_PREP    INSIDE\_FREE

## 1.258 inside\_free

#### WORD

INSIDE\_FREE

#### SYNTAX

```
aObjHnd  INSIDE_FREE
```

#### PARAMETERS

aObjHnd - Handle to an object prepared with the word INSIDE\_PREP

#### DESCRIPTION

Frees the memory allocated by INSIDE\_PREP.

#### ALSO SEE

INSIDE\_PREP    INSIDE\_TEST

---