rpl

COLLABORATORS

	TITLE : rpl	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		October 23, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

rpl

1.1 rpl.guide

```
REAL 3D PROGRAMMING LANGUAGE - RPL

     Locking

     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 locking

```
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:

    "r3d3: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            STORE
  &             +             -             *             /
  :             ;             <             <=            =
  >             >=            <>            >R            >RAD
  ?&            ?DUP          ?ELSE         ?ENDIF        ?IF
  FETCH         AGAIN         AND           ACOS          ASIN
  ATAN          B.            BFETCH        BAND          BEGIN
  BNOT          BOR           BXOR          CAT           COMPARE
  CONSTANT      COS           CPY           DEPTH         DO
  DROP          DUP           ELSE          EMIT          ENDIF
  ERROR         EVAL          EXECUTE       EXIT          EXP
  F.            FSTORE        F+            F-            F*
  F/            F<            F<=           F<>           F=
  F>            F>=           F>I           FFETCH        FCONSTANT
  FMOD          FORGET        FVARIABLE     H.            I
```

```
  I>F          IF           J            K            LEAVE
  LEN          LOAD         LOG          LOG10        LOOP
  +LOOP        MOD          NCPY         NCAT         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        WSTORE
  WFETCH       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_SKELETON    C_TRISET
         C_VIEWPOINT
```

1.5 modification

```
MODIFICATION WORDS
         CS_DEFINE     CS_ROTATE
         CS_SCALE      CS_TRANSFABS
         CS_TRANSLATE

         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_MAKENAME
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

FCLOSE
FEOF
FGETS
FOPEN
FPUTS
FREAD
FSEEK
FTELL
FWRITE
FIL_LOAD
FIL_SAVE
REMOVE
SCANDIR

1.9 materials

MATERIAL SPECIFIC WORDS

MAT_CREATE
MAT_DELETE
MAT_FIND
MAT_LOCK

1.10 miscs

```
MISCELLANEOUS WORDS

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

1.11 userinterface

```
USER INTERFACE WORDS

         ADDMENU
         GET_KEY
         GET_STR
         GET_FLT
         GET_VECT
         GET_FILE
         PRJ_SETHOOK
         UI_WINDOW
         UI_BUTTON
         UI_CHECKBOX
         UI_STRING
         UI_TEXT
         UI_SLIDER
         UI_MX
         UI_DELETE
         UI_GETATTRS
         UI_SETATTRS
         UI_REALIZE
```

1.12 arexx

```
AREXX WORDS

         RX
         RX_RC
         RX_RESULT
         RX_SETCLIP
```

1.13 vectors

```
VECTOR OPERATIONS

         VVARIABLE
         VSTORE
         VFETCH
         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 addmenu

```
WORD

  ADDMENU
```

```
TEMPLATE

  aWord sName ADDMENU

DESCRIPTION

  Adds a new menu item to the menu strip. When the menu is selected, the
  associated RPL word is called.

  If the name string contains the character '/', a new sub item is created.
  Otherwise a new menu item is created.


EXAMPLE

  This example creates a new menu item 'MyTools' consisting of two sub items:
  'Fancy Tool' and 'Cool Tool'.

  : myFancyTool
      "Fancy Tool selected" PUTS
  ;

  : myCoolTool
      "Cool Tool selected" PUTS
  ;

  & myFancyTool "MyTools/Fancy Tool" ADDMENU
  & myCoolTool  "MyTools/Cool Tool"  ADDMENU
```

1.16 compare

```
WORD

  COMPARE

TEMPLATE

  sFirst sSecond COMPARE iCmp

RETURNS

  The return value indicates the lexicographic relation
  of sFirst to sSecond, as follows:

  iCmp < 0    sFirst is less than sSecond
  iCmp = 0    sFirst is equal to sSecond
  iCmp > 0    sFirst is greater than sSecond

PARAMETERS

  sFirst, sSecond - strings to be compared

DESCRIPTION

  Compares two strings lexicographically and returns a value indicating their
```

```
  relationship.
```

1.17 cs_define

```
WORD

  CS_DEFINE

TEMPLATE

  vX vY vZ vOrigin CS_DEFINE

PARAMETERS

  vX vY vZ - Three axes defining the orientation of the coordinate system
  vOrigin - Origin

DESCRIPTION

  Defines the current coordinate system for CS_ROTATE, CS_SCALE, CS_TRANSLATE
  and CS_TRANSFABS words.

EXAMPLE

  0 0 0
  1 0 0
  0 1 0
  0 0 1
  CS_DEFINE
```

1.18 cs_rotate

```
WORD

  CS_ROTATE

TEMPLATE

  0 ... aObj vDir fAngle CS_ROTATE

DESCRIPTION

  Rotates objects about a given axis that passes through the
  coordinate origin. Rotation is applied in the current coordinate system.

EXAMPLE

  O_GETSEL
  1 0 0 90 CS_ROTATE

SEE ALSO
```

```
  CS_DEFINE
```

1.19 cs_scale

```
WORD

  CS_SCALE

TEMPLATE

  0 ... aObj vScale CS_SCALE

DESCRIPTION

  Scales the given objects in the current coordinate system.

EXAMPLE

  O_GETSEL
  0.5 1 1 CS_SCALE

SEE ALSO

  CS_DEFINE
```

1.20 cs_transfabs

```
WORD

  CS_TRANSFABS

TEMPLATE

  0 ... aObj CS_TRANSFABS

DESCRIPTION

  Modifies the given objects so that their object spaces will match
  the current coordinate system.

SEE ALSO

  CS_DEFINE
```

1.21 cs_translate

```
WORD

  CS_TRANSLATE
```

```
TEMPLATE

  0 ... aObj vMove CS_TRANSLATE

DESCRIPTION

  Moves the given objects. The transformation is applied to objects in
  the current coordinate system.

EXAMPLE

  0.1 0 0 CS_TRANSLATE

SEE ALSO

  CS_DEFINE
```

1.22 datetime

```
WORD

  DATETIME

TEMPLATE

  DATETIME iSec iMin iHour iMday iMon iYear

RETURNS

  Current date and time

DESCRIPTION

  Returns the current date and time as follows:
  iSec  - seconds after the minute (0-59)
  iMin  - minutes after the hour (0-59)
  iHour - hours since midnight (0-23)
  iMday - day of month (1-31)
  iMon  - month (1-12, january = 1)
  iYear - current year
```

1.23 fclose

```
WORD

  FCLOSE

TEMPLATE

  aFile FCLOSE iSuccess

RETURNS
```

```
  0 if FCLOSE succeeds

PARAMETERS

  aFile - pointer to a file

DESCRIPTION

  Closes the file opened by FOPEN.
```

1.24 feof

```
WORD

  FEOF

TEMPLATE

  aFile FEOF iAtEof

RETURNS

  nonzero if at end of file, 0 otherwise

PARAMETERS

  aFile - pointer to a file

DESCRIPTION

  Returns a nonzero value after the first read operation that attempts
  to read past the end of the file. It returns 0 if the current position
  is not end-of-file.
```

1.25 fgets

```
WORD

  FGETS

TEMPLATE

  sString iMaxLen aFile FGETS iSuccess

RETURNS

  sString if FGETS succeeds, NULL otherwise

PARAMETERS

  sString - buffer for data to be read
```

```
  iMaxLen - maximim number of characters to read
  aFile - pointer to a file

DESCRIPTION

  FGETS reads a string from the file to the buffer.
  Characters are read from the current file position up to and
  including the first newline character, up to the end of the file,
  or until the number of characters read is equal to iMaxLen-1,
  whichever comes first. The result is stored in sString,
  and a null character is appended. The newline character, if read,
  is included in the string.
```

1.26 fopen

```
WORD

  FOPEN

TEMPLATE

  sFilename sMode FOPEN aFile

RETURNS

  Pointer to the opened file or NULL if failed

PARAMETERS

  sFilename - Filename

  sMode - Access type

DESCRIPTION

  Opens the file specified by filename.
  The sMode parameter specifies the type of access requested for the file:

  "r"     Opens for reading only. If the file does not exist or cannot be found, ↩
     the FOPEN word will return NULL.
  "w"     Opens a file for writing. If the given file exists, its contents are ↩
     destroyed.
  "a"     Opens for writing at the end of the file (appending); creates the file ↩
     first if it doesn't exist.
  "r+"    Opens for both reading and writing. The file must exist.
  "w+"    Opens a file for both reading and writing. If the given file exists, its ↩
      contents are destroyed.
  "a+"    Opens a file for reading and appending; creates the file if it doesn't ↩
     exist.

  When the "r+", "w+", or "a+" access type is specified,
  both reading and writing are allowed.
  When switching between reading and writing, there must be an intervening FSEEK ↩
     operation.
```

```
  The following characters can be included in sMode to specify the translation  ↩
     mode for newline characters:

  t    Open the file in text mode. In text mode, carriage-return-linefeed (CR-LF)  ↩
     combinations are translated into single linefeeds (LF) on input and LF
       characters are translated to CR-LF combinations on output. Also , CTRL+Z is ↩
            interpreted as an end-of-file character on input. In files opened for
       reading/writing, FOPEN checks for a CTRL+Z at the end of the file and  ↩
          removes it, if possible. This is done because using the FSEEK and FTELL  ↩
          words
       to move within a file that ends with a CTRL+Z may cause FSEEK to behave  ↩
          erratically near the end of the file.
  b    Open the file in binary mode. The translations are not carried out.
```

1.27 fputs

```
WORD

  FPUTS

TEMPLATE

  sString aFile FPUTS iSuccess

RETURNS

  a non-negative value if FPUTS succeeds

PARAMETERS

  sString - string to be written
  aFile - pointer to a file

DESCRIPTION

  Writes the string to the file.
```

1.28 fread

```
WORD

  FREAD

TEMPLATE

  sBuffer iSize iCount aFile FREAD iRead

RETURNS

  number of items fully read

PARAMETERS
```

```
  sBuffer - buffer for data to be read
  iSize - size of one item
  iCount - number of items to be read
  aFile - pointer to a file

DESCRIPTION

  Reads up to iCount items from aFile file to sBuffer. Each item is of
  length iSize.
```

1.29 fseek

```
WORD

  FSEEK

TEMPLATE

  aFile iOffset iOrigin FSEEK iSuccess

RETURNS

  0 if FSEEK succeeds

PARAMETERS

  aFile - pointer to a file
  iOffset - number of bytes from origin
  iOrigin - initial position

DESCRIPTION

  Repositions the pointer in a file. The iOrigin parameter can be any of the
  following
  SEEK_CUR - current file position
  SEEK_END - end of file
  SEEK_SET - beginning of file
```

1.30 ftell

```
WORD

  FTELL

TEMPLATE

  aFile FTELL iPosition

RETURNS

  current file position
```

```
PARAMETERS

  aFile - pointer to a file

DESCRIPTION

  Returns the current file position.
```

1.31 fwrite

```
WORD

  FWRITE

TEMPLATE

  sBuffer iSize iCount aFile FWRITE iWritten

RETURNS

  number of items written

PARAMETERS

  sBuffer - data to be written
  iSize - size of one item
  iCount - number of items to be written
  aFile - pointer to a file

DESCRIPTION

  Writes up to iCount items from sBuffer to aFile file. Each item is of
  length iSize.
```

1.32 len

```
WORD

  LEN

TEMPLATE

  sString LEN iLen

RETURNS

  The length of the string.

PARAMETERS
```

```
  sString - string whose length is to be measured

DESCRIPTION

  Returns the length of a string.
```

1.33 ncpy

```
WORD

  NCPY

TEMPLATE

  sDest sSource iCount NCPY

PARAMETERS

  sDest - destination string
  sSource - source string
  iCount - maximum number of characters to copy

DESCRIPTION

  Copies the first iCount characters of sSource to sDest.
  If iCount is less than or equal to the length of sSource, a null
  character is not appended to the copied string.
  If iCount is greater than the length of sSource, the destination
  string is padded with null characters.

--------------------------------------------------
```

1.34 ncat

```
WORD

  NCAT

TEMPLATE

  sDest sSource iCount NCAT

PARAMETERS

  sDest - destination string
  sSource - source string
  iCount - maximum number of characters to append

DESCRIPTION

  Appends, at most, the first iCount characters of sSource to sDest.
  The initial character of sSource overwrites the terminating null
```

```
  character of sDest. If a null character appears in sSource before iCount
  characters are appended, NCAT appends all characters from sSource
  up to the terminating null character. If iCount is greater than
  the length of sSource, the length of sSource is used in place of
  iCount. The resulting string is terminated with a null character.
```

1.35 noise

```
WORD

  NOISE

TEMPLATE

  vPos fMin fMax fPower iOctaves NOISE vResult

PARAMETERS

  vPos - A parameter value for the noise function
  fMin, fMax - Clipping interval.
  fPower - Defines variation speed. MUST be greater than 0.
  iOctaves - Level of detail of the noise. Must be greater than 0.

RETURNS

  vResult - A noise vector

DESCRIPTION

  A multi-purpose three dimensional noise function, which computes
  continouously and randomly varying vector value from the parameter
  vector. The parameter vPos can be any point in space. The result
  vector's x,y and z coordinate component values are always inside the
  range [0,1].

  The 'Octaves' parameter defines the level of detail i.e. how many
  iterations of fractal noise is included in the result.

  The fMin, fMax and fPower parameters define an output value filter
  for the result. The initial coordinate values of the computed noise
  are clipped to the interval defined by fMin and fMax and then rescaled
  back to the [0,1] interval. In practise this means that the smaller the
  difference between fMin and fMax, the larger constant value areas
  are created. Finally, the noise result components are raised to the
  power 'fPower'. The bigger the value, the more rapidly noise values
  change from one extreme to the other.

  Good default values are for example:
  - Octaves = 3
  - fMin = 0, fMax = 1 (no clipping) or fMin = 0.3, fMax = 0.7 (some clipping)
  - fPower = 1 (Linear variation speed) or fPower = 4 (rapid variations)
```

1.36 o_makename

```
WORD

  O_MAKENAME

TEMPLATE

  aObj sName O_MAKENAME

DESCRIPTION

  Builds up a string consisting of a full object path name from the given
  object.

EXAMPLE

  ( print the name of the first selected object

  400 STRING sName

  O_GETSEL sName O_MAKENAME
  sName PUTS

SEE ALSO

  O_FIND
```

1.37 prj_sethook

```
WORD

  PRJ_SETHOOK

TEMPLATE

  aHook PRJ_SETHOOK

PARAMETERS

  aHook   - address of the RPL hook word

DESCRIPTION

  This words allows you to read and process input plane 3d coordinates
  directly.

  The word sets a callback hook for all view windows. The hook procedure
  gets notification messages when the mouse is moved over any active
  view window or when a mouse button is pressed or released
  over a view window. The user must remove the hook when no longer
  needed. To remove a callback hook, call PRJ_SETHOOK with 0 as the
  callback address, i.e.
      0 PRJ_SETHOOK
```

```
EXAMPLE

  "ui.rpl" LOAD
  100 STRING sBuf

  : cbViewWindow
      PARAM
          FVARIABLE PosX
          FVARIABLE PosY
          FVARIABLE PosZ
          VARIABLE iEvent
      ENDPARAM

      UIWM_Move iEvent FETCH = IF
          PosX FFETCH PosY FFETCH PosZ FFETCH
              "View mouse pos %g %g %g" sBuf SPRINTF
          sBuf PUTS
      ENDIF

      UIWM_LMBDown iEvent FETCH = IF
          "View mouse button clicked" sBuf CPY
          sBuf PUTS
      ENDIF

      UIWM_LMBUp iEvent FETCH = IF
          "View mouse button released" sBuf CPY
          sBuf PUTS
      ENDIF
  ;

  ( set callback hook for view windows

  & cbViewWindow PRJ_SETHOOK
```

1.38 remove

```
WORD

  REMOVE

TEMPLATE

  sFileName REMOVE iSuccess

RETURNS

  0 if REMOVE succeeds, -1 otherwise

PARAMETERS

  sFileName - name of the file to be removed

DESCRIPTION
```

```
  Deletes the file specified by sFileName
```

1.39 scandir

```
WORD

  SCANDIR

TEMPLATE

  sPaths sCallBack aData SCANDIR

PARAMETERS

  sPaths - paths to be scanned
  sCallBack - name of the RPL word to be called for each file found
  aData - any user specified data to be forwarded to the RPL callback

DESCRIPTION

  Scans the directories given in the sPaths parameter. For every
  file found the sCallBack RPL word is called.
  The directories in the sPaths parameter are separated by a semicolor (;).
  The callback function should leave a positive value on the stack to
  continue scanning. A value of 0 or a negative value terminates scanning.
  The callback function has two parameters, full path name of the file
  and the user specified data.

EXAMPLE

  VARIABLE count

  : MyCallBack
      DUP FETCH 1 + SWAP STORE    ( add 1 to count )
      "File is " PUTS PUTS 10 13 EMIT EMIT
    1                             ( go on scanning )
  ;

  "macro;rpl" "MyCallBack" count SCANDIR
  count FETCH .
```

1.40 ui_window

```
WORD

  UI_WINDOW

TEMPLATE

  ... aCallBack iLeft iTop iWidth iHeight sTitle UI_WINDOW aHnd

RETURNS
```

```
  Handle to the opened window or NULL if failed

FIXED PARAMETERS

  aCallBack - Callback word for the window to be created
  iLeft, iTop - Left top edge for the window
  iWidth, iHeight - Size of the window
  sTitle - String for the window title bar

TAGS
      -

DESCRIPTION

  Opens a window. Currently only the mouse events are reported to the
  callback. The callback takes the following parameters:

      : cbWindow
          PARAM
              VARIABLE iEvent
              VARIABLE iMouseX
              VARIABLE iMouseY
          ENDPARAM
          ...
      ;

      Callbacks must not return any values.

  The iEvent parameter can be one of the following:

      UIWM_Move     ( Mouse moved
      UIWM_LMBDown  ( Left mouse button clicked
      UIWM_LMBUp    ( Left mouse button released

  iMouseX and iMouseY reflect the current position of the mouse.
```

1.41 ui_button

```
WORD

  UI_BUTTON

TEMPLATE

  ... aWindow aCallback aLeft aTop aWidth aHeight sText UI_BUTTON aHnd

RETURNS

  Pointer to the created button or NULL.

FIXED PARAMETERS

  aWindow - Pointer to the window created by using UI_WINDOW
```

```
  aCallBack - Word which is called when the user clicks the button
  iLeft, iTop - Left-top edge of the button in a given window
  iWidth, iHeight - Size of the button.
  sText - Text for the button.

TAGS

  UI_Disabled - 1 = disabled, 0 = enabled

DESCRIPTION

  Creates a button in the given window. When the user clicks the button,
  the word pointed by aCallBack is called. The callback doesn't take any
  parameters

  : cButton
      "Button Clicked" PUTS
  ;
```

1.42 ui_checkbox

```
WORD

  UI_CHECKBOX

TEMPLATE

  ... aWindow aCallBack iLeft iTop iWidth iHeight sText UI_CHECKBOX aHnd

RETURNS

  Handle to the created checkbox

FIXED PARAMETERS

  See UI_BUTTON

TAGS

  UI_Disabled - 1 = disabled, 0 = enabled
  UICB_Checked - 1 or 0 depending on the desired state of the check box.

DESCRIPTION

  Creates a checkbox. The default state is 'not checked'.

  The callback is called with one parameter which defines whether or not
  the checkbox was set or reset:

  : cbChecBox
      IF
          "Checked" PUTS
      ELSE
          "Not checked" PUTS
      ENDIF
```

```
  ;
```

1.43 ui_string

```
WORD

  UI_STRING

TEMPLATE

  ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_STRING aHnd

RETURNS

  Address of the created string gadget or NULL if failed

FIXED PARAMETERS

  See UI_BUTTON

TAGS

  UI_Disabled - 1 = disabled, 0 = enabled
  UIST_String - String for the edit control

DESCRIPTION

  Creates a string gadget (edit control). When the callback word is called
  depends on the platform. In Amiga, the callback occurs when the user
  pressed either Enter or Tab in the string gadget. In Windows, the
  callback occurs when the user leaves the edit control (either by using
  the Tab key or by activating another control by using the mouse).

  : cbString
      "The user entered the string: %s" PUTS
  ;
```

1.44 ui_text

```
WORD

  UI_TEXT

TEMPLATE

  ... aWindow aCallBack iLeft iTop iWidth iHeight sText UI_TEXT aHnd

RETURNS

  Pointer to the created gadget

FIXED PARAMETERS
```

```
  See UI_BUTTON.

TAGS

  UITX_Text - Text to be displayed
  UITX_Border - If 1, the text will be surrounded by a frame

DESCRIPTION

  Creates a 'read-only' text gadget.

NOTE

  Because this creates a 'read-only' gadget, the parameter 'aCallback' is
  unused and can be NULL. Also the general purpose tag UI_Disabled has
  no effect.
```

1.45 ui_slider

```
WORD

  UI_SLIDER

TEMPLATE

  ... aWindow aCallBack iLeft iTop iWidth iHeight sText UI_SLIDER aHnd

RETURNS

  Pointer to the created slider

FIXED PARAMETERS

  See UI_BUTTON

TAGS

  UI_Disabled  - 1 = Disabled 0 on enabled
  UISL_Min     - Minimun value
  UISL_Max     - Maximum value
  UISL_Level   - Current value


DESCRIPTION

  Creates a slider. When the user plays with the slider, the callback is
  called with one parameter reflecting the current level of the slider.

  : cbSlider
      "Current level=%d" PUTS
  ;
```

1.46 ui_mx

```
WORD

  UI_MX

TEMPLATE

  ... aWindow aCallback iLeft iTop iWidth iHeight sText UI_MX aHnd

RETURNS

  Address of the created gadget

FIXED PARAMETERS

  See UI_BUTTON

TAGS

  UIMX_Active - Currently active item
  UIMX_Labels - Pointer array of strings terminated with null. For more
                information about how to create string pointer arrays
                using RPL, see rpl/ui.rpl.

DESCRIPTION

  Creates a 'Mutual Exclusive' gadget. The gadget consists of two or more
  choices and the user can select only one item at a time. In Amiga,
  the Cycle gadget is used. The Windows version uses 'Combo Box'.

  The callback is called with one parameter which defines the ordinal number
  of the selected object.

  : cbMx
      "You selected the item %d" PUTS
  ;
```

1.47 ui_delete

```
WORD

  UI_DELETE

TEMPLATE

  aHnd UI_DELETE

RETURNS

  -

FIXED PARAMETERS
```

```
  aHnd - Address of the window to be deleted (closed)

TAGS

  None

DESCRIPTION

  Deletes the given window. All the gadgets created on the window will
  automatically be deleted as well.
```

1.48 ui_setattrs

```
WORD

  UI_SETATTRS

TEMPLATE

  .... aHnd UI_SETATTRS

RETURNS

  -

FIXED PARAMETERS

  aHnd - Handle to the gadget to be manipulated

TAGS

  Possible tags depend on the type of the gadget.

DESCRIPTION

  This word is used for setting window and gadget attributes. The parameter
  list consist of a number of tag-value pairs defining attributes and new
  values for them.

  For example, to change the current level of the slider to 10:

      UI_Done
      20  UISL_Level
      aMySlider FETCH UI_SETATTRS
```

1.49 ui_getattrs

```
WORD

  UI_GETATTRS
```

```
TEMPLATE

  ... aHnd UI_GETATTRS

RETURNS

  -

FIXED PARAMETERS

  aHnd - Handle to the gadget to be read

TAGS

  Tags depend on the gadget in question

DESCRIPTION

  This word can be used for reading attributes from the display elements.
  For example, to read the current value of the slider:

      VARIABLE aStrPtr

      UI_Done
      aStrPtr UIST_String
      UI_GETATTRS

      aStrPtr FETCH "The user typed: %s" PUTS
```

1.50 ui_realize

```
WORD

  UI_REALIZE

TEMPLATE

  aHnd UI_REALIZE

RETURNS

  -

FIXED PARAMETERS

  aHnd - Handle to a window to be realized

TAGS

  None

DESCRIPTION

  Displays all the created gadgets in the given window. The following
```

```
  example shows the basic structure of all GUI oriented programs:

  ( create a window
  UI_Done ... UI_WINDOW aWnd STORE

  ( create number of gadgets
  UI_Done ... aWnd FETCH UI_BUTTON aButton STORE
  UI_Done ... aWnd FETCH UI_....
  ...

  ( Now the window is complete and must be realized
  aWnd FETCH UI_REALIZE

  A window needs to be realized only once.
```

1.51 (

```
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
```

1.52)

```
WORD

  )

TEMPLATE

  )

DESCRIPTION
```

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

EXAMPLE

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

1.53 .

```
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.
```

1.54 .s

```
WORD

  .S

TEMPLATE

  .S

DESCRIPTION

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

1.55 store

```
WORD

  STORE

TEMPLATE

  iValue aVariable STORE

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 FSTORE or FFETCH, but only
  with STORE and FETCH. This is because the internal representations for
  similar integer and floating-point values are different.

EXAMPLE

  VARIABLE MyVar  ( define an integer variable )
  3 MyVar STORE

ALSO SEE

 FETCH        VARIABLE
```

1.56 &

```
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
```

1.57 +

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

1.58 -

```
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 - .
```

1.59 *

```
WORD

  *

TEMPLATE

  i2 i1 * iResult

DESCRIPTION
```

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

EXAMPLE

  3 4 * .
```

1.60 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 / .
```

1.61 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

 ;
```

1.62 ;

```
WORD

  ;

TEMPLATE

  ;

DESCRIPTION

  Ends a word definition.

ALSO SEE

 :
```

1.63 <

```
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.
```

1.64 <=

```
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.
```

1.65 <>

```
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
```

1.66 =

```
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
```

1.67 >

```
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
```

1.68 >=

```
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
```

1.69 >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>
```

1.70 >rad

```
WORD

  >RAD

TEMPLATE

  fDeg >RAD fRad

DESCRIPTION

  Converts a value given in degrees to radians.

EXAMPLE

  180.0 >RAD F. ( 3.141593
```

1.71 ?&

```
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
```

1.72 ?dup

```
WORD

  ?DUP

TEMPLATE

  e ?DUP e e

DESCRIPTION

  Duplicates the stack top value if it is not zero.

ALSO SEE

 ?DUP
```

1.73 ?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
```

1.74 ?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
```

1.75 ?if

```
WORD

  ?IF

TEMPLATE

  l ?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
```

1.76 fetch

```
WORD

  FETCH

TEMPLATE

  aInteger FETCH 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 FETCH 1 + MyVar FETCH STORE
```

```
ALSO SEE

 STORE        VARIABLE
```

1.77 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
```

1.78 and

```
WORD

  AND

TEMPLATE

  l2 l1 AND l

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
```

1.79 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
```

1.80 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
```

1.81 atan

```
WORD
  ATAN
TEMPLATE
  fRad ATAN fAng
RETURNS
  fAng - value from -PI/2 to PI/2
DESCRIPTION
  Arctangent function.
ALSO SEE
 ASIN         ACOS         TAN
```

1.82 b.

```
WORD
  B.
TEMPLATE
  i B.
DESCRIPTION
  Takes an integer off the stack and prints it as a binary number.
EXAMPLE
```

```
  1 B.  ( 00000000000000000000000000000001
  2 B.  ( 00000000000000000000000000000010
  3 B.  ( 00000000000000000000000000000011
  4 B.  ( 00000000000000000000000000000100

ALSO SEE

 .            H.            O.
```

1.83 bstore

```
WORD

  BSTORE

TEMPLATE

  b aByte BSTORE

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

 WSTORE   WFETCH   BFETCH
```

1.84 bfetch

```
WORD

  BFETCH

TEMPLATE

  aByte BFETCH 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 BSTORE.
```

```
ALSO SEE

 WSTORE   WFETCH   BSTORE
```

1.85 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
```

1.86 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
```

1.87 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
```

1.88 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
```

1.89 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
```

1.90 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
```

1.91 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
```

1.92 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
```

1.93 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
```

1.94 depth

```
WORD

  DEPTH

TEMPLATE

  DEPTH i

DESCRIPTION

  Puts the count of the stack items onto the stack.

ALSO SEE

 RDEPTH
```

1.95 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
```

1.96 drop

```
WORD

  DROP

TEMPLATE

  e DROP

DESCRIPTION

  Removes the top value from the stack.
```

1.97 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
```

1.98 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
```

1.99 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
```

1.100 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
```

1.101 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 FETCH IF
    "Return value is not zero" ERROR
  ENDIF
```

1.102 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
aadepth   - 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 FSTORE
  "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 )
```

1.103 execute

```
WORD

  EXECUTE

TEMPLATE

  aCFA EXECUTE
```

```
DESCRIPTION

  Executes a word given it's  CFA .

ALSO SEE

 &            ?&
```

1.104 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
```

1.105 exp

```
WORD
```

```
  EXP

TEMPLATE

  fPar EXP fRet

DESCRIPTION

  Raises the natural logarithm base E to the fPar power.
```

1.106 f.

```
WORD

  F.

TEMPLATE

  f F.


DESCRIPTION

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

ALSO SEE

 .            H.           O.           B.
```

1.107 fstore

```
WORD

  FSTORE

TEMPLATE

  f aFloat FSTORE

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 STORE

ALSO SEE

 FFETCH        FVARIABLE
```

1.108 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.
```

1.109 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.
```

1.110 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.
```

1.111 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.
```

1.112 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.
```

1.113 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 or equal to the first value, F<= puts
  TRUE on the stack, otherwise FALSE is put on the stack.
```

1.114 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 not equal to the first value, F<> puts TRUE on
  the stack, otherwise FALSE is put on the stack.
```

1.115 f=

```
WORD

  F=

TEMPLATE

  f2 f1 F= l

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.
```

1.116 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 greater than the first value, F> puts TRUE on
  the stack, otherwise FALSE is put on the stack.
```

1.117 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 greater than or equal to the first value, F>=
  puts TRUE on the stack, otherwise FALSE is put on the stack.
```

1.118 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
```

1.119 ffetch

```
WORD

  FFETCH

TEMPLATE

  aFloat FFETCH 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 STORE

ALSO SEE

 FSTORE       FVARIABLE
```

1.120 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
```

1.121 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.
```

1.122 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.

1.123 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

FSTORE FFETCH FCONSTANT VARIABLE CONSTANT

1.124 h.

WORD

H.

TEMPLATE

i H.

DESCRIPTION

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

ALSO SEE

. O. B.

1.125 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

1.126 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

1.127 if

WORD

IF

```
TEMPLATE

  l 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
```

1.128 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
```

1.129 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
```

1.130 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
```

1.131 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.
```

1.132 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
```

1.133 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
```

1.134 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
```

1.135 +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
```

1.136 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.
```

1.137 not

```
WORD

  NOT

TEMPLATE

  l1 NOT l2

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
```

1.138 o.

```
WORD

  O.

TEMPLATE

  i O.

DESCRIPTION

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

ALSO SEE
```

```
 .            H.            B.
```

1.139 or

```
WORD

  OR

TEMPLATE

  l2 l1 OR l

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
```

1.140 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
```

1.141 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
```

1.142 pow

```
WORD

  POW

TEMPLATE

  fPow fVal POW fRet

DESCRIPTION

  Raises fVal to the fPow power.

EXAMPLE

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

1.143 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
```

1.144 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
```

1.145 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
```

1.146 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
```

1.147 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.
```

1.148 rdepth

```
WORD

  RDEPTH

TEMPLATE

  RDEPTH i

DESCRIPTION

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

ALSO SEE

 DEPTH
```

1.149 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
```

1.150 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
```

1.151 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
```

1.152 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
```

1.153 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
```

1.154 sqrt

```
WORD

  SQRT

TEMPLATE

  fVal SQRT fRet

PARAMETER

  fVal - Positive value

DESCRIPTION

  Squareroot of fVal.

EXAMPLE

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

1.155 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
```

1.156 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
```

1.157 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
```

1.158 tan

```
WORD

  TAN

TEMPLATE

  f1 TAN f2

DESCRIPTION
```

```
  Tangent function

ALSO SEE

 SIN          COS
```

1.159 until

```
WORD

  UNTIL

TEMPLATE

  l 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
```

1.160 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

 STORE        FETCH        CONSTANT     FVARIABLE    FCONSTANT
```

1.161 vlist

```
WORD

  VLIST

TEMPLATE

  VLIST

DESCRIPTION

  Lists the names of all words on Vocabulary Stack.
```

1.162 wstore

```
WORD

  WSTORE

TEMPLATE

  w aWord WSTORE

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

 WFETCH BFETCH BSTORE

1.163 wfetch

WORD

  WFETCH

TEMPLATE

  aWord WFETCH 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 WSTORE .

ALSO SEE

 WSTORE BFETCH BSTORE

1.164 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
```

1.165 xor

```
WORD

  XOR

TEMPLATE

  l2 l1 XOR l

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
```

1.166 c_aimpoint

```
WORD

  C_AIMPOINT

TEMPLATE

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

1.167 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 )
  lOF_TEXTURE    ( Attrib )
  "CEND"         ( Tags )
  "wood" "SMAT"
  C_ATTRIB DROP
```

1.168 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
```

1.169 c_coordsys

```
WORD

  C_COORDSYS

TEMPLATE

  Vorigin
  Vx
  Vy
  Vz
  bR bG bB bA
  sName
  iAttr
  TagList
  C_COORDSYS aObject
```

1.170 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
```

1.171 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 )
  lOF_SECTOR     ( attr: sector )
  "CEND"
```

```
  C_CUTCONE DROP
```

1.172 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.
```

1.173 c_cutpyramid

```
WORD

  C_CUTPYRAMID

TEMPLATE

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

1.174 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
```

1.175 c_ellipse

```
WORD

  C_ELLIPSE

TEMPLATE

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

1.176 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
```

1.177 c_ellipsoid

```
WORD

  C_ELLIPSOID

TEMPLATE

  Vcentre
  Va
  Vb
  Vaxis
  bR bG bB bA
  sName
  iAttr
  TagList
  C_ELLIPSOID aObject
```

1.178 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 )
  lOF_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
```

1.179 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
```

1.180 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
```

1.181 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.
```

1.182 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
```

1.183 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
```

1.184 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 )
  lOF_LIGHTSOURCE lOF_SHADOWLESS BOR
  "CEND"
  C_OFFSET DROP
```

1.185 c_polygon

```
WORD

  C_POLYGON

TEMPLATE

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

```
  TagList
  C_POLYGON aObject
```

1.186 c_polyhedron

```
WORD

  C_POLYHEDRON

TEMPLATE

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

1.187 c_polymid

```
WORD

  C_POLYMID

TEMPLATE

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

1.188 c_pyramid

```
WORD

  C_PYRAMID

TEMPLATE
```

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

1.189 c_rectangle

```
WORD

  C_RECTANGLE

TEMPLATE

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

1.190 c_triset

```
WORD

  C_TRISET

TEMPLATE

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

1.191 c_skeleton

```
WORD

  C_SKELETON

TEMPLATE

  Joint0
  [Joint ..]
  iCount
  iType
  iSkelflags
  bR bG bB bA
  sName
  iAttr
  TagList
  C_SKELETON aObject

PARAMETERS

  iCount    - Number of joints in the skeleton
  iType     - 0 = a freely moving skeleton, 1 = a fixed start point
  iSkelflags - unused, set to zero
  Jointn    - Joints for the skeleton

  Each joint consists of the following parameters:

  sName
  vPos
  vDir
  fLen
  fFric
  fFid
  fAS fAE
  fBS fBE
  fCS fCE
```

```
  sName - Name of the joint
  vPos  - Position of the joint
  vDir  - Normal vector for the joint opening plane
  fLen  - Length of the bone ( unused, set to zero )
  fFric - Friction value (0 ... 1.0)
  fFid  - Fidelity value (0 ... 1.0)
  fAS - fCE - Constraint angles in radians ( CS & CE unused currently )

DESCRIPTION

  Creates a skeleton primitive.
```

1.192 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.
```

1.193 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
```

1.194 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
```

1.195 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
```

1.196 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
```

1.197 m_delete

```
WORD

  M_DELETE

TEMPLATE

  0 aObject1 [aObjectn ..] M_DELETE

DESCRIPTION

  Deletes targets from hierarchy.

EXAMPLE
```

```
  O_GETSEL ( objects )
  M_DELETE
```

1.198 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
```

1.199 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
```

1.200 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
```

1.201 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
```

1.202 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
```

1.203 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
```

1.204 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
```

1.205 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
```

1.206 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
```

1.207 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
```

1.208 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
```

1.209 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
```

1.210 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
```

1.211 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
```

1.212 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
```

1.213 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
```

1.214 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 )
```

1.215 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
```

1.216 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
```

1.217 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
  ;
```

1.218 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
```

1.219 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
```

1.220 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
```

1.221 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.
```

1.222 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.
```

1.223 o_getsel

```
WORD

  O_GETSEL

TEMPLATE

  O_GETSEL 0 aObject1 [aOjectn ..]

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
```

1.224 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.
```

1.225 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
```

1.226 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.
  ;
```

1.227 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
```

1.228 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
```

1.229 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
```

1.230 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
```

1.231 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 XXXXXXX

  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 FETCH ( 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
```

1.232 mth_delete

```
WORD

  MTH_DELETE

TEMPLATE

  aMthAddr MTH_DELETE

DESCRIPTION

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

1.233 mth_find

```
WORK

  MTH_FIND

TEMPLATE

  sName MTH_FIND aMthAddr

DESCRIPTION

  Returns the address of the method given its name.
```

1.234 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
```

1.235 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.
```

1.236 mat_create

```
WORD

  MAT_CREATE

TEMPLATE

  sName
  wSpecularity
  wSpecBright
  wBrilliance
  wTransparancy
  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.
```

1.237 mat_delete

```
WORD

  MAT_DELETE

TEMPLATE

  aMaterial MAT_DELETE

DESCRIPTION

  Deletes the material pointed by aMaterial from the material
  library.
```

1.238 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.
```

1.239 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.
```

1.240 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 STORE

    ( 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 FETCH 512 MEM_FREE
  ;


  DoSomething
```

1.241 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

  : MyErrHandler
    ( .... )
  ;


  & MyErrHandler ERR_INSTALL
  ( .... )
  & MyErrHandler ERR_REMOVE
```

1.242 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 STORE
    aMyStr FETCH 63 "Enter any string" GET_STR
    IF
      aMyStr FETCH PUTS
    ENDIF
    aMyStr FETCH 64 MEM_FREE
  ;
```

1.243 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
```

1.244 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
```

1.245 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
```

1.246 refresh

```
WORD

  REFRESH

TEMPLATE

  REFRESH

DESCRIPTION

  Refreshes all windows using their individual refresh settings.
```

1.247 render

```
WORD

  RENDER

TEMPLATE

  RENDER

DESCRIPTION

  Renders all Views using their individual render settings
```

1.248 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.
```

1.249 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
```

1.250 system

```
WORD

  SYSTEM

TEMPLATE

  sCLI SYSTEM
```

```
DESCRIPTION

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

EXAMPLE

  "SYS:Tools/IconEdit" SYSTEM
```

1.251 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.
```

1.252 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
```

1.253 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_SENDMSG DROP

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

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

1.254 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 STORE

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

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

  ( free intersection handle )
  iHnd FETCH RAY_FREE
;
```

ALSO SEE

```
 RAY_PREP      RAY_FREE
```

1.255 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 STORE

  ....

  RayHandle FETCH RAY_FREE

ALSO SEE

 RAY_INTERS    RAY_PREP
```

1.256 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
```

1.257 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 FETCH "Rendering..."
      I BUSY_UPDATE ( Rendering )
        aBusyHnd FETCH BUSY_CANCEL
      IF
        LEAVE
      ENDIF
    LOOP
    aBusyHnd FETCH BUSY_CLOSE
  ;
```

1.258 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 FETCH BUSY_CLOSE
```

1.259 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 STORE
    100 0 DO
      O_GETSEL
      0.1 0 0 0 M_MOVE
      REFRESH
      aHnd FETCH 0 I BUSY_UPDATE
    LOOP
    aHnd FETCH BUSY_CLOSE
  ;
```

1.260 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 STORE
    100 0
      aBusyHnd FETCH 0 I
      BUSY_UPDATE ( Optimizing )
```

```
        LOOP
        100 0 DO
      aBusyHnd FETCH "Rendering..." I
      BUSY_UPDATE ( Rendering )
        LOOP
    aBusyHnd FETCH BUSY_CLOSE
  ;
```

1.261 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 .
```

1.262 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 FSTORE
    MyFlt "Define Angle" GET_FLT
    IF
      MyFlt FFETCH F.
    ENDIF
  ;

  MyTest
```

1.263 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
  ;
```

1.264 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

   VVARIABLE vDelta

   : MyTest
     0.0 1.0 3.1 vDelta VSTORE
     vDelta "Move Selected Objects" GET_VECT
     IF
       O_GETSEL
       vDelta FFETCH 0 M_MOVE
     ENDIF
```

```
  ;
```

1.265 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
    "r3d3:projects" Path CPY

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

```
  MyLoad
```

1.266 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
```

1.267 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   ( 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.
```

1.268 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   ( fetch return value )
  IF
    "ERROR:Cannot Execute" ERROR
  ENDIF
  RX_RESULT PUTS ( process result string )
```

1.269 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
```

1.270 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

 VFETCH            VSTORE
```

1.271 vstore

```
WORD

  VSTORE

TEMPLATE

  fX fY fZ aVariable VSTORE

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 VSTORE
```

1.272 vfetch

```
WORD

  VFETCH

TEMPLATE

  aVariable VFETCH 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 VFETCH V.
```

1.273 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

  VVARIABLE v1 VVARIABLE v2 VVARIABLE vRes

  1 0 0 v1 VSTORE       ( v1 = 1 0 0 )
  0 1 0 v2 VSTORE       ( v2 = 0 1 0 )
  v1 VFETCH v2 VFETCH VADD vRes VSTORE ( vRes = v1 + v2 )
```

1.274 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.
```

1.275 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.
```

1.276 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.
```

1.277 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 )
```

1.278 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
```

1.279 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.
```

1.280 v.

```
WORD

  V.

TEMPLATE

  v V.

PARAMETERS

  v - vector

DESCRIPTION

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

1.281 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.
```

1.282 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.

1.283 [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 sectored:

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

1.284 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.

1.285 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
```

1.286 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 )
```

1.287 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
        ------------------------------------------------------------------
        lOF_INVERTED     Volume inverted in Boolean Operations
        lOF_PAINTED      Surface properties affects in Booleans
        lOF_WFINVISIBLE  Wire frame is invisible
        lOF_LIGHTSOURCE  Object is a light source
        lOF_HOLLOW       Represented as a surface instead of solid object
        lOF_INFINITE     Infinite object
        lOF_SCENE        Invisible in primary ray tracing
        lOF_RTINVISIBLE  Ray tracing invisible
        lOF_NOBP1        No 1st. bounding plane
        lOF_NOBP2        No 2nd bounding plane
```

```
        lOF_TEXTURE      Primitive used for mapping textures to objects
        lOF_SECTOR       Sector Primitive
        lOF_PROTECTED    Primitive cannot be modified
        lOF_SEGMENT      Segment instead of sector
        lOF_NOTREFL      Not reflected
        lOF_MOTION       Motion Blurred object
        lOF_SHADOWLESS   Does not Cast Shadows
        lOF_MATTE        Matte Object
        lOF_SECTOR       Sector primitive
        lOF_SEGMENT      Segment primitive
```

1.288 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.
```

1.289 modify flags

```
MODIFY FLAGS


FLAG      DESCRIPTION
-----------------------------------------------------

lMF_ROTEXT           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
```

1.290 getvstack

```
WORD

    GETVSTACK

SYNTAX

    iCnt GETVSTACK iActual ... vP1

PARAMETERS
```

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

RESULT

    iActual - Actual number of items fetched.
    ... vP1 - 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
    ;
```

1.291 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.

1.292 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.

1.293 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

1.294 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

1.295 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
```