Pascal
Language Reference Booklet

RT PC Graphics Development Toolkit

Programming Family

Personal
Computer
Software

59X8619

IBM RT PC Pascal Language Reference Booklet

This reference booklet describes the IBM RT PC Pascal syntax required to write an application program using the IBM RT PC Graphics Development Toolkit. Any general or special considerations for the Pascal 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 Pascal language interface to the Toolkit.

Only the native mode of the IBM RT PC Pascal language is supported by the IBM RT PC Graphics Development Toolkit.

Writing in Pascal

Writing in Pascal 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 starting with 1 and continuing to n.

  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 32-bit integers if they are not otherwise declared.

  All character strings are defined as *lstring(n)* with a length of n. Character strings with a single character are defined as *char.*

  The super array data type, named *intary,* has been predeclared in *pasvdi.int.* Do not declare this data type yourself. All integer arrays passed to or received from a Toolkit routine must be of intary type.

  intary has been defined as follows:

```
type intary=super array [1..*] of
integer;
```

An example of declaring a 10 element integer array named array_name for a Toolkit routine follows:

```
var
        array_name  : intary(10);
```

To use Toolkit, begin your program with the lines shown in the following example. The include statement for **pasvdi.int** resolves external reference calls to Toolkit routines. This include statement must appear after the program statement of a program. The following example assumes there is a copy of **pasvdi.int** in the directory with the program. Otherwise, the include statement should specify the pathname, **/usr/include/pasvdi.int**.

```
Program name (input,output ....);

{
Pascal example
}

{ ( $include : 'pasvdi.int' ) }


CONST
        Max=15;
TYPE
        String_one_type=LSTRING(80);
        String_two_type=LSTRING(40);
        Array_type_ten=intary(10);
var
        String_one : String_one_type;
        String_two : String_two_type;
        Array_ten : Array_type_ten;
BEGIN
 your program
  .
  .
  .
END.
```

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

  status: = vname(a, b, c);

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

  All Toolkit routines return an integer value that is the status assigned by the routine. 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 line:

```
pascal filename -M
 /usr/lpp/vdi/lib/pasvdi.a -o filename
```

Programming Considerations

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

Pascal Language Syntax

All parameters that are not defined in "Data Types:" are of integer type.

A double asterisk (**) following the generic routine name indicates that the routine is device-dependent. Using these routines 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_application_data
(*handle, funct, data_count, appli_data*);

Data Types: funct:lstring(n);
appli_data:intary(data_count);

Clear Workstation
status:=v_clear_workstation
(*handle*);

Close Workstation
status:=v_close_workstation
(*handle*);

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

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

Data Types: xy:intary(6);

Cursor Down**
status:=v_cursor_down
(*handle*);

Cursor Home**
status:=v_cursor_home
(*handle*);

Cursor Left**
status:=v_cursor_left
(*handle*);

Cursor Right**
status:=v_cursor_right
(*handle*);

Cursor Up**
status:=v_cursor_up
(*handle*);

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

Display Graphic Input Cursor**

status:=v_display_gin_cursor
(*handle, x, y*);

Enter Cursor Addressing Mode**

status:=v_enter_cursor_mode
(*handle*);

Erase to End of Line**

status:=v_erase_end_of_line
(*handle*);

Erase to End of Screen**

status:=v_erase_end_of_screen
(*handle*);

Exit Cursor Addressing Mode**

status:=v_exit_cursor_mode
(*handle*);

Get Pels**

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

Data Types: xy:intary(4);
pel_array:intary(n);

Hardcopy**

status:=v_hardcopy
(*handle*);

Input Choice (request mode)

status:=v_request_choice
(*handle, initial_choice,* final_choice);

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

Input Choice (sample mode)

status:=v_sample_choice
(*handle,* choice);

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

Input Locator (request mode)

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

Data Types: initial_xy:intary(2);
final_xy:intary(2);
terminator:char;

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

Input Locator (sample mode)

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

Data Types: in_xy:intary(2);
out_xy:intary(2);

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

Input String (request mode)

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

Data Types: echo_xy:intary(2);
char_string:lstring(n);

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

Input String (sample mode)

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

Data Types: echo_xy:intary(2);
char_string:lstring(n);

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

Input Valuator (request mode)

status:=v_request_valuator
(*handle, initial_value, echo_handle,* final_value);

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

Input Valuator (sample mode)
status:=v_sample_valuator
(*handle*, final_value);

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

Inquire Addressable Character Cells
status:=v_inq_char_cells
(*handle*, rows, columns);

Inquire Alpha Text Capabilities
status:=v_inq_alpha_text_capabilities
(*handle*, capabilities);

Data Types: capabilities:intary(15);

Inquire Alpha Text Cell Location
status:=v_inq_alpha_cell_location
(*handle, row, column*, proportion_flag, x_out, y_out);

Inquire Alpha Text Font Capability
status:=v_inq_alpha_font_availability
(*handle, font_number, text_size*, capabilities);

Data Types: capabilities:intary(7);

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

Inquire Alpha Text Position
status:=v_inq_alpha_position
(*handle*, x_out, y_out);

Inquire Alpha Text String Length
status:=v_inq_alpha_string_length
(*handle, char_string*);

Data Types: char_string:lstring(n);

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

Inquire Cell Array
status:=v_inq_cell_array
(*handle, xy, row_length, number_rows,*
elements_per_row, rows_used, vflag, colors);

Data Types: xy:intary(4);
colors:intary(n);

Inquire Color Representation
status: = v_inq_color_representation
(*handle*, *color_number*, *set_flag*, rgb_returned);

Data Types: rgb_returned:intary(3);

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

Inquire Current Cursor Text Address**
status: = v_inq_cursor_address
(*handle*, row, column);

Inquire Current Fill Area Attributes
status: = v_inq_fill_attributes
(*handle*, attributes);

Data Types: attributes:intary(4);

Inquire Current Graphic Text Attributes
status: = v_inq_text_attributes
(*handle*, attributes);

Data Types: attributes:intary(10);

Inquire Current Polyline Attributes
status: = v_inq_polyline_attributes
(*handle*, attributes);

Data Types: attributes:intary(4);

Inquire Current Polymarker Attributes
status: = v_inq_polymarker_attributes
(*handle*, attributes);

Data Types: attributes:intary(4);

Inquire Cursor Text Mode**
status: = v_inq_cursor_text_mode
(*handle*);

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

Inquire Error
status: = v_inq_error;

Inquire Graphic Color Burst Mode**
status: = v_inq_graphic_burst_mode
(*handle*);

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

Inquire Page**

status:=v_inq_page
(*handle*, gr_mode, cur_mode);

Data Types: gr_mode:intary(3);
cur_mode:intary(3);

Message**

status:=v_message
(*handle*, *msg*, *wait_flag*);

Data Types: msg:lstring(n);

Open Workstation

status:=v_open_workstation
(*work_in*, handle, work_out);

Data Types: work_in:intary(19);
work_out:intary(66);

Output Alpha Text

status:=v_alpha_text
(*handle*, *char_string*, x_out, y_out);

Data Types: char_string:lstring(n);

Output Arc

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

Output Bar

status:=v_bar
(*handle*, *xy*);

Data Types: xy:intary(4);

Output Cell Array

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

Data Types: xy:intary(4);
colors:intary(n);

Output Circle

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

Output Cursor Addressable Text**

status:=v_cursor_text
(*handle*, *char_string*);

Data Types: char_string:lstring(n);

Output Filled Area
status:=v_fill_area
(*handle, count, xy*);

Data Types: xy:intary(2*count);

Output Graphic Text
status:=v_graphic_text
(*handle, x, y, char_string*);

Data Types: char_string:lstring(n);

Output Pie Slice
status:=v_pie_slice
(*handle, x_center, y_center, radius, start_angle, end_angle*);

Output Polyline
status:=v_polyline
(*handle, count, xy*);

Data Types: xy:intary(2*count);

Output Polymarker
status:=v_polymarker
(*handle, count, xy*);

Data Types: xy:intary(2*count);

Put Pels**
status:=v_put_pels
(*handle, xy, pel_array*);

Data Types: xy:intary (2);
pel_array:intary(n);

Read Cursor Movement Keys**
status:=v_read_cursor_keys
(*handle, input_mode*, direction, key);

Data Types: key:char;

Remove Graphic Input Cursor**
status:=v_remove_gin_cursor
(*handle*);

Reverse Video Off**
status:=v_reverse_video_off
(*handle*);

Reverse Video On**
status:=v_reverse_video_on
(*handle*);

Set Alpha Text Color Index

status: = v_set_alpha_color_index
(*handle, color_number*);

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

Set Alpha Text Font and Size

status: = v_set_alpha_font
(*handle, font_number, size_number,* capabilities);

Data Types: capabilities:intary(8);

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

Set Alpha Text Line Spacing

status: = v_set_alpha_line_spacing
(*handle, spacing_requested*);

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

Set Alpha Text Overstrike Mode

status: = v_set_alpha_overstrike_mode
(*handle, mode_number*);

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

Set Alpha Text Pass Through Mode

status: = v_set_alpha_pass_thru_mode
(*handle, mode_number*);

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

Set Alpha Text Position

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

Set Alpha Text Quality

status: = v_set_alpha_quality_mode
(*handle, mode_in*);

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

Set Alpha Text Subscript Superscript Mode
status:=v_set_alpha_sub_super_mode
(*handle, mode_number*);

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

Set Alpha Text Underline Mode
status:=v_set_alpha_underline_mode
(*handle, mode_number*);

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

Set Background Color Index
status:=v_set_background_color_index
(*handle, color_number*);

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

Set Character Height
status:=v_set_text_height
(*handle, height_requested,* char_width, cell_width, cell_height);

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

Set Color Representation
status:=v_set_color_representation
(*handle, color_number, rgb_input,* rgb_output);

Data Types: rgb_input:intary(3);
rgb_output:intary(3);

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

Set Cursor Text Attributes**
status:=v_set_cursor_attributes
(*handle, req_att,* sel_att);

Data Types: req_att:intary(4);
sel_att:intary(4);

Set Cursor Text Color Index**
status:=v_set_cursor_color_indices
(*handle, fore_requested, back_requested,* fore_selected, back_selected);

Set Cursor Text Mode**

status:=v_set_cursor_text_mode
(*handle, mode*);

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

Set Fill Color Index

status:=v_set_fill_color_index
(*handle, index_requested*);

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

Set Fill Interior Style

status:=v_set_fill_interior_style
(*handle, style_number*);

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

Set Fill Style Index

status:=v_set_fill_style_index
(*handle, style_number*);

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

Set Graphic Color Burst Mode**

status:=v_set_graphic_burst_mode
(*handle, mode*);

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

Set Graphic Text Alignment

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

Set Graphic Text Color Index

status:=v_set_text_color_index
(*handle, color_number*);

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

Set Graphic Text Font

status:=v_set_text_font
(*handle, font_number*);

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

Set Graphic Text String Baseline Rotation

status:=v_set_text_baseline_rotation
(*handle, angle_of_rotation*);

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

Set Line Edit Characters

status:=v_set_line_edit_chars
(*handle, line_del, char_del*);

Data Types: line_del:char;
char_del:char;

Set Page**

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

Data Types: gr_in:intary(2);
cur_in:intary(2);
gr_out:intary(2);
cur_out:intary(2);

Set Pen Speed**

status:=v_set_pen_speed
(*handle, speed*);

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

Set Polyline Color Index

status:=v_set_polyline_color_index
(*handle, color_number*);

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

Set Polyline Line Type

status:=v_set_polyline_linetype
(*handle, type_number*);

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

Set Polyline Line Width
status:=v_set_polyline_linewidth
(*handle, width*);

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

Set Polymarker Color Index
status:=v_set_polymarker_color_index
(*handle, color_number*);

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

Set Polymarker Height
status:=v_set_polymarker_height
(*handle, marker_height*);

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

Set Polymarker Type
status:=v_set_polymarker_type
(*handle, type_number*);

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

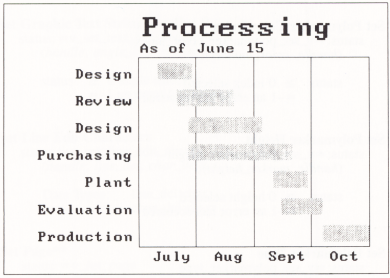
Set Writing Mode
status:=v_set_writing_mode
(*handle, mode_number*);

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

Update Workstation
status:=v_update_workstation
(*handle*);

Program Example

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




```
program gantt;
{ $list - }
{ $include: 'pasvdi.int'}
{ $list + }

type
  intary2 = intary(2);
  intary100 = intary(100);
  intary19 = intary(19);
  intary7 = intary(7);
  lstring23 = lstring(23);
  lstring13 = lstring(13);
                        { define types using the
                          supertypes }
var
  dev_handle, xheight,
  report, xwidth, cwidth,
  i, j, cheight           : integer;
  input_string            : lstring(2);
  echo_xy                 : intary2;
  work_in                 : intary19;
  savary                  : intary(66);
  start_dates, end_dates  : intary7;
  tasks                   : array [1..7] of lstring(17);
  title                   : lstring23;
  y_label                 : lstring13;
  months                  : array [1..7] of lstring(9);
  xy                      : intary100;
```

```
function fntrx(percent : integer) : integer;
  { function that changes percent of the page
    into NDC space for the X axis }
begin
  fntrx := round(percent / 100.0 * savary[52]);
end;



function fntry(percent : integer) : integer;
  { function that changes percent of the page
    into NDC space for the Y axis }
begin
  fntry :=  round(percent / 100.0 * savary[53]);
end;



procedure box(x_min,x_max,y_min,y_max : integer;
              var xy : intary100);
  { procedure to create coordinates for a frame
    from mins and maxs }
begin
  xy[1] := fntrx(x_min); xy[2] := fntry(y_min);
  xy[3] := fntrx(x_max); xy[4] := xy[2];
  xy[5] := xy[3]; xy[6] := fntry(y_max);
  xy[7] := xy[1]; xy[8] := xy[6];
  xy[9] := xy[1]; xy[10] := xy[2]
end;


begin  { the beginning of the main routine }

  echo_xy[1]  := 0;   echo_xy[2]  := 0;

  { work_in is the open workstation array.  It is the
    default attributes plus the string 'DISPLAY' }
  work_in[1] := 0;   work_in[2] := 1;
  work_in[3] := 1;   work_in[4] := 3;
  work_in[5] := 1;   work_in[6] := 1;
  work_in[7] := 1;   work_in[8] := 0;
  work_in[9] := 0;   work_in[10] := 1;
  work_in[11] := 1;  work_in[12] := 68;
  work_in[13] := 73; work_in[14] := 83;
  work_in[15] := 80; work_in[16] := 76;
  work_in[17] := 65; work_in[18] := 89;
  work_in[19] := 32;

  { the beginning and ends of all the bars }
  start_dates[1] := 83; start_dates[2] := 72;
  start_dates[3] := 70; start_dates[4] := 48;
  start_dates[5] := 48; start_dates[6] := 45;
  start_dates[7] := 40;

  end_dates[1] := 95; end_dates[2] := 83;
  end_dates[3] := 79; end_dates[4] := 75;
  end_dates[5] := 67; end_dates[6] := 60;
  end_dates[7] := 49;
```

```
{ define all the values of all the strings }
title :=  'Processing';
y_label :=  'As of June 15';
tasks[1] := 'Production';
tasks[2] := 'Evaluation';
tasks[3] := 'Plant';
tasks[4] := 'Purchasing';
tasks[5] := 'Design';
tasks[6] := 'Review';
tasks[7] := 'Design';
months[1] := 'July';
months[2] := 'Aug';
months[3] := 'Sept';
months[4] := 'Oct';

{ open the workstation }
report := v_open_workstation(work_in,
                             dev_handle,savary);

{ set the constants for drawing the grids }
xy[2] := fntry(10); xy[4] := fntry(80);

{ the x location of the first grid line in percent
  of the page }
i := 50;

while (i <= 80) do { for each one of the grid lines }
  begin  { set the variable part of the grid lines }
    xy[1] := fntrx(i); xy[3] := xy[1];
    report := v_polyline(dev_handle,2,xy);
    i := i + 15;  { move to the next grid }
  end;

{ set a new character height}
xheight := v_set_text_height(dev_handle,fntry(4),
                             xwidth,cwidth,cheight);

{ set the text alignment to the top center }
report := v_set_text_alignment(dev_handle,1,2,1,j);

i := 43;  { the location of the first month label }
j := 1;   { an index into the array of months }

{ for all the month label locations }
while (i <= 88) do
  begin
    { write the month label }
    report := v_graphic_text(dev_handle,fntrx(i),
                             fntry(10),months[j]);
    j := j + 1;
    i := i + 15;
  end;

{ set text alignment to right center }
report := v_set_text_alignment(dev_handle,2,1,1,j);
```

```
j := 1;  { index into the tasks }
i := 15; { the location of the first task label }
while (i <= 75) do{ for all the task label locations }
  begin
    { write out the task label }
    report := v_graphic_text(dev_handle,fntrx(33),
                             fntry(i),tasks[j]);
    i := i + 10;  { the next location }
    j := j + 1;   { the next task }
  end;

{ set the text alignment to the bottom left }
report := v_set_text_alignment(dev_handle,0,0,1,1);

{ write out the sublabel }
report := v_graphic_text(dev_handle,
                         fntrx(35),
                         fntry(82),
                         y_label);

{ set the text height up for the title }
report := v_set_text_height(dev_handle,fntry(9),
                            xwidth,cwidth,cheight);

{ put the title on the chart }
report := v_graphic_text(dev_handle,fntrx(35),
                            fntry(88),title);

{ set the fill pattern to 45 degree hatching }
report := v_set_fill_style_index(dev_handle,2);
report := v_set_fill_interior_style(dev_handle,3);

j := 1;  { index into the bar arrays }
i := 12; { y location of the first bar }
{ for every one of the bar locations }
while (i <= 72) do
  begin
    { create the two corners that describe a bar }
    xy[2] := fntry(1); xy[4] := fntry(i + 6);
    xy[1] := fntrx(start_dates[j]);
    xy[3] := fntrx(end_dates[j]);

    report := v_bar(dev_handle,xy); { draw the bar }
    i := i + 10;      { the next bar location }
    j := j + 1;       { index to the next label }
  end;

box(35,95,10,80,xy); { create the box coordinates }

{draw a frame around the chart}
report := v_polyline(dev_handle,5,xy);
box(0,100,0,100,xy);  { create border coordinates }
```

```
  { draw the border }
  report := v_polyline(dev_handle,5,xy);
  { wait for a carriage return
    before closing the workstation }
  report := v_request_string(dev_handle,2,0,echo_xy,
                             input_string);

  { close the workstation down }
  report := v_close_workstation(dev_handle);
end.
```

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

Printed in the
United States of America

59X8619