C
Language Reference Booklet

RT PC Graphics Development Toolkit

Programming Family

Personal
Computer
Software

59X8613

IBM RT PC C Language Reference Booklet

This reference booklet describes the IBM RT PC C language syntax required to write an application program using the Graphics Development Toolkit. Any general or special considerations for the C language are described in this booklet. It is important that you keep this booklet in a safe place. This booklet is the only source of information that specifically describes the IBM RT PC C language interface to the Toolkit.

Writing in C

Writing in the C language requires a few special considerations. They are as follows:

- **Array numbering.** Array tables in the *IBM RT PC Graphics Development Toolkit* show the array index 1 to n. A C language array index is numbered 0 to n–1. All array indexes are 1 less in C than the Graphics Development Toolkit array index.

  For some routines the size of an array for one parameter is based on the value of another parameter. This is shown in the "Data Types:" by using the parameter name in the array size. This notation is used only to show the relationship between the parameters and does not imply actual coded values or refer to storage allocation.

- **Data types.** All integer variables are INT16 (signed 16-bit integer). INT16 is defined as follows:

  ```
  typedef short INT16;
  ```

  When a parameter listed in "Data Types:" is preceded by an asterisk (*), it indicates that the number is a pointer to a memory location.

  All character strings are defined as **char** (unsigned 8-bit character). This is an example of char that defines the variable "char_string" to be 128 bytes long.

  ```
  char char_string[128];
  ```

  Scalar input parameters are passed by value; array input parameters are passed by reference. All output parameters are passed by reference.

- **Routine calls.** The call to a C Toolkit routine takes the following form:

  status=vname(a, b, c);

  Where: status=status returned
  vname=subroutine name
  a, b, and c=parameters

  Since all Toolkit routine names return an INT16 value that is the status assigned by the routine, all routines must be declared as INT16. Unless otherwise indicated, a value of zero indicates successful completion and a value of minus one indicates an error has occurred.

- **Compiling and Linking.** Compile your program and link it to the Graphics Development Toolkit subroutine library with the following command:

  ```
  cc filename.c /usr/lpp/vdi/lib/cvdi.a -o
   filename
  ```

Programming Considerations

In a situation where both Toolkit routines and C statements are available to perform the same operations, use the Toolkit routine to ensure that you get correct results.

C Language Syntax

All parameters are of INT16 type unless defined in "Data Types:" after the routine syntax description.

A double asterisk (**) following the generic function name indicates the routine is device-dependent. Using these functions in an application program makes that application device-dependent.

Throughout the "Routines" section of this booklet, the input parameters are italicized and the output parameters are shown in regular type.

Routines

Application Data**
status = v_appl
(*handle, function, data_count, appli_data*);

Data Types: char function[];
INT16 appli_data[data_count];

Clear Workstation
status = v_clrwk
(*handle*);

Close Workstation
status = v_clswk
(*handle*);

Copy Page**
status = vc_page
(*handle, source, destination*);

Copy Pels**
status = v_copy_pels
(*handle, xy*);

Data Types: INT16 xy[6];

Cursor Down**
status = v_curdown
(*handle*);

Cursor Home**
status = v_curhome
(*handle*);

Cursor Left**
status = v_curleft
(*handle*);

Cursor Right**
status = v_curright
(*handle*);

Cursor Up**
status = v_curup
(*handle*);

Direct Cursor Address**
status = vs_curaddress
(*handle, row, column*);

Display Graphic Input Cursor**

status = v_dspcur
(*handle, x, y*);

Enter Cursor Addressing Mode**

status = v_enter_cur
(*handle*);

Erase to End of Line**

status = v_eeol
(*handle*);

Erase to End of Screen**

status = v_eeos
(*handle*);

Exit Cursor Addressing Mode**

status = v_exit_cur
(*handle*);

Get Pels**

status = v_get_pels
(*handle, xy,* pel_array);

Data Types: INT16 xy[4];
INT16 pel_array[];

Hardcopy**

status = v_hardcopy
(*handle*);

Input Choice (request mode)

status = vrq_choice
(*handle, initial_choice,* final_choice);

Data Types: INT16 *final_choice;

status = 0 request unsuccessful
> 0 request successful
= -1 an error has occurred

Input Choice (sample mode)

status = vsm_choice
(*handle,* choice);

Data Types: INT16 *choice;

status = 0 sample unsuccessful
> 0 sample successful
= -1 an error has occurred

Input Locator (request mode)

status = vrq_locator
(*handle, initial_xy, ink, rubberband, echo_handle,* final_xy, terminator);

Data Types:	INT16	initial_xy[2];
	INT16	final_xy[2];
	char	*terminator;

status = 0 request unsuccessful
> 0 request successful
= -1 an error has occurred

Input Locator (sample mode)

status = vsm_locator
(*handle, in_xy,* out_xy, pressed, released, key_state);

Data Types:	INT16	in_xy[2];
	INT16	out_xy[2];
	INT16	*pressed;
	INT16	*released;
	INT16	*key_state;

status = 0 sample unsuccessful
> 0 sample successful
= -1 an error has occurred

Input String (request mode)

status = vrq_string
(*handle, maximum_length, echo_mode, echo_xy,* char_string);

Data Types:	INT16	echo_xy[2];
	char	char_string[];

status = 0 request unsuccessful
> 0 number of characters returned
= -1 an error has occurred

Input String (sample mode)

status = vsm_string
(*handle, maximum_length, echo_mode, echo_xy,* char_string);

Data Types:	INT16	echo_xy[2];
	char	char_string[];

status = 0 sample unsuccessful
> 0 number of characters
= -1 an error has occurred

Input Valuator (request mode)

status = vrq_valuator
(*handle, initial_value, echo_handle,* final_value);

Data Types: INT16 *final_value;

status = 0 request unsuccessful
> 0 request successful
= -1 an error has occurred

Input Valuator (sample mode)

status = vsm_valuator
(*handle,* final_value);

Data Types: INT16 *final_value;

status = 0 sample unsuccessful
> 0 sample successful
= -1 an error has occurred

Inquire Addressable Character Cells

status = vq_chcells
(*handle,* rows, columns);

Data Types: INT16 *rows;
INT16 *columns;

Inquire Alpha Text Capabilities

status = vqa_cap
(*handle,* capabilities);

Data Types: INT16 capabilities[15];

Inquire Alpha Text Cell Location

status = vqa_cell
(*handle, row, column,* proportion_flag, x_out, y_out);

Data Types: INT16 *proportion_flag;
INT16 *x_out;
INT16 *y_out;

Inquire Alpha Text Font Capability

status = vqa_font
(*handle, font_number, text_size,* capabilities);

Data Types: INT16 capabilities[7];

status = 0 font unavailable
> 0 font available
= -1 an error has occurred

Inquire Alpha Text Position

status = vqa_position
(*handle*, x_out, y_out);

Data Types: INT16 *x_out;
INT16 *y_out;

Inquire Alpha Text String Length

status = vqa_length
(*handle*, *char_string*);

Data Types: char char_string[];

status ≥ 0 string length
= −1 an error has occurred

Inquire Cell Array

status = vq_cellarray
(*handle*, *xy*, *row_length*, *number_rows*,
elements_per_row, rows_used, vflag, colors);

Data Types: INT16 xy[4];
INT16 *elements_per_row;
INT16 *rows_used;
INT16 *vflag;
INT16 colors[];

Inquire Color Representation

status = vq_color
(*handle*, *color_number*, *set_flag*, rgb_returned);

Data Types: INT16 rgb_returned[3];

status ≥ 0 actual index selected
= −1 an error has occurred

Inquire Current Cursor Text Address**

status = vq_curaddress
(*handle*, row, column);

Data Types: INT16 *row;
INT16 *column;

Inquire Current Fill Area Attributes

status = vqf_attributes
(*handle*, attributes);

Data Types: INT16 attributes[4];

Inquire Current Graphic Text Attributes

```
status=vqt_attributes
    (handle, attributes);

    Data Types    INT16    attributes[10];
```

Inquire Current Polyline Attributes

```
status=vql_attributes
    (handle, attributes);

    Data Types:   INT16    attributes[4];
```

Inquire Current Polymarker Attributes

```
status=vqm_attributes
    (handle, attributes);

    Data Types:   INT16    attributes[4];
```

Inquire Cursor Text Mode**

```
status=vq_curmode
    (handle);

    status  ≥ 0 current mode
            =-1 an error has occurred
```

Inquire Error

```
status=vq_error();
```

Inquire Graphic Color Burst Mode**

```
status=vq_gclbu
    (handle);

    status  ≥ 0 actual mode selected
            =-1 an error has occurred
```

Inquire Page**

```
status=vq_page
    (handle, gr_mode, cur_mode);

    Data Types:   INT16    gr_mode[3];
                  INT16    cur_mode[3];
```

Message**

```
status=v_msg
    (handle, msg, wait_flag);

    Data Types:   char     msg[];
```

Open Workstation

status = v_opnwk
(*work_in*, handle, work_out);

Data Types:	INT16	work_in[19];
	INT16	*handle;
	INT16	work_out[66];

Output Alpha Text

status = v_atext
(*handle, char_string*, x_out, y_out);

Data Types:	char	char_string[];
	INT16	*x_out;
	INT16	*y_out;

Output Arc

status = v_arc
(*handle, x_center, y_center, radius, begin_angle, end_angle*);

Output Bar

status = v_bar
(*handle, xy*);

Data Types:	INT16	xy[4];

Output Cell Array

status = v_cellarray
(*handle, xy, row_length, elements_per_row, number_rows, writing_mode, colors*);

Data Types:	INT16	xy[4];
	INT16	colors[];

Output Circle

status = v_circle
(*handle, x_center, y_center, radius*);

Output Cursor Addressable Text**

status = v_curtext
(*handle, string*);

Data Types:	char	string[];

Output Filled Area

status = v_fillarea
(*handle, count, xy*);

Data Types:	INT16	xy[2*count];

Output Graphic Text

status = v_gtext
(*handle, x, y, char_string*);

Data Types: char char_string[];

Output Pie Slice

status = v_pieslice
(*handle, x_center, y_center, radius, begin_angle, end_angle*);

Output Polyline

status = v_pline
(*handle, count, xy*);

Data Types: INT16 xy[2*count];

Output Polymarker

status = v_pmarker
(*handle, count, xy*);

Data Types: INT16 xy[2*count];

Put Pels**

status = v_put_pels
(*handle, xy, pel_array*);

Data Types: INT16 xy[2];
INT16 pel_array[];

Read Cursor Movement Keys**

status = vrd_curkeys
(*handle, input_mode,* direction, key);

Data Types: INT16 *direction;
char *key;

Remove Graphic Input Cursor**

status = v_rmcur
(*handle*);

Reverse Video Off**

status = v_rvoff
(*handle*);

Reverse Video On**

status = v_rvon
(*handle*);

Set Alpha Text Color Index

status = vsa_color
(*handle, color_number*);

status ≥ 0 index selected
= −1 an error has occurred

Set Alpha Text Font and Size

status = vsa_font
(*handle, font_number, size_number,* capabilities);

Data Types: INT16 capabilities[8];

status = 0 font unavailable
〉 1 font selected
= −1 an error has occurred

Set Alpha Text Line Spacing

status = vsa_spacing
(*handle, spacing_requested*);

status ≥ 0 spacing selected
= −1 an error has occurred

Set Alpha Text Overstrike Mode

status = vsa_overstrike
(*handle, mode_number*);

status ≥ 0 mode selected
= −1 an error has occurred

Set Alpha Text Pass Through Mode

status = vsa_passthru
(*handle, mode_number*);

status ≥ 0 mode selected
= −1 an error has occurred

Set Alpha Text Position

status = vsa_position
(*handle, x_in, y_in,* x_out,y_out);

Data Types: INT16 *x_out;
INT16 *y_out;

Set Alpha Text Quality

status = vsa_quality
(*handle, mode_in*);

status ≥ 0 mode selected
= −1 an error has occurred

Set Alpha Text Subscript Superscript Mode

status = vsa_supersub
(*handle, mode_number*);

status ≥ 0 mode selected
= -1 an error has occurred

Set Alpha Text Underline Mode

status = vsa_underline
(*handle, mode_number*);

status ≥ 0 mode selected
= -1 an error has occurred

Set Background Color Index

status = vsb_color
(*handle, color_number*);

status ≥ 0 index selected
= -1 an error has occurred

Set Character Height

status = vst_height
(*handle, height_requested,* char_width, cell_width,
cell_height);

Data Types: INT16 *char_width;
INT16 *cell_width;
INT16 *cell_height;

status ≥ 0 height selected
= -1 an error has occurred

Set Color Representation

status = vs_color
(*handle, color_number, rgb_input,* rgb_output);

Data Types: INT16 rgb_input[3];
INT16 rgb_output[3];

status ≥ 0 actual index selected
= -1 an error has occurred

Set Cursor Text Attributes**

status = vcur_att
(*handle, req_att,* sel_att);

Data Types: INT16 req_att[4];
INT16 sel_att[4];

Set Cursor Text Color Index**

status = vcur_color
(*handle, fore_requested, back_requested,*
fore_selected, back_selected);

Data Types: INT16 *fore_selected;
INT16 *back_selected;

Set Cursor Text Mode**

status = vs_curmode
(*handle, mode*);

status ≥ 0 mode selected
= −1 an error has occurred

Set Fill Color Index

status = vsf_color
(*handle, index_requested*);

status ≥ 0 color index selected
= −1 an error has occurred

Set Fill Interior Style

status = vsf_interior
(*handle, style_number*);

status ≥ 0 style selected
= −1 an error has occurred

Set Fill Style Index

status = vsf_style
(*handle, style_number*);

status ≥ 0 index selected
= −1 an error has occurred

Set Graphic Color Burst Mode**

status = vs_gclbu
(*handle, mode*);

status ≥ 0 actual mode selected
= −1 an error has occurred

Set Graphic Text Alignment

status = vst_alignment
(*handle, horizontal_requested, vertical_requested,*
horizontal_selected, vertical_selected);

Data Types: INT16 *horizontal_selected;
INT16 *vertical_selected;

Set Graphic Text Color Index

status = vst_color
(*handle, color_number*);

status ≥ 0 index selected
= −1 an error has occurred

Set Graphic Text Font

status = vst_font
(*handle, font_number*);

status ≥ 0 font type selected
= −1 an error has occurred

Set Graphic Text String Baseline Rotation

status = vst_rotation
(*handle, angle_of_rotation*);

status ≥ 0 angle selected
= −1 an error has occurred

Set Line Edit Characters

status = vs_editchars
(*handle, line_del, char_del*);

Data Types: char line_del;
char char_del;

Set Page**

status = vs_page
(*handle, gr_in, cur_in,* gr_out, cur_out);

Data Types: INT16 gr_in[2];
INT16 cur_in[2];
INT16 gr_out[2];
INT16 cur_out[2];

Set Pen Speed**

status = vs_penspeed
(*handle, speed*);

status ≥ 0 actual pen speed
= −1 an error has occurred

Set Polyline Color Index

status = vsl_color
(*handle, color_number*);

status ≥ 0 index selected
= −1 an error has occurred

Set Polyline Line Type

status = vsl_type
(*handle, type_number*);

status ≥ 0 type selected
= −1 an error has occurred

Set Polyline Line Width

status = vsl_width
(*handle, width*);

status ≥ 0 width selected
= −1 an error has occurred

Set Polymarker Color Index

status = vsm_color
(*handle, color_number*);

status ≥ 0 index selected
= −1 an error has occurred

Set Polymarker Height

status = vsm_height
(*handle, marker_height*);

status ≥ 0 height selected
= −1 an error has occurred

Set Polymarker Type

status = vsm_type
(*handle, type_number*);

status ≥ 0 type selected
= −1 an error has occurred

Set Writing Mode

status = vswr_mode
(*handle, mode_number*);

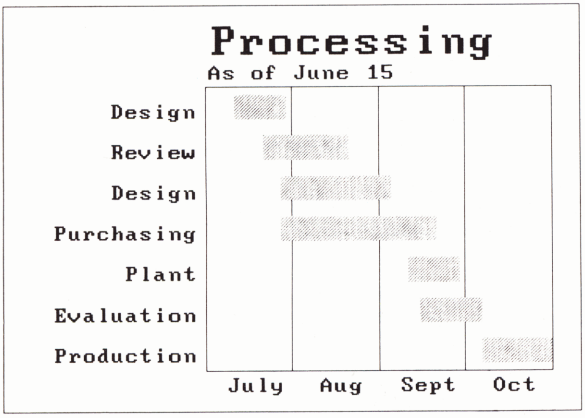
status ≥ 0 actual mode selected
= −1 an error has occurred

Update Workstation

status = v_updwk
(*handle*);

Program Example

This program demonstrates how to create and display a Gantt chart. The output from this program should appear as follows:




```
#include "stdio.h"

typedef short  INT16; /* Define signed 16 bit integer */

main()
{
  static char *tasks[] = {"Production", "Evaluation",
                          "Plant", "Purchasing",
                          "Design", "Review", "Design"};
  static char *title = {"Processing "};
  static char *y_label = {"As of June 15"};
  static char *y_ticks[]= {"July","Aug","Sept","Oct"};

  extern INT16 box();
  extern INT16 fnxtr();
  extern INT16 fnytr();


  INT16 dev_handle, xheight, istring[2],
        gdms_err, xy[10], savary[66],
        xwidth, cwidth, i, j, cheight;

  static INT16 echo_xy[2] = {0,0};
  static INT16 work_in[] = {0, 1, 1, 3, 1, 1, 1, 0,
                            0, 1, 1, 'D', 'I', 'S',
                            'P', 'L', 'A', 'Y', ' '};
  static INT16 start_dates[] = {83, 72, 70, 48,
                                48, 45, 40};
  static INT16 end_dates[] = {95, 83, 79, 75,
                              67, 60, 49};
```

```
/* open the device */
gdms_err = v_opnwk(work_in,&dev_handle,savary);

/* set the constants for the grid */
xy[1] = fnytr(10, savary);
xy[3] = fnytr(80, savary);

for (i=50;i<=80;i+=15)  {
  /* set variable elements in array for grid */
  xy[2] = xy[0] = fnxtr(i, savary); xy[2] = xy[0];

  /* draw the line */
  gdms_err = v_pline(dev_handle,2,xy);
 }

/* set character height */
xheight = vst_height(dev_handle,fnytr(4, savary),
                     &xwidth,&cwidth,&cheight);

/* set text alignment */
gdms_err = vst_alignment(dev_handle,1,2,&i,&j);

/* index into tick labels */
j = 0;

for (i=43;i<=88;i+=15)  {
  /* write text */
  gdms_err = v_gtext(dev_handle, fnxtr(i, savary),
                     fnytr(10, savary), y_ticks[j++]);
 }

/* set text alignment */
gdms_err = vst_alignment(dev_handle,2,1,&i,&j);

j = 0;   /* index into y axis labels */

for (i=15; i<=75; i+=10)  {
  /* write out text */
  gdms_err = v_gtext(dev_handle,fnxtr(33, savary),
                     fnytr(i, savary), tasks[j++]);
 }

/* set text alignment */
gdms_err = vst_alignment(dev_handle,0,0,&i,&j);

/* write out the y axis label */
gdms_err = v_gtext(dev_handle,fnxtr(35, savary),
                   fnytr(82, savary),y_label);

/* set new character height */
gdms_err = vst_height(dev_handle,
                      fnytr(9, savary),
                      &xwidth,
                      &cwidth,
                      &cheight);
```

```
  /* write out title text */
  gdms_err = v_gtext(dev_handle,fnxtr(35, savary),
                     fnytr(88, savary),title);

  /* set fill pattern */
  gdms_err = vsf_style(dev_handle,2);
  gdms_err = vsf_interior(dev_handle,3);

  j = 0;  /* set index into data arrays */

  for (i=12; i<=72; i+=10)  {
    /* set dimensions for bars */
    xy[0] = fnxtr(start_dates[j], savary);
    xy[1] = fnytr(i, savary);
    xy[2] = fnxtr(end_dates[j++], savary);
    xy[3] = fnytr(i+6, savary);

    /* draw the bars */
    gdms_err = v_bar(dev_handle,xy);
   }

  /* create chart frame */
  box(35,95,10,80,xy, savary);
  gdms_err = v_pline(dev_handle,5,xy);

  /* create page border */
  box(0,100,0,100,xy, savary);
  gdms_err = v_pline(dev_handle,5,xy);

  /* wait for user input before returning */
  gdms_err = vrq_string(dev_handle,2,0,echo_xy,istring);

  /* close workstation */
  gdms_err = v_clswk(dev_handle);
 }

INT16 box(xmin,xmax,ymin,ymax,xyout,savary)
  INT16 xmin, xmax, ymin, ymax, xyout[], savary[];
{
  extern INT16 fnxtr();
  extern INT16 fnytr();

  xyout[8] = xyout[6] = xyout[0] = fnxtr(xmin, savary);
  xyout[9] = xyout[3] = xyout[1] = fnytr(ymin, savary);
  xyout[4] = xyout[2] = fnxtr(xmax, savary);
  xyout[7] = xyout[5] = fnytr(ymax, savary);
 }


INT16 fnxtr(percent, savary)
  INT16 percent, *savary;
{
  return((float) percent / 100.0 * savary[51]);
 }
```

```
INT16 fnytr(percent, savary)
  INT16 percent, *savary;
{
  return((float) percent / 100.0 * savary[52]);
 }
```

International Business
Machines Corporation
Dept. 997, Bldg. 998
11400 Burnet Rd.
Austin, Texas 78758

Printed in the
United States of America

59X8613