Oracle® Forms Developer

Graphics Builder Reference

Release 6*i*

January, 2000

Part No. A73075-01

ORACLE®

Table of Contents

We Appreciate Your Comments

Reader's Comment Form - A73075-01

Oracle Corporation welcomes your comments about this manual's quality and usefulness. Your feedback is an important part of our revision process.

- Did you find any errors?
- Is the information presented clearly?
- Are the examples correct? Do you need more examples?
- What features did you like?

If you found any errors or have any other suggestions for improvement, please write the topic, chapter, and page number below:

Please send your comments to:

Forms Developer Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Or send comments by e-mail to: oddoc@us.oracle.com

Please include your name, address, and telephone number for a reply:

Thank you for your help.

Preface

Welcome to Release 6*i* of the *Oracle Forms Developer: Graphics Builder Reference*.

This reference guide includes information to help you effectively work with Forms Developer Graphics Builder and contains detailed information about the following:

- Built ins
- Properties
- Attributes
- Global variables

This preface explains how this user's guide is organized and introduces other sources of information that can help you use Forms Developer Graphics Builder.

Prerequisites

You should be familiar with your computer and its operating system. For example, you should know the commands for deleting and copying files and understand the concepts of search paths, subdirectories, and path names. Refer to your Microsoft Windows 95 or NT and DOS product documentation for more information.

You should also understand the fundamentals of Microsoft Windows, such as the elements of an application window. You should also be familiar with such programs as the Explorer, Taskbar or Task Manager, and Registry.

Notational Conventions

The following typographical conventions are used in this guide:

Convention	**Meaning**
fixed-width font	Text in a fixed-width font indicates commands that you enter exactly as shown. Text typed on a PC is not case-sensitive unless otherwise noted. In commands, punctuation other than brackets and vertical bars must be entered exactly as shown.
lowercase	Lowercase characters in a command statement represent a variable. Substitute an appropriate value.
UPPERCASE	Uppercase characters within the text represent command names, SQL reserved words, and keywords.
boldface	Boldface is used to indicate user interface items such as menu choices and buttons.
C>	C> represents the DOS prompt. Your prompt may differ.

Built-in Subprograms

Chart Built-ins

OG_Delete_Column

Description This procedure deletes a column from a custom query.

Syntax

```
 PROCEDURE OG_Delete_Column
   (query_hdl  OG_Query,
    indx       NUMBER,
    total      NUMBER);
```

Parameters

query_hdl	Is the handle to the query from which to delete the column.
indx	Is the index of the first column to delete from the query.
total	Is the total number of columns to delete.

OG_Delete_Column Example

```
/* The following procedure deletes a column
 ** from the query ‘query0’:
 */

PROCEDURE example(col_num number) IS
   query   OG_Query;
BEGIN
   query:=OG_Get_Query(‘query0’);
   OG_Delete_Column(query, col_num, 1);
END;
```

OG_Delete_Field

Description This procedure deletes one or more fields from the specified chart object.

Syntax

```
PROCEDURE OG_Delete_Field
  (chart_hdl  OG_Object,
   indx       NUMBER,
   total      NUMBER);
```

Parameters

chart_hdl	Is the handle to the chart object.
indx	Is the index of the first field to delete from the field list.
total	Is the total number of fields to delete.

Usage Notes Deleting a field only removes it from the specified chart. It does not delete (or otherwise modify) the field template that the field may reference. In addition, any changes you make to the chart's field list will not be applied until the chart is updated via a call to OG_Update_Chart.

OG_Delete_Field Examples

```
/* Suppose one chart currently displays plots for both salary
 ** and commission data, and you want to remove the
 ** commission plot from that chart and plot it on another one.
 */

PROCEDURE transfer_comm(chart1 IN OG_Object, chart2 IN
OG_Object, field_index in number) IS
   the_field   OG_Field;
BEGIN
   the_field:=OG_Get_Field(Chart1, field_index);
   OG_Delete_Field(Chart1, field_index, 1);
   OG_Insert_Field(Chart2, the_field, OG_Last);
   OG_Update_Chart(Chart1, OG_All_Chupda);
   OG_Update_Chart(Chart2, OG_All_Chupda);
END;
```

OG_Get_Chart_Element

Description Given the handle to a group of chart elements (bars, pie slices, etc) and a row number, this function returns the individual element corresponding to that row number.

Syntax

```
FUNCTION OG_Get_Chart_Element
```

```
  (group_hdl  OG_Object,
   row_num    NUMBER)
RETURN OG_Object;
```

Parameters

group_hdl	Is the handle to the group containing the chart element.
row_num	Is the row number corresponding to the chart element you wish to get.

Returns The individual chart element for the specified row number.
Usage Notes The group handle can be retrieved from the chart object using OG_Get_Object with the appropriate name.

OG_Get_Chart_Element Examples

```
/* The following procedure changes the color of the first:
 ** bar in a column chart, regardless of its value:
 */

PROCEDURE example(chart OG_Object) IS
  bars_group  OG_Object;
  elem        OG_Object;
BEGIN
  bars_group := OG_Get_Object('Sal_Bars', chart);
  elem := OG_Get_Chart_Element(Bars_Group, 0);
  OG_Set_Fillcolor(Elem, 'red');
END;
```

OG_Get_Column

Description This function returns the name of the query column represented by a specific chart element.
Syntax
```
FUNCTION OG_Get_Column
  (chelement_hdl  OG_Object)
RETURN VARCHAR2;
```

Parameters

chelement_hdl	Is the handle to the chart element.

Returns The name of the column associated with the chart element.

OG_Get_Column Examples

```
/* The following function returns the query column represented by
 ** the first bar in a column chart:
*/

FUNCTION example(chart OG_Object) RETURN CHAR IS
  bars  OG_Object;
  elem  OG_Object;
  col   VARCHAR2(15);
BEGIN
  bars := OG_Get_Object('Sal_Bars', chart);
  elem := OG_Get_Chart_Element(Bars, 0);
  col := OG_Get_Column(Elem);
  RETURN(col);
END;
```

OG_Get_Field

Description This function returns a record containing the field's attribute values in the specified chart.

Syntax

```
FUNCTION OG_Get_Field
  (chart_hdl  OG_Object,
   indx       NUMBER)
RETURN OG_Field;
```

Parameters

chart_hdl	Is the handle to the chart object.
indx	Is the index of the field in the chart's field list to be returned.

Returns The attributes of the specified field.

OG_Get_Field Examples

```
/* Suppose one chart currently displays plots for both salary
 ** and commission data, and you want to remove the
 ** commission plot from that chart and plot it on another one:
 */

PROCEDURE transfer_comm(chart1 IN OG_Object, chart2 IN
  OG_Object, field_index IN NUMBER) IS
   the_field   OG_Field
BEGIN
   the_field:=OG_Get_Field(The_Chart, field_index);
   OG_Delete_Field(Chart1, field_index, 1);
   OG_Insert_Field(Chart2, the_field, OG_Last);
END;
```

OG_Get_Row

Description This function returns the query row number that is represented by a specific chart element.

Syntax

```
FUNCTION OG_Get_Row
  (chelement_hdl  OG_Object,
RETURN NUMBER;
```

Parameters

chelement_hdl	Is the handle to the chart element.

Returns The row number associated with the chart element.

OG_Get_Row Examples

```
/* The following format trigger explodes the pie slice
 ** representing SAL for employee 'SMITH':
 */

PROCEDURE OGFORMATTRIG0(elem IN OG_Object,
                        query IN OG_Query) IS
  ename    VARCHAR2(10);
  chart    OG_Object;
  row_num  NUMBER;
BEGIN
  ename := OG_Get_Charcell(Query, 'ENAME');
  IF ename = 'SMITH' THEN
    chart := OG_Get_Object('Chart0');
    row_num := OG_Get_Row(Elem);
    OG_Set_Explosion(Chart, row_num, 'SAL', 25);
  END IF;
END;
```

OG_Insert_Field

Description This procedure inserts a new field into the specified chart.

Syntax

```
PROCEDURE OG_Insert_Field
  (chart_hdl  OG_Object,
   field_rec  OG_Field,
   indx       NUMBER);
```

Parameters

chart_hdl	Is the handle to the chart object,
field_rec	Is the record containing the field's attributes.
indx	Is the index at which to insert the new field in the chart's field list. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of fields in the chart prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new field at the beginning of the chart's field list (index = 0). **OG_Last** Means insert the new field at the end of the chart's field list (index = the number of fields in the chart prior to the insertion).

Usage Notes Any changes you make to the chart's field list are not applied until the chart is updated via a call to OG_Update_Chart.

OG_Insert_Field Examples

```
/* Suppose one chart currently displays plots for both
 ** salary and commission data, and you want to remove
 ** the commission plot from that chart and plot it on another one:
 */

PROCEDURE transfer_comm (chart1 IN OG_Object, chart2 IN
OG_Object, field_index IN NUMBER) IS
   the_field   OG_Field;
BEGIN
   the_field:=OG_Get_Field(The_Chart, field_index);
   OG_Delete_Field(Chart1, field_index, 1);
   OG_Insert_Field(Chart2, the_field, OG_Last);
END;
```

OG_Make_Chart

Description This function creates a chart.

Syntax

```
FUNCTION OG_Make_Chart
  (position  OG_Point,
   height    NUMBER,
   width     NUMBER,
   template  OG_Template,
   query      OG_Query)
RETURN OG_Object;
```

Parameters

position	The x- and y-coordinates of the chart frame.
height	The chart height.
width	The chart width.
template	The template to use for the chart.
query	The query to use for the chart.

Returns A handle to the newly created chart.
Usage Notes The chart will not be complete until you add fields to it using OG_Insert_Field and update it using OG_Update_Chart.

OG_Make_Chart Examples

```
/* The following function creates a chart using
 ** the specified template and query:
 */

FUNCTION example(template OG_Template, query OG_Query) RETURN OG_Object IS
  chart   OG_Object;
  pos     OG_Point;
  height  NUMBER;
  width   NUMBER;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;
  height := 4* OG_Inch;
  width := 4* OG_Inch;

  chart := OG_Make_Chart(Pos, height, width, template, query);
  RETURN(chart);
END;
```

OG_Update_Chart

Description This procedure updates the specified part(s) of the specified chart to reflect new query results or new attributes that have been applied to chart elements. You must have executed the query at least once before you can update a chart that is based upon it.
Syntax

```
PROCEDURE OG_Update_Chart
  (chart_hdl    OG_Object,
   chart_mask   NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

chart_hdl	Is the handle to the chart to be updated.
chart_mask	Specifies which part(s) of the chart should be updated. The value of this argument may be one of the following built-in constants: **OG_All_Chupda** Means update the entire chart. **OG_Dep1axis_Chupda** Means update only the parts of the chart associated with the first dependent axis.. **OG_Dep2axis_Chupda** Means update only the parts of the chart associated with the second dependent axis..

	OG_Frame_Chupda Means update only the parts of the chart associated with the frame.
	OG_Indaxis_Chupda Means update only the parts of the chart associated with the independent axis.
	OG_Inframe_Chupda Means update only the parts of the chart that appear within the frame.
	OG_Legend_Chupda Means update only the parts of the chart associated with the legend.
	OG_None_Chupda Means do not update any parts of the chart.
	OG_Title_Chupda Means update only the chart title.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes When this procedure is invoked, Graphics Builder will destroy the current chart and rebuild it, based on updated query results and attribute settings. Because of this, any changes to a chart element's attribute settings since the last invocation of OG_Update_Chart will be lost. For example, if you use OG_Set_Attr to set the attributes for a specific bar in the chart and then update it, you will see the desired results. If you call OG_Update_Chart again, however, the changes will be lost and the bar will appear with its default settings.
Remember, then, that each time you update the chart, you must first make the changes to the chart elements. In many cases you will find that this extra step is desirable, since the criteria for modifying chart elements may change as the data changes.

OG_Update_Chart Examples

```
/* Suppose you want to update a chart periodically.
 ** You could write the following timer trigger:
 */

PROCEDURE my_timer IS+
   my_query   OG_Query;
   my_chart   OG_Object;
BEGIN
   my_query:=OG_Get_Query('Emp_Query');
   my_chart:=OG_Get_Object('Emp_Chart');
   OG_Execute_Query(My_Query);
   OG_Update_Chart(My_Chart, OG_All_Chupda);
END;
```

Database Built-ins

OG_Connect
OG_Logged_On
OG_Logoff
OG_Logon

OG_Connect

Description This function shows the Connect dialog box.

Syntax
```
FUNCTION OG_Connect
RETURN BOOLEAN;
```

Parameters:

None.

OG_Connect Examples

```
/* Suppose your application requires the
** user to be connected to a database.  The
** following procedure checks if a connection
** exists and, if not, prompts the user to
** connect by showing the Connect dialog box:
*/

PROCEDURE ensure_connection IS
BEGIN
   IF NOT OG_Logged_On THEN
   status:=OG_Connect;
   END IF;
END;
```

OG_Logged_On

Description This function returns TRUE if the user is currently connected to a database, and FALSE if not connected.

Syntax
```
FUNCTION OG_Logged_On
RETURN BOOLEAN;
```

Parameters:

None.

Returns		
	TRUE	If the user is connected to a database.
	FALSE	If the user is not connected to a database.

OG_Logged_On Examples

```
/* Suppose your application requires the user to be
** connected to a database.  The following procedure
** checks if a connection exists and, if not, prompts the
** user to connect by showing the Connect dialog box:
*/

PROCEDURE ensure_connection IS
BEGIN
   IF NOT OG_Logged_On THEN
      OG_Connect;
   END IF;
END;
```

OG_Logoff

Description This procedure closes the existing database connection.

Syntax

```
PROCEDURE OG_Logoff;
```

Parameters:

None.

OG_Logoff Examples

```
/* Suppose you want to disconnect from a database when the display is closed.
** You could write the following Close Display trigger:
*/

PROCEDURE close_trig IS
BEGIN
   IF OG_Logged_On THEN
      OG_Logoff;
   END IF;
END;
```

OG_Logon

Description This procedure connects to the specified database.

Syntax

```
PROCEDURE OG_Logon
  (username        VARCHAR2  :=  NULL,
   password        VARCHAR2  :=  NULL,
   connect_string  VARCHAR2  :=  NULL);
```

Parameters

username	Is the username to use.
password	Is the password to use.
connect_string	Is the database connect string to use. To connect to a remote database, you must provide the appropriate SQL*Net database connect string. For more information, see the *Oracle Network Manager Administrator's Guide*.

Usage Notes If a connection already exists, it is first dropped, regardless of whether the connection then attempted by this procedure is successful.

OG_Logon Examples

```
/* Suppose your application requires the user to be connected to a database.
** The following procedure checks if a connection exists and, if not,
**automatically establishes a connection:
*/

PROCEDURE ensure_connection IS
BEGIN
   IF NOT OG_Logged_On THEN
      OG_Logon('Scott', 'tiger', 't:london:MY_DB');
   END IF;
END;
```

Display Built-ins

OG_Close_Display
OG_Generate_Display
OG_Get_Display
OG_Isnull
OG_Open_Display
OG_Save_Display

OG_Close_Display

Description This procedure closes the specified display and destroys all windows used by that display. It also causes the Close Display trigger for the specified display to execute.

Syntax

```
PROCEDURE OG_Close_Display
  (display_hdl  OG_Display);
```

Parameters

display_hdl	Is the handle to the display to be closed.

Usage Notes Note that if you call a procedure that closes the current display (i.e., the display in which the executing procedure is defined), OG_Close_Display must appear on the last line of that procedure. In other words, you cannot execute any further PL/SQL instructions in a display after you have closed it.

OG_Close_Display Examples

```
/* Suppose the user is through with one display,
** and you want to close it and open another one.
*/

PROCEDURE continue (old_disp_name, new_disp_name) IS
   old_disp   OG_Display;
```

OG_Generate_Display

Description This function generates the current of the display. The generated display may be run by the Graphics Builder Runtime and Batch executables.

Syntax

```
PROCEDURE OG_Generate_Display;
PROCEDURE OG_Generate_Display
  (name        VARCHAR2,
   repository  OG_Number);
```

Parameters

name	Is the name to which the display is generated. If the display is to be stored in the database, this argument should contain only the name of the display. If the display is to be stored in the file system, this argument should contain the absolute or relative pathname of the display file.
repository	Specifies whether the display is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the display is to be stored in the database. **OG_Filesystem** Means the display is to be stored in the file system.

Usage Notes If you omit *name* and *repository*, the display is generated to the name and repository from which it was most recently opened.

OG_Generate_Display Examples

```
/* Suppose your display allows the user to interactively specify
**which queries to view, and what chart types to use.
**When the user selects a `generate' button, you may want to
**generate a runtime version of the display
** that the user can use in the future.
*/

PROCEDURE gen(buttonobj IN OG_Object, hitobj IN OG_Object,
  win IN OG_Window, eventinfo IN OG_Event) IS
```

OG_Get_Display

Description Note that *display_name* must already be open in the current Graphics Builder session. To open a display other than the one that is currently running, use OG_Open_Display.

Syntax

```
FUNCTION OG_Get_Display
RETURN OG_Display;
FUNCTION OG_Get_Display
  (display_name  VARCHAR2,
   repository    NUMBER)
RETURN OG_Display;
```

Parameters

display_name	Is the name of the display. If the display is stored in the database, this argument should contain only the name of the display. If the

	display is stored in the file system, this argument should contain the absolute or relative pathname of the display file.
repository	Specifies whether the display is stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the display is to be stored in the database. **OG_Filesystem** Means the display is to be stored in the file system.

Returns A handle to the specified display. If the display does not exist or is not open, this function returns a null handle.

Usage Notes If *display_name* and *repository* are omitted, this function returns a handle to the current display.

OG_Get_Display Examples

```
/* Suppose the user is through with one display,
** and you would like to close it and open another one.
*/
.
PROCEDURE continue(old_disp_name, new_disp_name) IS
   old_disp   OG_Display;
   new_disp   OG_Display;
BEGIN
   old_dispb:=OG_Get_Display(Old_Disp_Name, OG_Filesystem);
   OG_Close_Display(Old_Disp);
   new_dispb:=OG_Open_Display(New_Disp_Name, OG_Filesystem);
END;
```

OG_Isnull

Description This function determines if the specified handle is a null handle.

Syntax

```
FUNCTION OG_Isnull                          query
  (handle  OG_Query)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          object
  (handle  OG_Object)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          chart template
  (handle  OG_Template)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          button procedure
  (handle  OG_Buttonproc)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          sound
  (handle  OG_Sound)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          window
  (handle  OG_Window)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          layer
  (handle  OG_Layer)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          timer
  (handle  OG_Timer)
RETURN BOOLEAN;
FUNCTION OG_Isnull                          display
```

```
  (handle  OG_Display)
RETURN BOOLEAN;
FUNCTION OG_Isnull                            axis
  (handle  OG_Axis)
RETURN BOOLEAN;
FUNCTION OG_Isnull                            field template
  (handle  OG_Ftemp)
RETURN BOOLEAN;
FUNCTION OG_Isnull                            reference line
  (handle  OG_Refline)
RETURN BOOLEAN;
```

Parameters

handle	Is the handle to be evaluated.
Returns TRUE	If the handle is a null handle.
FALSE	If the handle is not a null handle.

OG_Isnull Examples

```
/* Suppose your display occasionally creates a text object that contains a warning message.  At times you
** may want to remove the warning message before continuing with the execution of the display.  Rather
** than keeping track of whether a warning has been generated, you can check for the existence of the
** text object before deleting it.
*/

PROCEDURE remove_warning IS
   warning_obj   OG_Object;
BEGIN
   warning_obj:=OG_Get_Object('warning');
   IF NOT (OG_Isnull(warning_obj)) THEN;
      OG_Destroy(warning_obj);
   END IF;
END;
```

OG_Open_Display

Description This function opens the specified display and executes its Open Display trigger. It returns a handle to the display, which you may later use as an argument for OG_Close_Display. If the display does not exist, this function returns a null handle.

Syntax

```
FUNCTION OG_Open_Display
  (display_name  VARCHAR2,
   repository    NUMBER)
RETURN OG_Display;
```

Parameters

display_name	Is the name of the display. If the display is stored in the database, this argument should contain only its name. If the display is stored in the file system, this argument should contain the absolute or relative pathname of the display file.
repository	Specifies whether the display is stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the display is stored in the database. **OG_Filesystem** Means the display is stored in the file system.

Returns A handle to the newly opened display.

Usage Notes Note that this function does *not* accept a handle to a display as an argument. This is because the existence of the display in the file system or database must be verified before the display can be opened. Suppose you used OG_Get_Display to get the display handle, then you deleted the display from the file system or database. If you tried to pass the display handle to OG_Open_Display, it would not be able to find the display that the handle referred to. Therefore, the display name must again be used.

OG_Open_Display Examples

```
/* Suppose the user is through with one display,
** and you would like to close it and open another one.
*/
.
PROCEDURE continue(old_display_name IN CHAR,
new_display_name IN CHAR) IS
   old_display   OG_Display;
   new_display   OG_Display;
BEGIN
   old_display:=OG_Get_Display(old_display_name, OG_Filesystem);
   new_display:=OG_Open_Display(new_display_name, OG_Filesystem);
   OG_Close_Display(old_display);
END;
```

OG_Save_Display

Description This function saves the current state of the display. The saved display is complete and may be opened and edited in the Graphics Builder Builder.

Syntax

```
PROCEDURE OG_Save_Display;
PROCEDURE OG_Save_Display
  (name        VARCHAR2,
   repository  OG_Number);
```

Parameters

name	Is the name to which the display is saved. If the display is to be stored in the database, this argument should contain only the name of the display. If the display is to be stored in the file system, this argument should contain the absolute or relative pathname of the display file.
repository	Specifies whether the display is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the display is to be stored in the database. **OG_Filesystem** Means the display is to be stored in the file system.

Usage Notes If you omit *name* and *repository*, the display is saved to the name and repository from which it was most recently opened.

OG_Save_Display Examples

```
/* Suppose you want to import 100 TIFF images.  Doing this
**manually is tedious and would take a long time.
**The solution is to write the following procedure,
**which imports images from the files named `image00'
**through `image99'.  When finished, it saves the display
**so that you can open it again in the Builder.
*/

PROCEDURE import_100 IS
   the_image   OG_Image;
   file_name   VARCHAR2(7);
BEGIN
   FOR i IN 0..99 LOOP
      file_name:='image'||SUBSTR(TO_CHAR(i, `09'), 2);
      the_image:=OG_Import_Image(File_Name, OG_Filesystem, OG_Tiff_Iformat);
   END LOOP;
   OG_Save_Display;
```

```
END;
```

Graphic Object Built-ins

OG_Clone (Object)
OG_Damage (Object)
OG_Delete_Child
OG_Delete_Cmptext
OG_Delete_Point
OG_Delete_Property
OG_Delete_Smptext
OG_Destroy (Object)
OG_Draw
OG_Export_Drawing (Display)
OG_Export_Drawing (Object/Layer)
OG_Export_Drawing (Window)
OG_Export_Image
OG_Get_Char_Property
OG_Get_Child
OG_Get_Cmptext
OG_Get_Date_Property
OG_Get_Num_Property
OG_Get_Object
OG_Get_Point
OG_Get_Smptext
OG_Import_Drawing
OG_Import_Image
OG_Insert_Child
OG_Insert_Cmptext
OG_Insert_Point
OG_Insert_Smptext
OG_Make_Ellipse
OG_Make_Group
OG_Make_Image
OG_Make_Line
OG_Make_Poly
OG_Make_Rect
OG_Make_Rrect
OG_Make_Symbol
OG_Make_Text
OG_Move
OG_Point_In
OG_Point_Near
OG_Property_Exists
OG_Rotate
OG_Same
OG_Scale
OG_Set_Edgecolor
OG_Set_Fillcolor
OG_Set_Property
OG_Synchronize
OG_Update_Bbox

OG_Clone (Object)

Description This function creates a new object that is identical to the specified object.

Syntax

```
FUNCTION OG_Clone
  (object_hdl   OG_Object,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE)
RETURN OG_Object;
```

Parameters

object_hdl	Is the handle to the object to be cloned.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Returns The handle to the newly created object.

OG_Clone (Object) Examples

```
/* Suppose you have created an object, and you want to
** create another identical object without having to
**again specify the same properties.
*/

PROCEDURE dup_object(old_object IN OG_Object) IS
   new_object   OG_Object;
BEGIN
   new_object:=OG_Clone(Old_Object);
END;
```

OG_Damage (Object)

Description This procedure damages an object on the layout.

Syntax

```
PROCEDURE OG_Damage
  (object_hdl  OG_Object);
```

Parameters

object_hdl	Is the handle to the object to damaged.

OG_Damage (Object) Examples

```
/*Suppose you want to move an object.  The default behavior of the built-in
**procedure OG_Move is to update the bounding boxes of all of the modified
**object's antecedants, including the layer on which the object resides.
**To update a layer's bounding boxes, Graphics Builder must examine every object
**on that layer.  If the layer contains a large number of objects,
**this operation can be very time-consuming.
*/
 /*To make your application more efficient, you can move the object
**while inhibiting this automatic bounding box update, then explicitly
**update only that object's bounding boxes.  (Note that since the
**automatic bounding box update does not occur, the bounding boxes
**of the object's antecedants may be inaccurate.)
*/
 /*When you modify an object with a FALSE bounding box update flag,
**you may also want to use a FALSE damage flag.  In this case,
**when you are through modifying the object, you would invoke
**OG_Damage to explicitly damage the object.
*/
PROCEDURE move_efficiently (the_object OG_Object) IS
```

```
   offset   OG_Point;
BEGIN
   offset.x:=OG_Inch;
   offset.y:=OG_Inch;
   OG_Move(The_Object, offset, FALSE, FALSE)
   OG_Update_Bbox(The_Object, OG_Bothbbox);
   OG_Damage(The_Object);
END;
```

OG_Delete_Child

Description This procedure deletes one or more child objects from the specified group object.

Syntax

```
PROCEDURE OG_Delete_Child
  (group_hdl    OG_Object,
   indx         NUMBER,
   total        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

group_hdl	Is the handle to the group object.
indx	Is the index of the first object to delete from the group's child list.
total	Is the total number of child objects to delete.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes .To delete a child means only that the object will no longer be associated with its parent group; it does *not* mean that the child object will be destroyed. The parent attribute in the deleted child's generic attribute record will be set to a null handle, and the child will no longer exist in the group's object list. Note that an object that has a null handle for a parent will not be displayed on the layout.
You can delete a layer by treating the display's root object as a group, and by passing it and the layer index to this procedure.

OG_Delete_Child Examples

```
/*Suppose you have a several objects representing products
**in a warehouse, and you want to move one of the products
**from one warehouse to another.  Your display may use a group
**comprised of the products to represent the inventory for each
**warehouse.  To move a product from one warehouse to another,
**you would want to get the handle to the product object,
**delete it from one warehouse group, and add it to another
**warehouse group.
*/
 /*Note that this procedure changes only the internal composition
**of the group objects.  To move or change the appearance of the
**product object, you must use other Graphics Builder built-in procedures.
*/
PROCEDURE move_prod(warehouse1 IN OG_Object, warehouse2 IN
OG_Object, prod_index IN number) IS
   the_prod   OG_Object;
BEGIN
   the_prod:=OG_Get_Child(Warehouse1, prod_index);
   OG_Delete_Child(Warehouse1, prod_index, 1);
   OG_Insert_Child(Warehouse2, the_prod, OG_Last);
END;
```

OG_Delete_Cmptext

Description This procedure deletes one or more compound text elements from the specified text object. As described in "Text Attributes," a compound text element represents one line of text in a text object.

Syntax

```
PROCEDURE OG_Delete_Cmptext
  (text_hdl     OG_Object,
   indx         NUMBER,
   total        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text_hdl	Is the handle to the text object.
indx	Is the index within the text object of the first compound text element to delete from the compound text element list.
total	Is the total number of compound text elements to delete.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes When you delete a compound text element, it and all of the simple text elements that compose it will be destroyed.

OG_Delete_Cmptext Examples

```
/*Suppose you use a text object to display messages to the user.
**A previous part of your application produced two-line messages,
**but the part of the display that is currently being used produces
**only one-line messages.  You may want to delete the extraneous
**compound text element.
*/

PROCEDURE delete_msg_line(msg_object IN OG_Object,
line_index IN number) IS
BEGIN
   OG_Delete_Cmptext(Msg_Object, line_index, 1);
END;
```

OG_Delete_Point

Description This procedure deletes one or more points from the specified polygon or polyline object.

Syntax

```
PROCEDURE OG_Delete_Point
  (poly_hdl     OG_Object,
   indx         NUMBER,
   total        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

poly_hdl	Is the handle to the polygon or polyline object.
indx	Is the index of the first point to delete from the point list.
total	Is the total number of points to delete.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes If the object was created in the Builder, the initial index values for the points will correspond to the order in which the mouse was selected when the object was drawn (with the first point having an index of 0). Index values for points that were inserted programmatically will depend on the index that was specified when they were inserted.

OG_Delete_Point Examples

```
/*Suppose you have several polygons on a map, each of which
**connects the cities along a specific distribution route.
**If a city is transferred from one distribution route to another,
**you would want to get the point representing that city,
**delete it from one polygon, and add it to another polygon.
*/

PROCEDURE move_city(route1 IN OG_Object, route2 IN
OG_Object, city_index IN number) IS
   the_city   OG_Point;
BEGIN
   the_city:=OG_Get_Point(Route1, city_index);
   OG_Delete_Point(Route1, city_index, 1);
   OG_Insert_Point(Route2, OG_Last, the_city);
END;
```

OG_Delete_Property

Description This procedure deletes an object's user-defined property.

Syntax

```
PROCEDURE OG_Delete_Property
  (object_hdl  OG_Object,
   prop_name   VARCHAR2);
```

Parameters

object_hdl	Is the handle to the object whose property you want to delete.
prop_name	Is the name of the property to delete.

OG_Delete_Property Examples

```
/* The following procedure deletes the property 'priority'
** from every child object in a named group:
*/

PROCEDURE example(group_name VARCHAR2) IS
  group_obj    OG_Object;
  child_count  NUMBER;
  child_obj    OG_Object;
BEGIN
  group_obj := OG_Get_Object(Group_Name);
  child_count := OG_Get_Childcount(Group_Obj);

  FOR i IN 0..child_count LOOP
    child_obj := OG_Get_Child(Group_Obj, i);
    OG_Delete_Property(Child_Obj, 'priority');
  END LOOP;
END;
```

OG_Delete_Smptext

Description This procedure deletes one or more simple text elements from the specified compound text element in the specified text object. As described in "Text Attributes," a simple text element represents a text string in a compound text element.

Syntax

```
PROCEDURE OG_Delete_Smptext
  (text_hdl     OG_Object,
   cmpindex     NUMBER,
   smpindex     NUMBER,
   total        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text_hdl	Is the handle to the text object.
Cmpindex	Is the index of the compound text element that contains the simple text element(s) to delete.
smpindex	Is the index of the first simple text element to delete
total	Is the total number of simple text elements to delete.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes Deleting simple text will not affect the existence or index of its compound text element. It is possible, in fact, to delete all of the simple text for a compound text element, and be left with an empty compound text element.

OG_Delete_Smptext Examples

```
/*Suppose you have created a message text object.  To change
**the message it contains, you would delete the simple text element
**containing the current message and insert a new simple text element
**containing the new message.  To maintain the font and other attributes,
**however, you first would want to get the simple text element into an
**attribute record.  That way, you could modify only the text string,
**and leave the other attribute settings (such as font) unchanged.
*/
```

```
PROCEDURE put_msg(mess IN VARCHAR2) IS
  msgobj    OG_Object;
  msgrec    OG_Smptext_Attr;
BEGIN
  msgobj := OG_Get_Object('msg');
  OG_Get_Smptext(msgobj, 0, 0, msgrec);
  OG_Delete_Smptext(msgobj, 0, 0, 1);
  msgrec.mask:= OG_STR_SMPTEXTA;
  msgrec.str:= mess;
  OG_Insert_Smptext(msgobj, msgrec, 0, OG_LAST);
END;
```

OG_Destroy (Object)

Description This procedure destroys the specified object If you destroy a group object, all of that group's children are also destroyed

Syntax

```
PROCEDURE OG_Destroy
  (object_hdl   OG_Object,
   recurse      BOOLEAN    :=  FALSE,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object_hdl	Is the handle to the object to destroy.
Recurse	Is the recursive-destroy flag. This argument is optional; its value will default to FALSE if not otherwise specified. Also, this argument is ignored if the object you are destroying is not the only child of a group.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes .If the object you are destroying is the only child of a group, a *recurse* value of TRUE indicates that that parent group also should be destroyed. This action will continue up the object's group tree (i.e., if the object's parent is the only child of *its* parent group, then *that* parent group also will be destroyed, etc.). Finally, if the last object on a layer is destroyed, the layer itself is also destroyed (unless it is active in some window).
If *recurse* is FALSE, only the object specified by *object_hdl* will be destroyed.

OG_Destroy (Object) Examples

```
/* The following procedure destroys the specified object:
*/

PROCEDURE destroy_obj(obj_name VARCHAR2) IS
  object  OG_Object;
BEGIN
  object := OG_Get_Object(Obj_Name);
  OG_Destroy(Object);
END;
```

OG_Draw

Description This procedure causes the specified object to be drawn on the layout.

Syntax

```
PROCEDURE OG_Draw
  (object_hdl  OG_Object);
```

Parameters

object_hdl	Is the handle to the object to draw.

Usage Notes Unlike other procedures that modify objects, this procedure does not "damage" a rectangular area on the layout. It draws only the specified object, and disturbs nothing else.
The benefit of using this procedure is that you can have an object appear on the layout, while preventing Graphics Builder from re-drawing a rectangular damage region that may be larger than necessary.

OG_Draw Examples

```
/*Suppose you want to clone an object and have it appear on the
**layout smoothly, without causing a damage region to be redrawn.
**First, you would create the object by calling OG_Clone with a FALSE
**damage flag .  Then, you can make the object appear on the layout
**by calling OG_Draw.
*/

PROCEDURE clone_object IS
   the_object   OG_Object;
   new_object   OG_Object;
BEGIN
   the_object:=OG_Get_Object('My_Object');
   new_object:=OG_Clone(The_Object, FALSE);
   OG_Draw(new_object);
END;
```

OG_Export_Drawing (Display)

Description This procedure exports the entire layout (including hidden layers0 as a drawing.
Syntax

```
PROCEDURE OG_Export_Drawing
  (name          VARCHAR2,
   repository    NUMBER,
   format        NUMBER,
   compression   NUMBER    :=  OG_No_Icompression);
```

Parameters

name	Is the name to which the drawing will be exported. If the drawing is to be stored in the database, this argument should contain only the name of the drawing. If the drawing is to be stored in the file system, this argument should contain the absolute or relative pathname of the drawing file.
repository	Specifies whether the drawing is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the drawing is to be stored in the database. **OG_Filesystem** Means the drawing is to be stored in the file system.
format	Specifies the format in which the drawing is exported. The value of this argument may be one of the following built-in constants:

OG_Cgm16_Dformat Means the drawing is saved in the CGM 2-byte format.

OG_Cgm32_Dformat Means the drawing is saved in the CGM 4-byte format.

OG_Oracle_Dformat Means the drawing is saved in the Oracle Format, used by other Oracle products.

compression Is the type of compression used to compress images that are part of the drawing. The value of this argument may be one of the following built-in constants:

OG_No_Icompression Means images are not compressed.

OG_H3g_Icompression Means images are compressed using CCITT Group 3 with Huffman encoding compression

OG_G3fax_Icompression Means images are compressed using Group 3 Fax compression. This compression type is valid for monochrome images only.

OG_G4fax_Icompression Means images are compressed using Group 4 Fax compression. This compression type is valid for monochrome images only.

OG_Pack_Icompression Means images are compressed using PackBits compression. This compression type is valid for monochrome TIFF images only.

OG_Lzwhdiff_Icompression Means images are compressed using LZW compression with horizontal differencing.

OG_Lzwnohdiff_Icompression Means images are compressed using LZW compression without horizontal differencing.

OG_Jpeg_Lowest_Icompression Means images are compressed using JPEG compression resulting in the lowest compression ratio and highest quality.

OG_Jpeg_Low_Icompression Means images are compressed using JPEG compression resulting in a low compression ratio and high quality.

OG_Jpeg_Medium_Icompression Means images are compressed using JPEG compression resulting in a medium compression ratio and medium quality.

OG_Jpeg_High_Icompression Means images are compressed using JPEG compression resulting in a high compression ratio and low quality.

OG_Jpeg_Highest_Icompression Means images are compressed using JPEG compression resulting in the highest compression ratio and lowest quality.

OG_Export_Drawing (Display) Examples

```
/*Suppose you want to export the display contents to the CGM file
**`my_draw' so that you can later import it into some other application.
**The following procedure does this:
*/
PROCEDURE export_the_drawing IS
BEGIN
   OG_Export_Drawing('My_Draw', OG_Filesystem, OG_Cgm16_Dformat);
END;
```

OG_Export_Drawing (Object/Layer)

Description This procedure exports the specified object or layer as a drawing.

Syntax

```
PROCEDURE OG_Export_Drawing
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   object_hdl   OG_Object,
   compression  NUMBER     :=  OG_No_Icompression);
```

Parameters

name	Is the name to which the drawing will be exported. If the drawing is to be stored in the database, this argument should contain only the name of the drawing. If the drawing is to be stored in the file system, this argument should contain the absolute or relative pathname of the drawing file.
Repository	Specifies whether the drawing is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the drawing is to be stored in the database. **OG_Filesystem** Means the drawing is to be stored in the file system.
format	Specifies the format in which the drawing is exported. The value of this argument may be one of the following built-in constants: **OG_Cgm16_Dformat** Means the drawing is saved in the CGM 2-byte format. **OG_Cgm32_Dformat** Means the drawing is saved in the CGM 4-byte format. **OG_Oracle_Dformat** Means the drawing is saved in the Oracle Format, used by other Oracle products.
object_hdl	Is the handle to the object to be exported. The object can be either a group or the object handle to a layer; the object and all of its descendants will be exported. To specify a layer to export, use OG_Get_Object to specify

an object handle to the layer.

Compression Is the type of compression used to compress images that are part of the drawing. The value of this argument may be one of the following built-in constants:

OG_No_Icompression Means images are not compressed.

OG_H3g_Icompression Means images are compressed using CCITT Group 3 with Huffman encoding compression

OG_G3fax_Icompression Means images are compressed using Group 3 Fax compression. This compression type is valid for monochrome images only.

OG_G4fax_Icompression Means images are compressed using Group 4 Fax compression. This compression type is valid for monochrome images only.

OG_Pack_Icompression Means images are compressed using PackBits compression. This compression type is valid for monochrome TIFF images only.

OG_Lzwhdiff_Icompression Means images are compressed using LZW compression with horizontal differencing.

OG_Lzwnohdiff_Icompression Means images are compressed using LZW compression without horizontal differencing.

OG_Jpeg_Lowest_Icompression Means images are compressed using JPEG compression resulting in the lowest compression ratio and highest quality.

OG_Jpeg_Low_Icompression Means images are compressed using JPEG compression resulting in a low compression ratio and high quality.

OG_Jpeg_Medium_Icompression Means images are compressed using JPEG compression resulting in a medium compression ratio and medium quality.

OG_Jpeg_High_Icompression Means images are compressed using JPEG compression resulting in a high compression ratio and low quality.

OG_Jpeg_Highest_Icompression Means images are compressed using JPEG compression resulting in the highest compression ratio and lowest quality.

OG_Export_Drawing (Object/Layer) Examples

```
/* Suppose you want to export the contents of `layer0' to the CGM file
** `my_draw' so that you can later import it into some other application.
** The following procedure does this:
*/

PROCEDURE export_the_drawing IS
   the_layer   OG_Object;
BEGIN
   the_layer:=OG_Get_Object('Layer0');
   OG_Export_Drawing('My_Draw', OG_Filesystem,
OG_Cgm16_Dformat, the_layer);
END;
```

OG_Export_Drawing (Window)

Description This procedure exports the visible contents of the specified window as a drawing.

Syntax

```
PROCEDURE OG_Export_Drawing
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   window_hdl   OG_Window,
   compression  NUMBER     :=  OG_No_Icompression);
```

Parameters:

name	Is the name to which the drawing will be exported. If the drawing is to be stored in the database, this argument should contain only the name of the drawing. If the drawing is to be stored in the file system, this argument should contain the absolute or relative pathname of the drawing file.
Repository	Specifies whether the drawing is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the drawing is to be stored in the database. **OG_Filesystem** Means the drawing is to be stored in the file system.
format	Specifies the format in which the drawing is exported. The value of this argument may be one of the following built-in constants: **OG_Cgm16_Dformat** Means the drawing is saved in the CGM 2-byte format. **OG_Cgm32_Dformat** Means the drawing is saved in the CGM 4-byte format. **OG_Oracle_Dformat** Means the drawing is saved in the Oracle Format, used by other Oracle products.
window_hdl	Is the handle to the window that contains the drawing to be exported. All of the layers that

	are showing in the window will be exported.
Compression	Is the type of compression used to compress images that are part of the drawing. The value of this argument may be one of the following built-in constants: **OG_G3fax_Icompression** Means images are compressed using Group 3 Fax compression. This compression type is valid for monochrome images only. **OG_G4fax_Icompression** Means images are compressed using Group 4 Fax compression. This compression type is valid for monochrome images only. **OG_H3g_Icompression** Means images are compressed using CCITT Group 3 with Huffman encoding compression **OG_Jpeg_High_Icompression** Means images are compressed using JPEG compression resulting in a high compression ratio and low quality. **OG_Jpeg_Highest_Icompression** Means images are compressed using JPEG compression resulting in the highest compression ratio and lowest quality. **OG_Jpeg_Low_Icompression** Means images are compressed using JPEG compression resulting in a low compression ratio and high quality. **OG_Jpeg_Lowest_Icompression** Means images are compressed using JPEG compression resulting in the lowest compression ratio and highest quality. **OG_Jpeg_Medium_Icompression** Means images are compressed using JPEG compression resulting in a medium compression ratio and medium quality. **OG_Lzwhdiff_Icompression** Means images are compressed using LZW compression with horizontal differencing. **OG_Lzwnohdiff_Icompression** Means images are compressed using LZW compression without horizontal differencing. **OG_No_Icompression** Means images are not compressed. **OG_Pack_Icompression** Means images are compressed using PackBits compression. This compression type is valid for monochrome TIFF images only.

OG_Export_Drawing (Window) Examples

```
/* Suppose you want to export the contents of the `Main Layout'
** window to the CGM file `my_draw' so that you can later import it
** into some other application.  The following procedure does this:
*/

PROCEDURE export_the_drawing IS
   the_window   OG_Window;
BEGIN
   the_window:=OG_Get_Window('Main Layout');
   OG_Export_Drawing('My_Draw', OG_Filesystem,
OG_Cgm16_Dformat, the_window);
END;
```

OG_Export_Image

Description This procedure exports a Graphics Builder object, surrounded by a one-half inch border, to an image.

Syntax

```
PROCEDURE OG_Export_Image
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   image_hdl    OG_Object,
   compression  NUMBER     :=  OG_No_Icompression);
```

Parameters

name	Is the name to which the image will be exported. If the image is to be stored in the database, this argument should contain only the name of the image. If the image is to be stored in the file system, this argument should contain the absolute or relative pathname of the image file.
Repository	Specifies whether the image is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the image is to be stored in the database. **OG_Filesystem** Means the image is to be stored in the file system.
format	Specifies the format in which the image is to be exported. The value of this argument may be one of the following built-in constants: **OG_Bmp_Iformat** Means the image is saved in the Windows/OS2 Bitmap format. **OG_Cals_Iformat** Means the image is saved in the CALS Type 1 Raster format. **OG_Gif_Iformat** Means the image is saved in the CompuServe GIF format. You must compress GIF files using OG_Lzwhdiff_Icompression.

	OG_Jfif_Iformat Means the image is saved in the JPEG File Image Format.
	OG_Pict_Iformat Means the image is saved in the Macintosh PICT format.
	OG_Ras_Iformat Means the image is saved in the SUN Raster format.
	OG_Tiff_Iformat Means the image is saved in the Tag Image File Format.
image_hdl	Is the handle to the image object that will be exported. Can be any Graphics Builder object.
Compression	Is the type of compression used. The value of this argument may be one of the following built-in constants:
	OG_G3fax_Icompression Means the image is compressed using Group 3 Fax compression. This compression type is valid for monochrome images only.
	OG_G4fax_Icompression Means the image is compressed using Group 4 Fax compression. This compression type is valid for monochrome images only.
	OG_H3g_Icompression Means the image is compressed using CCITT Group 3 with Huffman encoding compression
	OG_Jpeg_High_Icompression Means the image is compressed using JPEG compression resulting in a high compression ratio and low quality.
	OG_Jpeg_Highest_Icompression Means the image is compressed using JPEG compression resulting in the highest compression ratio and lowest quality.
	OG_Jpeg_Low_Icompression Means the image is compressed using JPEG compression resulting in a low compression ratio and high quality.
	OG_Jpeg_Lowest_Icompression Means the image is compressed using JPEG compression resulting in the lowest compression ratio and highest quality.
	OG_Jpeg_Medium_Icompression Means the image is compressed using JPEG compression resulting in a medium compression ratio and medium quality.
	OG_Lzwhdiff_Icompression Means the image is compressed using LZW compression with horizontal differencing. You must use this type of compression on GIF files.
	OG_Lzwnohdiff_Icompression Means the image is compressed using LZW compression without horizontal differencing.
	OG_No_Icompression Means the image is not compressed.

OG_Pack_Icompression Means the image is compressed using Packbits compression. This compression type is valid for monochrome TIFF images only.

OG_Export_Image Examples

```
/* Suppose you want to export the image named `image0' to the TIFF file
**`my_image' so that you can later import it into some other application
**The following procedure does this:
*/

 PROCEDURE export_the_image IS
   the_image   OG_Object;
BEGIN
   the_image:=OG_Get_Object('image0');
   OG_Export_Image('my_image', OG_Filesystem,
      OG_Tiff_Iformat, the_image);
END;
```

OG_Get_Char_Property

Description This procedure gets the value of a user-defined CHAR property of an object.

Syntax

```
FUNCTION OG_Get_Char_Property
  (object_hdl  OG_Object,
   prop_name   VARCHAR2)
RETURN VARCHAR2;
```

Parameters

object_hdl	Is the handle to the object containing the property you want to get.
prop_name	Is the name of the property whose value you want to get.

Returns The value of the specified property.

OG_Get_Char_Property Examples

```
/*The following procedure gets the 'status' property
**in each child object in a group, and then changes
**the object’s color if the status is 'obsolete':
*/

PROCEDURE example(group_name VARCHAR2) IS
  group_obj    OG_Object;
  child_count  NUMBER;
  child_obj    OG_Object;
  stat         VARCHAR2(10);
BEGIN
  group_obj := OG_Get_Object(Group_Name);
  child_count := OG_Get_Childcount(Group_Obj);

  FOR i IN 0..child_count LOOP
    child_obj := OG_Get_Child(Group_Obj, i);
    stat := OG_Get_Char_Property(Child_Obj, 'status');
    IF stat = 'obsolete' THEN
      OG_Set_Fillcolor(Child_Obj, 'red');
    END IF;
  END LOOP;

END;
```

OG_Get_Child

Description This function returns a handle to a child object within a group object.
Syntax

```
FUNCTION OG_Get_Child
 (group_hdl  OG_Object,
  indx       NUMBER)
RETURN OG_Object;
```

Parameters

group_hdl	Is the handle to the group object containing the child.
indx	Is the index of the object in the group's child list whose handle should be returned.

Returns A handle to the specified child object within a group.

OG_Get_Child Examples

```
/*Suppose you have a several objects representing products
**in a warehouse, and you want to move one of the products
**from one warehouse to another.  Your display may use a group
**comprised of the products to represent the inventory for each
**warehouse.  To move a product from one warehouse to another,
**you would want to get the handle to the product object, delete it
**from one warehouse group, and add it to another warehouse group.
*/

PROCEDURE move_prod(warehouse1 IN OG_Object, warehouse2 IN
OG_Object, prod_index in number) IS
   the_prod   OG_Object;
BEGIN
   the_prod:=OG_Get_Child(Warehouse1, prod_index);
   OG_Delete_Child(Warehouse1, prod_index, 1);
   OG_Insert_Child(Warehouse2, the_prod, OG_Last);
END;
 /*Note that this procedure changes only the internal composition
**of the group objects.  To move or change the appearance of the
**product object, you must use other Graphics Builder built-in procedures.
*/
```

OG_Get_Cmptext

Description This procedure gets the attribute values of the specified compound text element and assigns them to the corresponding fields in the specified compound text attribute record. As described in "Text Attributes," a compound text element represents one line of text in a text object.
Syntax

```
PROCEDURE OG_Get_Cmptext
  (text_hdl          OG_Object,
   indx              NUMBER,
   attr      IN OUT  OG_Cmptext_Attr);
```

Parameters

text_hdl	Is the handle to the text object.
indx	Is the index of the compound text element in the compound text element list whose attributes you want to retrieve.
attr	Is the compound text attribute record that will

receive the compound text element's attributes.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to OG_Get_Cmptext.

OG_Get_Cmptext Examples

```
/* Suppose you want to determine how many simple text elements
**compose the first compound text element within a text object.
**Knowing this, you can loop through and examine each simple text element.
*/

FUNCTION how_many(my_text IN OG_Object) RETURN NUMBER IS
   ctext_rec   OG_Cmptext_Attr;
BEGIN
   ctext_rec.mask:=OG_Stcount_Cmptexta;
   OG_Get_Cmptxt(My_Text, 0, ctext_rec);
   RETURN(ctext_rec.stcount);
END;
```

OG_Get_Date_Property

Description This procedure gets the value of a user-defined DATE property of an object.

Syntax

```
FUNCTION OG_Get_Date_Property
  (object_hdl  OG_Object,
   prop_name   VARCHAR2,
   date_fmt    VARCHAR2   := 'DD-MON-YY')
RETURN DATE;
```

Parameters

object_hdl	Is the handle to the object containing the property you want to get.
prop_name	Is the name of the property whose value you want to get.
date_fmt	Is the date format mask used to set the date property with OG_Set_Property.

Returns The value of the specified property.

OG_Get_Date_Property Examples

```
/*The following procedure gets the 'due_date' property in each child object
**in a group, and then changes the object's color if the due date has past:
*/
PROCEDURE example(group_name VARCHAR2) IS
  group_obj    OG_Object;
  child_count  NUMBER;
  child_obj    OG_Object;
  due          DATE;
BEGIN
  group_obj := OG_Get_Object(Group_Name);
  child_count := OG_Get_Childcount(Group_Obj);

  FOR i IN 0..child_count-1 LOOP
    child_obj := OG_Get_Child(Group_Obj, i);
    due := OG_Get_Date_Property(Child_Obj, 'due_date');
    IF due < sysdate THEN
      OG_Set_Fillcolor(Child_Obj, 'red');
    END IF;
  END LOOP;
```

```
END;
```

OG_Get_Num_Property

Description This procedure gets the value of a user-defined NUMBER property of an object.

Syntax

```
FUNCTION OG_Get_Num_Property
  (object_hdl  OG_Object,
   prop_name   VARCHAR2)
RETURN NUMBER;
```

Parameters

object_hdl	Is the handle to the object containing the property you want to get.
prop_name	Is the name of the property whose value you want to get.

Returns The value of the specified property.

OG_Get_Num_Property Examples

```
/* The following procedure gets the 'priority' property in each child object
**in a group, and then sets the priority to one greater than its current value:
*/

PROCEDURE example(group_name VARCHAR2) IS
  group_obj    OG_Object;
  child_count  NUMBER;
  child_obj    OG_Object;
  current_p    NUMBER;
BEGIN
  group_obj := OG_Get_Object(Group_Name);
  child_count := OG_Get_Childcount(Group_Obj);

  FOR i IN 0..child_count-1 LOOP
    child_obj := OG_Get_Child(Group_Obj, i);
    current_p := OG_Get_Num_Property(Child_Obj, 'priority');
    OG_Set_Property(Child_Obj, 'priority', current_p + 1);
  END LOOP;

END;
```

OG_Get_Object

Description The object may be an arc, chart, group, image, line, polygon, rectangle, rounded rectangle, symbol, or text object. The object must be created and named either in the Builder or programmatically prior to retrieving it with this function. If the specified object does not exist, this function will return a null handle.

Syntax

```
FUNCTION OG_Get_Object
  (object_name  VARCHAR2)
RETURN OG_Object;
FUNCTION OG_Get_Object
  (object_name  VARCHAR2,
   root_hdl     OG_Object)
RETURN OG_Object;
```

Parameters

object_name	Is the name of the object whose handle should

	be returned. **Note:** OBJECT_NAME is case-sensitive.
root_hdl	Is the handle to the object in whose group tree you want to search.

Returns A handle to the specified object.
Usage Notes If you do not specify *root_hdl*, Graphics Builder will begin the search with the display's actual root object, thus searching every object in the display for the one whose name is specified. If you do specify *root_hdl*, Graphics Builder will search only in the group tree below that object for the named object. You will get unpredictable results if multiple objects in the search path have the same name.
If *object_name* is the name of a layer, this function treats it as a group object and returns a handle to that group. You can then use the group-related subprograms (OG_Insert_Child, OG_Delete_Child, etc.) to manipulate the objects on the layer.

OG_Get_Object Examples

```
/* Suppose you have a map of the world and you want to change
**the color of one of the countries.  First, you would get the handle
**to the country object, then you would change its color.
*/

PROCEDURE color_country(country_name) IS
   my_object    OG_Object;
   obj_record   OG_Graphic_Ca;
BEGIN
   my_object:=OG_Get_Object(Country_Name);
   obj_record.graphic_caob.mask:=OG_None_Generica;
   obj_record.graphic_caoh.mask:=OG_Ffcolor_Graphica;
   obj_record.graphic_caoh.ffcolor:='red';
   OG_Set_Attr(My_Object, obj_record);
END;
```

OG_Get_Point

Description This function returns a record containing the x- and y-coordinates of a point in the specified object.
Syntax
```
FUNCTION OG_Get_Point
  (object_hdl  OG_Object,
   indx        NUMBER
   rotated     BOOLEAN    :=  FALSE)
RETURN OG_Point;
```

Parameters

object_hdl	Is the handle to the object.
indx	Is the index of the point in the point list to be returned.
rotated	Specifies whether the point returned should reflect any rotation angle applied to the object.

Returns The location of the specified point.
Usage Notes
Polygon: Returns *indx*'th point of the object.
Arc, Chart, Rectangle, Rounded Rectangle: 0 index returns top-left corner; 1 index returns top-right corner; 2 index returns bottom-right corner; 3 index returns bottom-left corner.
Text: 0 index returns top-left corner; 1 index returns top-right corner; 2 index returns bottom-right corner; 3 index returns bottom-left corner.
Line: 0 index returns start point; 1 index returns end point

Image, Group, Symbol: Does not apply.
If the object was created in the Builder, the initial index values for the points will correspond to the order in which the mouse was selected when the object was drawn (with the first point having an index of 0). Index values for points that were inserted programmatically will depend on the index that was specified when they were inserted.

OG_Get_Point Examples

```
/* Suppose you have several polygons on a map, each of which connects
**the cities within a specific distribution area.  If a city is transferred from one
**distribution area to another, you would want to get a handle to the point
**representing that city, delete it from one polygon, and add it to another polygon.
*/

PROCEDURE move_city(area1 IN OG_Object, area2 IN OG_Object,
  city_index NUMBER) IS
   the_city   OG_Point;
BEGIN
   the_city:=OG_Get_Point(Area1, city_index);
   OG_Delete_Point(Area1, city_index, 1);
   OG_Insert_Point(Area2, OG_Last, the_city);
END;
```

OG_Get_Smptext

Description This procedure gets the attribute values of the specified simple text element within the specified compound text element and the specified text object. These attributes are then assigned to the corresponding fields in the specified simple text attribute record. As described in "Text Attributes," a simple text element represents a text string in a compound text element.

Syntax

```
PROCEDURE OG_Get_Smptext
  (text_hdl          OG_Object,
   cmpindex          NUMBER,
   smpindex          NUMBER,
   attr      IN OUT  OG_Smptext_Attr);
```

Parameters

text_hdl	Is the handle to the text object.
Cmpindx	Is the index within the text object of the compound text element that contains the simple text element whose attributes should be retrieved.
Smpindex	Is the index within the compound text element of the simple text element whose attributes should be retrieved.
attr	Is the simple text attribute record that will receive the simple text element's attributes.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to OG_Get_Smptext.

OG_Get_Smptext Examples

```
/* Suppose you have created a message text object.  To change the
**message it contains, you would delete the simple text element containing
**the current message and insert a new simple text element containing the
**new message.  To maintain the font and other attributes, however,
**you first would want to get the simple text element into an attribute record.
```

```
**That way you could modify only the text string, and leave the other attribute
**settings (such as font) unchanged.
*/

PROCEDURE put_msg (mess IN VARCHAR2) IS
   msgobj   OG_Object;
   msgrec   OG_Smptext_Attr;
BEGIN
   msgobj:=OG_Get_Object('Msg');
   msgrec.mask:=OG_Font_Smptexta+
                OG_Color_Smptexta;
   msgrec.font.mask:=OG_All_Fonta;
   OG_Get_Smptext(Msgobj, 0, 0, msgrec);
   OG_Delete_Smptext(Msgobj, 0, 0, 1, FALSE);
   msgrec.mask:=msgrec.mask + OG_Str_Smptexta;
   msgrec.str:=mess;
   OG_Insert_Smptext(Msgobj, msgrec, 0, OG_Last);
END;
```

OG_Import_Drawing

Description This procedure imports a drawing. It returns a handle to the first object in the drawing.

Syntax

```
FUNCTION OG_Import_Drawing
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   use_colors   BOOLEAN,
   damage       BOOLEAN   :=  TRUE,
   update_bbox  BOOLEAN   :=  TRUE)
RETURN OG_Object;
FUNCTION OG_Import_Drawing
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   use_colors   BOOLEAN,
   parent_hdl   OG_Object,
   damage       BOOLEAN   :=  TRUE,
   update_bbox  BOOLEAN   :=  TRUE)
RETURN OG_Object;
```

Parameters

name	Is the name of the drawing. If the drawing is stored in the database, this argument should contain only the name of the drawing. If the drawing is stored in the file system, this argument should contain the absolute or relative pathname of the drawing file.
Repository	Specifies whether the drawing is stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the drawing is stored in the database. **OG_Filesystem** Means the drawing is stored in the file system.
format	Specifies the format in which the drawing is saved. The value of this argument may be one of the following built-in constants: **OG_Cgm_Dformat** Means the drawing is saved in the CGM format (either 2-byte or 4-

	byte). **OG_Oracle_Dformat** Means the drawing is saved in the Oracle Format, used by other Oracle products.
use_colors	Specifies whether the drawing's color palette should be used. If this argument is TRUE, the drawing's palette will be used. If FALSE, the default Graphics Builder color palette will be used.
parent_hdl	Is the handle to a group object into which the imported drawing should be inserted as a child. If this argument is not supplied, the drawing is inserted as a child of the root object (although it is *not* recognized as a layer until you activate or show it).
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Returns A handle to the imported drawing.

OG_Import_Drawing Examples

```
/* Suppose you want to import the contents of the CGM file `my_draw'
**onto the layer `layer0'.  You can get the object handle to the layer,
**then use that for parent_hdl.  The following procedure does this:
*/

PROCEDURE import_the_drawing IS
   the_layer   OG_Object;
   dummy   OG_Object;
BEGIN
   the_layer:=OG_Get_Object('Layer0');
   dummy:=OG_Import_Drawing('My_Draw', OG_Filesystem,
OG_Cgm16_Dformat, FALSE, the_layer);
END;
```

OG_Import_Image

Description This procedure imports an image onto the active layer in the active window.

Syntax

```
FUNCTION OG_Import_Image
  (name         VARCHAR2,
   repository   NUMBER,
   format       NUMBER,
   damage       BOOLEAN   :=  TRUE,
   update_bbox  BOOLEAN   :=  TRUE)
RETURN OG_Object;
```

Parameters

name	Is the name of the image. If the image is stored in the database, this argument should contain only the name of the image. If the image is stored in the file system, this argument should contain the absolute or relative pathname of the image file.
Repository	Specifies whether the image is stored in the file system or database. The value of this argument may be one of the following built-in

	constants: **OG_Db** Means the image is stored in the database. **OG_Filesystem** Means the image is stored in the file system.
format	Specifies the format in which the image is saved. The value of this argument may be one of the following built-in constants: **OG_Any_Iformat** Means Graphics Builder automatically determines the image's format. **Note:** Specify this format if your image was exported in the Oracle Format (now obsolete). **OG_Bmp_Iformat** Means the image is saved in the BMP format. **OG_Cals_Iformat** Means the image is saved in the CALS format. **OG_Gif_Iformat** Means the image is saved in the GIF format. **OG_Jfif_Iformat** Means the image is saved in the JFIF format. **OG_Oracle_Sformat** Means the image is saved in the Oracle Format, used by other Oracle products. **OG_Pcd_Iformat** Means the image is saved in the PCD format. **OG_Pcx_Iformat** Means the image is saved in the PCX format. **OG_Pict_Iformat** Means the image is saved in the PICT format. **OG_Ras_Iformat** Means the image is saved in the Sun raster format. **OG_Tiff_Iformat** Means the image is saved in the TIFF format.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Returns A handle to the imported image.

OG_Import_Image Examples

```
/* Suppose you want to import the contents of the TIFF file `my_image'
**onto the layer `layer0'.  The following procedure does this:
*/

PROCEDURE import_the_image IS
   the_image   OG_Object;
BEGIN
   OG_Activate_Layer(OG_Get_Layer('Layer0'));
   the_image:=OG_Import_Image('My_Image', OG_Filesystem,
     OG_Tiff_Iformat);
END;
```

OG_Insert_Child

Description This procedure inserts a child object into the specified group object. If the object to be inserted is already a child of another group object, Graphics Builder will first automatically delete the child from its current parent.

Syntax

```
PROCEDURE OG_Insert_Child
  (group_hdl    OG_Object,
   child_hdl    OG_Object,
   indx         NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

group_hdl	Is the handle to the group object into which the child should be inserted.
child_hdl	Is the handle to the object that should be inserted as a child.
indx	Is the index in the group's child list at which to insert the new child. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of children in the group prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the object as the first child in the group (index = 0). **OG_Last** Means insert the object as the last child in the group (index = the number of children in the group prior to the insertion).
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes You can insert an object into a layer by treating the layer as a group object, and passing its object handle to this procedure. You can also create a new layer by inserting a group object into the display's root object. However, Graphics Builder will not recognize that group as a layer until you explicitly show it by calling OG_Activate_Layer or OG_Show_Layer.
Note that Graphics Builder does not check for loops in group objects, which would result it you insert a group as a child of one of its descendants. If this occurs, Graphics Builder will enter an infinite loop when it traverses the group tree to update the bounding boxes of the affected objects.
Note that this procedure changes only the internal composition of the group objects. To move or change the appearance of the product object, you must use other Graphics Builder built-in procedures.

OG_Insert_Child Examples

```
/* Suppose you have a several objects representing products in a warehouse,
**and you want to move one of the products from one warehouse to another.
**Your display may use a group comprised of the products to represent the
**inventory for each warehouse.  To move a product from one warehouse to
**another, you would want to get the handle to the product object, delete it
**from one warehouse group, and add it to another warehouse group.
*/

PROCEDURE move_prod (warehouse1 IN OG_Object, warehouse2 IN
  OG_Object, prod_index IN number) IS
   the_prod   OG_Object;
BEGIN
   the_prod:=OG_Get_Child(Warehouse1, prod_index);
   OG_Delete_Child(Warehouse1, prod_index, 1);
   OG_Insert_Child(Warehouse2, the_prod, OG_Last);
END;
```

OG_Insert_Cmptext

Description This procedure inserts a new compound text element into the specified text object. As described in "Text Attributes," a compound text element represents one line of text in a text object.

Syntax

```
PROCEDURE OG_Insert_Cmptext
  (text_hdl     OG_Object,
   indx         NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text_hdl	Is the handle to the text object.
indx	Is the index in the compound text element list within the text object at which to insert the new compound text element. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of compound text elements in the text object prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new compound text element at the beginning of the text object (index = 0). **OG_Last** Means insert the new compound text element at the end of the text object (index = the number of compound text elements in the text object prior to the insertion).
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes The attributes of a compound text element are set automatically by Graphics Builder. Therefore, you do not need to provide a compound text attribute record when creating new compound text. (In constrast, if you want to *get* the element's attributes, you must provide OG_Get_Cmptext with a compound text attribute record to receive them.)

OG_Insert_Cmptext Examples

```
/* Suppose you want to create a text object that contains a message for
**the user.  The following function will create the object, insert a compound text
**element, then insert a simple text element that contains the text of the message.
*/

PROCEDURE make_text (the_message IN VARCHAR2) IS
   text_rec   OG_Text_Ca;
   text_obj   OG_Object;
   smp_rec    OG_Smptext_Attr;
BEGIN
   text_rec.text_caob.mask:=OG_None_Generica;
   text_rec.text_caoh.mask:=OG_None_Graphica;
   text_rec.text_caot.mask:=OG_None_Texta;
   text_obj:=OG_Make(Text_Rec);
   OG_Insert_Cmptext(Text_Obj, OG_Last);
   smp_rec.mask:=OG_Str_Smptexta;
   smp_rec.str:=the_message;
   OG_Insert_Smptext(Text_Obj, smp_rec, 0, OG_Last);
END;
```

OG_Insert_Point

Description This procedure inserts a new point into the specified polygon object.

Syntax

```
PROCEDURE OG_Insert_Point
  (poly_hdl     OG_Object,
   indx         NUMBER,
   pt           OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

poly_hdl	Is the handle to the polygon or polyline object.
indx	Is the index in the point list at which to insert the new point. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of points in the object prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new point at the beginning of the object's point list (index = 0). **OG_Last** Means insert the new point at the end of the object's point list (index = the number of compound text elements in the text object prior to the insertion).
pt	Is the record containing the x- and y-coordinates of the point to be inserted.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes If the object was created in the Builder, the initial index values for the points will correspond to the order in which the mouse was selected when the object was drawn (with the first point having an index of 0). Index values for points that were inserted programmatically will depend on the index that was specified when they were inserted.

OG_Insert_Point Examples

```
/* Suppose you have several polygons on a map, each of which connects
**the cities within a specific distribution area.  If a city is transferred from one
**distribution area to another, you would want to get a handle to the point
**representing that city, delete it from one polygon, and add it to another polygon.
*/

PROCEDURE move_city(area1 IN OG_Object, area2 IN OG_Object,
city_index IN NUMBER) IS
   the_city   OG_Point;
BEGIN
   the_city:=OG_Get_Point(Area1, city_index);
   OG_Delete_Point(Area1, city_index, 1);
   OG_Insert_Point(Area2, OG_Last, the_city);
```

```
END;
```

OG_Insert_Smptext

Description This procedure inserts a new simple text element into the specified compound text element within the specified text object. As described in "Text Attributes," a simple text element represents a text string in a compound text element.

Syntax

```
PROCEDURE OG_Insert_Smptext
  (textobj      OG_Object,
   smp_attr     OG_Smptext_Attr,
   cmpindex     NUMBER,
   smpindex     NUMBER,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
```

Parameters

text_hdl	Is the handle to the text object.
smp_attr	Is the attribute record for the simple text element to be inserted.
Cmpindex	Is the index of the compound text element within the text object into which the simple text should be inserted.
Smpindex	Is the index within the compound text element at which the new simple text element should be inserted. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of simple text elements in the compound text element prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new simple text element at the beginning of the compound text element (index = 0). **OG_Last** Means insert the new simple text element at the end of the compound text element (index = the number of simple text elements in the compound text element prior to the insertion).
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes The specified simple text element attribute record (*smp_attr*) contains the attributes for the simple text element, including the text string. The only attribute values that will be set are those specified by the value of the *mask* attribute in that attribute record.

OG_Insert_Smptext Examples

```
/* Suppose you want to create a text object that contains a message
**for the user.  The following procedure will create the object, insert a
**compound text element, then insert a simple text element that contains
**the text of the message.
*/

PROCEDURE make_text (the_message IN VARCHAR2) IS
   text_rec   OG_Text_Ca;
   text_obj   OG_Object;
   smp_rec    OG_Smptext_Attr;
BEGIN
   text_rec.text_caob.mask:=OG_None_Generica;
   text_rec.text_caoh.mask:=OG_None_Graphica;
   text_rec.text_caot.mask:=OG_None_Texta;
   text_obj:=OG_Make(Text_Rec);
   OG_Insert_Cmptext(Text_Obj, OG_Last);
   smp_rec.mask:=OG_Str_Smptxta;
   smp_rec.str:=the_message;
   OG_Insert_Smptext(Text_Obj, smp_rec, 0, OG_Last);
END;
```

OG_Make_Ellipse

Description This function creates an ellipse.

Syntax

```
FUNCTION OG_Make_Ellipse
  (position  OG_Point,
   height    NUMBER,
   width     NUMBER)
RETURN OG_Object;
```

Parameters

position	The x- and y-coordinates of the ellipse.
height	The ellipse height.
width	The ellipse width.

Returns A handle to the newly created ellipse.

OG_Make_Ellipse Examples

```
/* The following function creates an ellipse:
*/

FUNCTION example RETURN OG_Object IS
  object  OG_Object;
  pos     OG_Point;
  height  NUMBER;
  width   NUMBER;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;
  height := 4* OG_Inch;
  width := 4* OG_Inch;

  object := OG_Make_Ellipse(Pos, height, width);
  RETURN(object);
END;
```

OG_Make_Group

Description This function creates a group object (note that the group will be empty until you add children to it using OG_Insert_Child).

Syntax

```
FUNCTION OG_Make_Group
RETURN OG_Object;
```

Parameters:

None.

Returns A handle to the newly created group.

OG_Make_Group Examples

```
/* The following function creates a group with the specified name:
*/

FUNCTION example(group_name VARCHAR2) RETURN OG_Object IS
  object  OG_Object;
BEGIN
  object := OG_Make_Group;
  OG_Set_Name(Object, group_name);
  RETURN(object);
END;
```

OG_Make_Image

Description This function creates an image from data stored in a database table.

Syntax

```
FUNCTION OG_Make_Image
  (query       OG_Query,
   which_data  NUMBER,
   colname     VARCHAR2)
RETURN OG_Object;
```

Parameters

query	Is the handle to the query that retrieves the image from a table in a database. Note that this table must be a user table, and not one the private tables used by Graphics Builder when you save or export a module to the database.
which_data	Specifies whether the image to be created is contained in a query's new or old data set. This value may be one of the following built-in constants: **OG_Newdata** Means the image is contained in the query's new data set. **OG_Olddata** Means the image is contained in the query's old data set.
Colname	Is the name of the query column that contains the image data. The image that is created is the one contained in the query cell at the intersection of the column specified by this attribute and the row pointed to by the query's cursor.

Returns A handle to the newly created image.

OG_Make_Image Examples

```
/* The following function creates an image from data in the sixth
**row of the query 'image_query' in the column IMAGE_COLUMN:
*/

FUNCTION example(image_name VARCHAR2) RETURN OG_Object IS
  query   OG_Query;
  object  OG_Object;
BEGIN
  query := OG_Get_Query('Image_Query');
  OG_Execute_Query(Query);
  OG_Start_From(Query, OG_Newdata, 5);
  object := OG_Make_Image(Query, OG_Newdata, 'IMAGE_COLUMN');
  OG_Set_Name(Object, image_name);
  RETURN(object);
END;
```

OG_Make_Line

Description This function creates a line.

Syntax

```
FUNCTION OG_Make_Line
  (startpt  OG_Point,
   endpt    OG_Point)
RETURN OG_Object;
```

Parameters

startpt	Is the starting point of the line (in layout units).
endpt	Is the ending point of the line (in layout units).

Returns A handle to the newly created line.

OG_Make_Line Examples

```
/* The following function creates a 2" horizontal line:
*/

FUNCTION example RETURN OG_Object IS
  object   OG_Object;
  startpt  OG_Point;
  endpt    OG_Point;
BEGIN
  startpt.x := OG_Inch;
  startpt.y := OG_Inch;
  endpt.x := 2 * OG_Inch;
  endpt.y := OG_Inch;

  object := OG_Make_Line(Startpt, endpt);
  RETURN(object);
END;
```

OG_Make_Poly

Description This function creates a polygon/polyline object (note that the object will contain no points until you add them using OG_Insert_Point).

Syntax

```
FUNCTION OG_Make_Poly
RETURN OG_Object;
```

Parameters:

None.

Returns A handle to the newly created polygon/polyline object.

OG_Make_Poly Examples

```
/* The following function creates a polygon with the specified name:
*/

FUNCTION example(poly_name VARCHAR2) RETURN OG_Object IS
  object  OG_Object;
BEGIN
```

```
  object := OG_Make_Poly;
  OG_Set_Name(Object, poly_name);
  RETURN(object);
END;
```

OG_Make_Rect

Description This function creates a rectangle.

Syntax

```
FUNCTION OG_Make_Rect
  (position  OG_Point,
   height    NUMBER,
   width     NUMBER)
RETURN OG_Object;
```

Parameters

position	Is the x- and y-coordinates of the rectangle.
height	Is the height of the rectangle.
width	Is the width of the rectangle.

Returns A handle to the newly created rectangle.

OG_Make_Rect Examples

```
/* The following function creates a rectangle:
*/

FUNCTION example RETURN OG_Object IS
  object  OG_Object;
  pos     OG_Point;
  height  NUMBER;
  width   NUMBER;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;
  height := 4* OG_Inch;
  width := 4* OG_Inch;

  object := OG_Make_Rect(Pos, height, width);
  RETURN(object);
END;
```

OG_Make_Rrect

Description This function creates a rounded rectangle.

Syntax

```
FUNCTION OG_Make_Rrect
  (position  OG_Point,
   height    NUMBER,
   width     NUMBER)
RETURN OG_Object;
```

Parameters

position	Is the x- and y-coordinates of the rounded rectangle.
height	Is the height of the rounded rectangle.
width	Is the width of the rounded rectangle.

Returns A handle to the newly created rounded rectangle.

OG_Make_Rrect Examples

```
/* The following function creates a rounded rectangle:
*/

FUNCTION example RETURN OG_Object IS
  object  OG_Object;
  pos     OG_Point;
  height  NUMBER;
  width   NUMBER;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;
  height := 4* OG_Inch;
  width := 4* OG_Inch;

  object := OG_Make_Rrect(Pos, height, width);
  RETURN(object);
END;
```

OG_Make_Symbol

Description This function creates a symbol.

Syntax

```
FUNCTION OG_Make_Symbol
  (position  OG_Point,
   indx      NUMBER,
   symsize   NUMBER)
RETURN OG_Object;
```

Parameters

position	Is the symbol's center point.
indx	Is the index (or number) of the symbol's position as it appears in the symbol palette in the Builder.
Symsize	Is the symbol's size. The value of this property may be one of the following built-in constants: **OG_Large_Symsize** **OG_Medium_Symsize** **OG_Small_Symsize**

Returns A handle to the newly created symbol.

OG_Make_Symbol Examples

```
/* The following function creates a symbol:
*/

FUNCTION example RETURN OG_Object IS
  symbol  OG_Object;
  pos     OG_Point;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;

  symbol := OG_Make_Symbol(Pos, 5, OG_Large_Symsize);
  RETURN(symbol);

END;
```

OG_Make_Text

Description This function creates a text object.

Syntax

```
FUNCTION OG_Make_Text
  (position  OG_Point,
RETURN OG_Object;
OG_Make_Text
  (position  OG_Point,
   string    VARCHAR2)
RETURN OG_Object;
OG_Make_Text
  (position  OG_Point,
   string    VARCHAR2,
   ptsize    NUMBER)
RETURN OG_Object;
```

Parameters

position	Is the x- and y-coordinates of the text object.
string	Is the text string.
ptsize	Is the point size.

Returns A handle to the newly created text object.

OG_Make_Text Examples

```
/* The following function creates a text object:
*/

FUNCTION example RETURN OG_Object IS
  object  OG_Object;
  pos     OG_Point;
BEGIN
  pos.x := OG_Inch;
  pos.y := OG_Inch;
  object := OG_Make_Text(Pos, 'Sales Rankings');
  RETURN(object);
END;
```

OG_Move

Description This procedure moves the specified object to another position on the layout.

Syntax

```
PROCEDURE OG_Move
  (object_hdl   OG_Object,
   offset       OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object_hdl	Is the handle to the object to move.
offset	Is the relative distance that the object should be moved from its current position.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes Note that you are able to move the object off the layout by specifying an offset that results in negative coordinates.
Positive values for the x- and y- components of *offset* will move the object to the right and down. Negative values will move the object to the left and up.

To move an object to an *absolute* position on the layout, set the new position in its appropriate attribute record.

OG_Move Examples

```
/* Suppose you have an object that represents inventory in a warehouse.
**If the inventory is moved from one warehouse to another, you would
**want to move the object that represents it.
*/

PROCEDURE move_inventory(invent_obj IN OG_Object) IS
   distance   OG_Point;
BEGIN
   distance.x:=(3*OG_Inch);
   distance.y:=(3*OG_Inch);
   OG_Move(Invent_Obj, distance);
END;
```

OG_Point_In

Description This function determines if the specified reference point lies within the fill region of an object.

Syntax

```
FUNCTION OG_Point_In
  (object_hdl  OG_Object,
   ref_pt      OG_Point,
   aperture    OG_Point)
RETURN OG_Object;
FUNCTION OG_Point_In
  (window_hdl  OG_Window,
   ref_pt      OG_Point,
   aperture    OG_Point)
RETURN OG_Object;
```

Parameters

object_hdl	Is the handle to the object to be checked.
window_hdl	Is the handle to the window to be checked.
ref_pt	Is the reference point.
Aperture	Is the maximum acceptable distance from the reference point (used only if object_hdl has a transparent fill).

Returns If the specified reference point lies within the fill region of an object, the function returns the handle to that object. If the point does not lie within an object's fill region, the function returns a null handle.

Usage Notes This function is most useful when monitoring a user's mouse activity. For example, you could write a button procedure for a group object and use the event record in the procedure header to determine the position of the mouse when it is selected or moved. Then you could call OG_Point_In or OG_Point_Near and pass in the group object and the mouse coordinates as arguments. The function will then return the single object in the group the user selected.

Note that this function determines only if the point lies within the *fill* region of an object. If the point lies exactly on an object's edge, this function returns a null handle. (This function assumes the object's edge to be the minimum possible thickness; the area covered by a thick edge is ignored.) If the object has a transparent fill pattern (and therefore no fill region), this function defaults to the behavior of OG_Point_Near. Note that the argument *aperture* is not used by OG_Point_In, but is passed on to OG_Point_Near if the object has a transparent fill pattern.

If a single object is specified as an argument, this function will check if the point lies within that object. If a group object is specified, this function will check each member of the group (and subgroups) and return the top-most single object that the point lies within (or a null handle, if the point does not lie within any object in the group). Note that a layer may be passed in as a group object. Similarly, if a window is specified instead of an object, this function will check each object in the window.
If the point lies within more than one object, the object that is highest in its group tree is returned.

OG_Point_In Examples

```
/* Suppose your application allows the user to select an object and drag it on
**top of other objects that are within a group.  When the user releases the mouse
**button, you want to determine which object the mouse is pointing to, and destroy it.
**The following procedure could be used as a button procedure for the object that was
dragged.
*/

PROCEDURE destroy_target (hitobj IN OG_Object, buttonobj IN
OG_Object, win IN OG_Window, eventinfo IN OG_Event) IS
   the_group    OG_Object;
   aper         OG_Point;
   target_obj   OG_Object;
BEGIN
   IF eventinfo.event_type=OG_Mouse_Up THEN
      the_group:=OG_Get_Object('Big_Group');
      aper.x:=3*OG_App.HSCREEN_RES;   /* three pixels */
      aper.y:=3*OG_App.VSCREEN_RES;   /* three pixels */
      target_obj:=OG_Point_In(The_Group,
        eventinfo.mouse_position, aper);
      IF not(OG_Isnull(Target_Obj)) THEN
         OG_Destroy(Target_Obj);
      END IF;
   END IF;
END;
```

OG_Point_Near

Description This function determines if the specified reference point lies along an object's edge.
Syntax

```
FUNCTION OG_Point_Near
  (object_hdl  OG_Object,
   ref_pt      OG_Point,
   aperture    OG_Point)
RETURN OG_Object;
```

```
FUNCTION OG_Point_Near
  (window_hdl  OG_Window,
   ref_pt      OG_Point,
   aperture    OG_Point)
RETURN OG_Object;
```

Parameters

object_hdl	Is the handle to the object to be checked.
window_hdl	Is the handle to the window to be checked.
ref_pt	Is the reference point.
Aperture	Is the maximum acceptable distance from the reference point.

Returns If the specified reference point lies along an object's edge, the function returns the handle to that object. If the point does not lie on an object's edge, the function returns a null handle.
Usage Notes This function is most useful when monitoring a user's mouse activity. For example, you could write a button procedure for a group object and use the event record in the procedure header to determine the position of the mouse when it is selected or moved. Then you could call OG_Point_In or

OG_Point_Near and pass in the group object and the mouse coordinates as arguments. The function will then return the single object the user selected.
Note that if the object has a transparent edge pattern, this function returns a null handle.
If a single object is specified as an argument, this function will check if the point lies along the edge of that object. If a group object is specified, this function will check each member of the group (and subgroups) and return the single object whose edge the point lies on (or a null handle, if the point does not lie along the edge of any object in the group). Note that a layer may be passed in as a group object. Similarly, if a window is specified instead of an object, this function will check each object in the window.

The argument *aperture* specifies the maximum distance that the reference point can be from an object's edge and still return the handle to that object. If the area specified by the aperture extends onto the object's edge, OG_Point_Near will return the handle to the object. Note that the aperture has both an x- and a y-component.
If the point lies within more than one object, the object that is highest in its group tree is returned.
Typically, you will use the aperture to give the user a margin of error, allowing for imprecise positioning of the mouse. In this situation, you would set both the x- and y-components of the aperture to the same value, possibly the equivalent of three screen pixels.

OG_Point_Near Examples

```
/* Suppose your application allows the user to select an object and drag it
**to the edge of other objects that are within a group.  When the user releases
**the mouse button, you want to determine which object's edge the mouse
**is pointing to, and destroy it. The following procedure could be used as a
**button procedure for the object that was dragged.
*/

PROCEDURE destroy_target (hitobj IN OG_Object, buttonobj IN
  OG_Object, win IN OG_Window, eventinfo IN OG_Event) IS
   the_group    OG_Object;
   aper         OG_Point;
   target_obj   OG_Object;
BEGIN
   IF eventinfo.event_type=OG_Mouse_Up THEN
      the_group:=OG_Get_Object('Big_Group');
      aper.x:=3*OG_App.HSCREEN_RES;   /* three pixels */
      aper.y:=3*OG_App.VSCREEN_RES;   /* three pixels */
      target_obj:=OG_Point_Near(The_Group,
        eventinfo.mouse_position, aper);
      IF not(OG_Isnull(Target_Obj)) THEN
         OG_Destroy(Target_Obj);
      END IF;
   END IF;
END;
```

OG_Property_Exists

Description This function determines if a user-defined property has been created for a particular object.
Syntax

```
FUNCTION OG_Property_Exists
  (object_hdl  OG_Object,
   prop_name   VARCHAR2)
RETURN BOOLEAN;
```

Parameters

object_hdl	Is the handle to the object containing the property.
prop_name	is the name of the property whose existence you want to check.

Returns TRUE	If the property exists.
FALSE	If the property does not exist.

OG_Property_Exists Examples

```
/* The following procedure adds the property 'priority' to an object, if it doesn't
already exist:
*/

PROCEDURE example(object OG_Object) IS
BEGIN
  IF NOT OG_Property_Exists(Object, 'priority') THEN
    OG_Set_Property(Object, 'priority', 10);
  END IF;
END;
```

OG_Rotate

Description This procedure rotates the specified object by the specified number of degrees counter-clockwise.

Syntax

```
PROCEDURE OG_Rotate
  (object_hdl   OG_Object,
   center_pt    OG_Point,
   degrees      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object_hdl	Is the handle to the object to be rotated.
center_pt	Is the point on the layout to be used as the center of rotation.
Degrees	Is the number of degrees counter-clockwise the object should be rotated.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

OG_Rotate Examples

```
/* Suppose you have a display that contains a dial.  The needle of the dial
**points at specific values along the face of the dial.  When the data changes,
**you may want to rotate the needle to point to a new value.
*/

PROCEDURE rotate_needle (degrees IN NUMBER,
  center_pt IN OG_Point) IS
   the_needle   OG_Object;
BEGIN
   the_needle:=OG_Get_Object('Needle 1');
   OG_Rotate(The_Needle, center_pt, degrees);
END;
```

OG_Same

Description This function compares the two handles to see if they are the same. For example, if you pass this function handles to two objects, it checks whether the two handles point to the same object.

Syntax

```
FUNCTION OG_Same                              object
  (handle1  OG_Object,
   handle2  OG_Object)
RETURN BOOLEAN;
FUNCTION OG_Same                              query
  (handle1  OG_Query,
   handle2  OG_Query)
RETURN BOOLEAN;
FUNCTION OG_Same                              chart template
  (handle1  OG_Template,
   handle2  OG_Template)
   RETURN BOOLEAN;
FUNCTION OG_Same                              button procedure
  (handle1  OG_Buttonproc,
   handle2  OG_Buttonproc)
RETURN BOOLEAN;
FUNCTION OG_Same                              sound
  (handle1  OG_Sound,
   handle2  OG_Sound)
RETURN BOOLEAN;
FUNCTION OG_Same                              window
  (handle1  OG_Window,
   handle2  OG_Window)
RETURN BOOLEAN;
FUNCTION OG_Same                              layer
  (handle1  OG_Layer,
   handle2  OG_Layer)
RETURN BOOLEAN;
FUNCTION OG_Same                              timer
  (handle1  OG_Timer,
   handle2  OG_Timer)
RETURN BOOLEAN;
FUNCTION OG_Same                              display
  (handle1  OG_Display,
   handle2  OG_Display)
RETURN BOOLEAN;
FUNCTION OG_Same                              axis
  (handle1  OG_Axis,
   handle2  OG_Axis)
RETURN BOOLEAN;
FUNCTION OG_Same                              field template
  (handle1  OG_Ftemp,
   handle2  OG_Ftemp)
RETURN BOOLEAN;
FUNCTION OG_Same                              reference line
  (handle1  OG_Refline,
   handle2  OG_Refline)
RETURN BOOLEAN;
```

Parameters

handle1	Is the first of two handles to compare.
handle2	Is the second of two handles to compare.

Returns TRUE	If the two handles are the same.
FALSE	If the two handles are not the same.

Usage Notes

This function is necessary because you cannot use "=" to compare the values of handles. For eample, the following procedure is *not* legal:

```
PROCEDURE invalid (obj1 OG_Object, obj2 OG_Object) IS
BEGIN
   IF obj1 = obj2 THEN     --illegal comparison
      NULL;
   END IF;
END;
```

OG_Same Examples

```
/* Suppose you want to compare two objects to see if they are the same.
**The following function returns TRUE if they are the same and FALSE if they are not:
*/

FUNCTION compare (obj1 OG_Object, obj2 OG_Object) RETURN BOOLEAN IS
BEGIN
   IF OG_Same(Obj1, obj2) THEN
      RETURN(TRUE);
   ELSE
      RETURN(FALSE);
   END IF;
END;
```

OG_Scale

Description This procedure resizes the specified object.

Syntax

```
PROCEDURE OG_Scale
  (object_hdl.  OG_Object,
   anchor       OG_Point,
   oldpt        OG_Point,
   newpt        OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object_hdl	Is the handle to the object to scale.
anchor	Is the anchor point of the object.
oldpt	Is the start point.
newpt	Is the end point.
Damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes The scaling factor is calculated by taking the distance between the new point and the anchor point, and the distance between the old point and the anchor point. The ratio of these two distances is the scaling factor. A separate scaling factor is calculated for the x- and y-coordinates.
The relative position of each of the object's control points with respect to the anchor point will then be scaled by this factor. Note that if the x- and y-scaling factors are equal, the object will be resized while maintaining its aspect ratio (e.g., a square will remain a square).
For example, to double the size of an object whose upper-left control point is at (OG_Inch, OG_Inch), you can use the following values: *anchor* = (OG_Inch, OG_Inch), *oldpt* = (OG_Inch+1, OG_Inch+1), *newpt* = (OG_Inch+2, OG_Inch+2). Thus, the x-scaling factor would be: (*newpt.x-anchor.x*) / (*oldpt.x-anchor.x*) = (OG_Inch+2-OG_Inch) / (OG_Inch+1-OG_Inch) = 2 / 1 = 2. The y-scaling factor would be: (*newpt.y-anchor.y*) / (*oldpt.y-anchor.y*) = (OG_Inch+2-OG_Inch) / (OG_Inch+1-OG_Inch) = 2 / 1 = 2.
The distance between the anchor point and each of the object's control points will then be scaled by this factor. Following the above example, if the scale factor is 2 and the upper-right control point of the object is 1.5 inches to the right of the anchor point, the control point will be moved to a position that is 3 inches to the right of the anchor point. The object's other control points will be moved in a similar manner.
Note that if a control point is used as an anchor point, its position will not change (since the distance between the control point and the anchor point would be 0).
You can also use this procedure to resize the specified object, as if you had used a Select tool to select on a control point and drag it to a new position on the layout. The anchor point is the control point that does not move during the operation (the point diagonally opposite the point you would drag on the layout).
The control point you want to move is *oldpt*, and *newpt* is the new position to move it to.

OG_Scale Examples

```
/* Suppose you want to double the size of the object that the user selects.
**Assume the object's center is at (OG_Inch, OG_Inch), and use this point as
**the anchor.  The following button procedure will double the size of the object:
*/

PROCEDURE double (buttonobj IN OG_Object, hitobj IN
  OG_Object, win IN OG_Window, eventinfo IN OG_Event) IS
   anchor OG_Point;
   newpt OG_Point;
   oldpt OG_Point;
BEGIN
   anchor.x:=OG_Inch;
   anchor.y:=OG_Inch;
   oldpt.x:=OG_Inch+1;
   oldpt.y:=OG_Inch+1;
   newpt.x:=OG_Inch+2;
   newpt.y:=OG_Inch+2;
   OG_Scale(Hitobj, anchor, oldpt, newpt);
END;
```

OG_Set_Edgecolor

Description This procedure sets the edge color of the specified object. It sets the edge pattern to 'transparent', and the background edge color to the specified color.

Syntax

```
PROCEDURE OG_Set_Edgecolor
  (object       OG_Object,
   color        VARCHAR2
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object	Is the handle to the object to change.
color	Is the name of the color.
Damage	Is the damage flag
update_bbox	Is the bounding box update flag.

OG_Set_Edgecolor Examples

```
/* The following procedure sets the edge color of the specified object:
*/

PROCEDURE example(object OG_Object) IS
BEGIN
  OG_Set_Edgecolor(Object, 'red');
END;
```

OG_Set_Fillcolor

Description This procedure sets the fill color of the specified object. It sets the fill pattern to 'transparent', and the background fill color to the specified color.

Syntax

```
PROCEDURE OG_Set_Fillcolor
  (object       OG_Object,
   color        VARCHAR2
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

object	Is the handle to the object to change.
color	Is the name of the color.
Damage	Is the damage flag
update_bbox	Is the bounding box update flag.

OG_Set_Fillcolor Examples

```
/* The following procedure sets the fill color of the specified object:
*/

PROCEDURE example(object OG_Object) IS
BEGIN
  OG_Set_Fillcolor(Object, 'red');
END;
```

OG_Set_Property

Description This procedure sets the value of an object's user-defined property.

Syntax

```
PROCEDURE OG_Set_Property                    date
  (object_hdl  OG_Object,
   prop_name   VARCHAR2,
   prop_value  VARCHAR2,
   date_fmt    VARCHAR2   := 'DD-MON-YY');
PROCEDURE OG_Set_Property                    number
  (object_hdl  OG_Object,
   prop_name   VARCHAR2,
   prop_value  NUMBER);
PROCEDURE OG_Set_Property                    char
  (object_hdl  OG_Object,:
   prop_name   VARCHAR2,
   prop_value  VARCHAR2,);
```

Parameters

object_hdl	Is the handle to the object whose property you want to set.
prop_name	Is the name of the property to set.
prop_value	Is the value to which the property will be set.
date_fmt	Is the date format mask for converting the *prop_value* character string into a date.

Usage Notes If the property exists, this procedure changes its value. If the property does not exist, this procedure creates it and sets its value without raising an error.

OG_Set_Property Examples

```
/* The following procedure gets the 'priority' property in each child object in a
**group, and then sets the priority to one greater than its current value:
*/

PROCEDURE example(group_name VARCHAR2) IS
  group_obj    OG_Object;
  child_count  NUMBER;
  child_obj    OG_Object;
  current_p    NUMBER;
BEGIN
  group_obj := OG_Get_Object(Group_Name);
  child_count := OG_Get_Childcount(Group_Obj);
  FOR i IN 0..child_count-1 LOOP
    child_obj := OG_Get_Child(Group_Obj, i);
    current_p := OG_Get_Num_Property(Child_Obj, 'priority');
    OG_Set_Property(Child_Obj, 'priority', current_p + 1);
  END LOOP;
END;
```

OG_Synchronize

Description This procedure forces the damage regions in all windows to be redrawn. It "synchronizes" the internal descriptions of all objects with their visual representations.

Syntax

```
PROCEDURE OG_Synchronize;
```

Parameters:

None.

Usage Notes Note that in your own PL/SQL programs, an implicit OG_Synchronize is executed at the end of the highest level procedure or function that is invoked.

OG_Synchronize Examples

```
/* Suppose you want to move an object across the display in ten 1/4" increments.
**Instead of moving it multiple times and having it update visually only at the end
**of the procedure, you may want to "synchronize" the layout with the internal
**representation of the object after each move.
*/

PROCEDURE slide_across(the_object IN OG_Object) IS
   offset   OG_Point;
BEGIN
   offset.x:=(1/4)*OG_Inch;
   offset.y:=0;
   FOR i IN 1..10 LOOP
      move(the_object, offset);
      OG_Synchronize;
   END LOOP;
END;
```

OG_Update_Bbox

Description This function updates an object's bounding box(es). If the object is a group, the bounding boxes of all of its descendants are also updated.

Syntax

```
PROCEDURE OG_Update_Bbox
  (object_hdl  OG_Object,
   which_bbox  NUMBER);
```

Parameters

object_hdl	Is the handle to the object to update.
which_bbox	Specifies whether the inner or outer bounding box is updated. The value of this argument may be one of the following built-in constants: **OG_Bothbbox** Means update both the inner and outer bounding boxes. **OG_Innerbbox** Means update only the inner bounding box. **OG_Outerbbox** Means update only the outer bounding box.

OG_Update_Bbox Examples

```
/*Suppose you want to move an object.  The default behavior
**of the built-in procedure OG_Move is to update the bounding
**boxes of all of the modified object's antecedants, including
**the layer on which the object resides.  To update a layer's
**bounding boxes, Graphics Builder must examine every object on that layer.
**If the layer contains a large number of objects, this operation
**can be very time-consuming.*/

 /*To make your application more efficient, you can move the
**object while inhibiting this automatic bounding box update,
**then explicitly update only that object's bounding boxes.
**(Note that since the automatic bounding box update does not
**occur, the bounding boxes of the object's antecedants
**may be inaccurate.)
*/

/*When you modify an object with a FALSE bounding box update
**flag, you may also want to use a FALSE damage flag.  In this case,
**when you are through modifying the object, you would invoke
**OG_Damage to explicitly damage the object.
*/

PROCEDURE move_efficiently (the_object OG_Object) IS
   offset   OG_Point;
BEGIN
   offset.x:=OG_Inch;
   offset.y:=OG_Inch;
   OG_Move(The_Object, offset, FALSE, FALSE)
   OG_Update_Bbox(The_Object, OG_Bothbbox);
   OG_Damage(The_Object);
END;
```

Layer Built-ins

OG_Activate_Layer
OG_Get_Layer
OG_Hide_Layer
OG_Show_Layer

OG_Activate_Layer

Description This procedure activates the specified layer in the specified window.

Syntax

```
PROCEDURE OG_Activate_Layer
  (layer_hdl   OG_Layer,
   window_hdl  OG_Window
   damage      BOOLEAN   :=  TRUE);
PROCEDURE OG_Activate_Layer
  (layer_hdl   OG_Layer,
   damage      BOOLEAN   :=  TRUE);
```

Parameters

layer_hdl	Is the handle to the layer that is activated.
window_hdl	Is the handle to the window in which the layer is activated. If not specified, the layer is

	activated in all windows.
damage	Is the damage flag.

Usage Notes Activating a hidden layer forces it to be shown. One layer only can be active at a time; when you activate a layer, the previously active layer is deactivated.
If you insert a group object as a child of the display's root object, you can then use OG_Get_Layer to get a layer handle to that group. Activating such a group object forces Graphics Builder to recognize it as a layer.

OG_Activate_Layer Examples

```
/* Suppose your layout contains several layers,
**each of which contains a set of buttons.
**If you want certain buttons to be active at
**a specific time, you can activate the layer
**that contains those buttons.  If the user
**selects a button object that is not in the active
**layer, nothing will happen.
*/

PROCEDURE activate_a_layer(layer_num NUMBER, the_window OG_Window) IS
   layer_name   VARCHAR2(6);
   the_layer    OG_Layer;
BEGIN
   layer_name:='layer'||TO_CHAR(layer_num);
   the_layer:=OG_Get_Layer(Layer_Name);
   OG_Activate_Layer(The_Layer, the_window);
END;
```

OG_Get_Layer

Description Note that you can also treat a layer as a group object by passing its name to OG_Get_Object.
Syntax

```
FUNCTION OG_Get_Layer
  (layer_name  VARCHAR2)
RETURN OG_Layer;
```

Parameters

layer_name	Is the name of the layer whose handle should be returned.

Returns A handle to the specified layer. If the layer does not exist, this function will return a null handle.
Usage Notes In addition, you can user OG_Get_Layer to get a layer handle to a group object, then force that group to become a layer by showing it or activating it.

OG_Get_Layer Examples

```
/* Suppose you want to hide "layer1".
*/

PROCEDURE make_layer1_invis (the_window) IS
   my_layer   OG_Layer;
BEGIN
   my_layer:=OG_Get_Layer('Layer1');
   OG_Hide_Layer(My_Layer, the_window);
END;
```

OG_Hide_Layer

Description This procedure hides the specified layer.

Syntax
```
PROCEDURE OG_Hide_Layer
  (layer_hdl   OG_Layer);
PROCEDURE OG_Hide_Layer
  (layer_hdl   OG_Layer,
   window_hdl  OG_Window);
```

Parameters

layer_hdl	Is the handle to the layer that is hidden.
window_hdl	Is the handle to the window in which the layer is hidden. If not specified, the layer is hidden in all windows.

Usage Notes If the layer is showing in more than one window, it will be hidden in the specified window only. The active layer cannot be hidden; to do so, you must first activate another layer.

OG_Hide_Layer Examples

```
/* Suppose "layer1" contains information that is no longer useful to view.
**The following procedure will hide it:
*/

PROCEDURE make_layer1_invis(the_window) IS
   my_layer   OG_Layer;
BEGIN
   my_layer:=OG_Get_Layer('Layer1');
   OG_Hide_Layer(My_Layer, the_window);
END;
```

OG_Show_Layer

Description This procedure shows the specified layer.

Syntax
```
PROCEDURE OG_Show_Layer
  (layer_hdl   OG_Layer,
   window_hdl  OG_Window);
```

Parameters

layer_hdl	Is the handle to the layer that is shown.
window_hdl	Is the handle to the window in which the layer is shown. If not specified, the layer is shown in all windows.

Usage Notes If the layer is hidden in more than one window, it will be shown in the specified window only.
If you insert a group object as a child of the display's root object, you can then use OG_Get_Layer to get a layer handle to that group. Showing such a group object forces Graphics Builder to recognize it as a layer.

OG_Show_Layer Examples

```
/* Suppose you want to show "layer1".
*/

PROCEDURE make_layer_visible (the_window) IS
   my_layer   OG_Layer;
BEGIN
```

```
   my_layer:=OG_Get_Layer('Layer1');
   OG_Show_Layer(My_Layer, the_window);
END;
```

Miscellaneous Built-ins

Do_Sql
OG_Append_Directory
OG_Append_File
OG_Center
OG_Damage (Region)
OG_Get_Attr (Application)
OG_Get_Attr (Axis)
OG_Get_Attr (Display)
OG_Get_Attr (Field Template)
OG_Get_Attr (Frame Template)
OG_Get_Attr (Object)
OG_Get_Attr (Printer)
OG_Get_Attr (Query)
OG_Get_Attr (Reference Line)
OG_Get_Attr (Sound)
OG_Get_Attr (Timer)
OG_Get_Attr (Window)
OG_Get_Buttonproc
OG_Help
OG_Host
OG_Pause
OG_Print
OG_Quit
OG_Root_Object
OG_Set_Attr (Application)
OG_Set_Attr (Axis)
OG_Set_Attr (Chart Element)
OG_Set_Attr (Display)
OG_Set_Attr (Field Template)
OG_Set_Attr (Frame Template)
OG_Set_Attr (Object)
OG_Set_Attr (Printer)
OG_Set_Attr (Query)
OG_Set_Attr (Reference Line)
OG_Set_Attr (Sound)
OG_Set_Attr (Timer)
OG_Set_Attr (Window)
OG_Translate_Envvar
OG_User_Exit

DO_SQL

Description This procedure executes the specified SQL statement.

Syntax

```
PROCEDURE do_sql
```

```
  (sql_stmt  VARCHAR2);
```

Parameters

sql_stmt	Is any valid SQL statement. This includes either DML (data manipulation language) or DDL (data definition language) statements.

Usage Notes Since standard PL/SQL does not allow you to execute DDL statements, use this procedure to execute them, instead. You can, however, include DML statements in your PL/SQL program units. In general, DML statements are executed more efficiently within program units than with the DO_SQL procedure.

Do_Sql Examples

```
/* The following procedure creates a table from within Graphics Builder:
*/

PROCEDURE create_table(table_name VARCHAR2) IS
BEGIN
  do_sql('create table' || table_name ||
         '  (empno  number(4),' ||
         '   ename  varchar2(10));');
END;
```

OG_Append_Directory

Description This function builds a string that specifies a pathname in your file system.

Syntax

```
FUNCTION OG_Append_Directory
  (dir     VARCHAR2,
   subdir  VARCHAR2)
RETURN VARCHAR2;
```

Parameters

dir	Is a string specifying the directory to which *subdir* is appended. This argument must contain the complete name of a valid directory.
subdir	Is a string specifying a subdirectory that is appended to *dir*.

Returns A character string containing the complete directory path.

Usage Notes You provide the names of the directory and subdirectory, and this function concatenates them using the directory separator that is appropriate for your system.

OG_Append_Directory Examples

```
/* Suppose you create a display that is run on several different systems,
**and one function of that display is to import an image from the file
**`my_image'.  Assume the identical directory structure exists on all systems
**on which the display is run; however, each system requires a different
**directory separator.  The following procedure creates a valid directory
**string for each system:
*/

PROCEDURE import_my_image(file_path VARCHAR2) IS
   the_image   OG_Object;
BEGIN
   file_path:=OG_Append_Directory(File_Path, 'home');
   file_path:=OG_Append_Directory(File_Path, 'smith');
   file_path:=OG_Append_Directory(File_Path, 'images');
   file_path:=OG_Append_File(File_Path, 'my_image');
   the_image:=OG_Import_Image(File_Path, OG_Filesystem,
OG_Tiff_Iformat);
END;
 /*Assume the intial value of file_path is `C:\'.  On MS-DOS systems,
**the value of **file_path that is passed to OG_Import_Image is:
**C:\home\smith\images\my_image.
*/
 /*Assume the initial value of file_path is `disk$ic1[]'.  On VMS systems,
**the value of file_path that is passed to OG_Import_Image is:
**disk$ic1:[home.smith.images]my_image.
*/
```

OG_Append_File

Description This function builds a string that specifies a file's pathname in your file system.

Syntax

```
FUNCTION OG_Append_File
  (dir       VARCHAR2,
   filename  VARCHAR2)
RETURN VARCHAR2;
```

Parameters

dir	Is a string specifying the directory to which *filename* is appended. This argument must contain the complete name of a valid directory.
filename	Is a string specifying a filename that is appended to *dir*.

Returns A character string containing the complete file path.

Usage Notes You provide the names of the directory and file, and this function concatenates them using the directory separator that is appropriate for your system.

OG_Append_File Examples

```
/* Suppose you create a display that is run on several different systems,
**and one function of that display is to import an image from the file `my_image'.
**Assume the identical directory structure exists on all systems on which the display
**is run; however, each system requires a different directory separator.
**The following procedure creates a valid directory string for each system:
*/

PROCEDURE import_my_image(file_path VARCHAR2) IS
   the_image   OG_Object;
BEGIN
   file_path:=OG_Append_Directory(File_Path, 'home');
   file_path:=OG_Append_Directory(File_Path, 'smith');
   file_path:=OG_Append_Directory(File_Path, 'images');
   file_path:=OG_Append_File(File_Path, 'my_image');
   the_image:=OG_Import_Image(File_Path, OG_Filesystem,
     OG_Tiff_Iformat);
END;
```

/*Assume the intial value of *file_path* is `C:\'. On MS-DOS systems, the value of
***file_path* that is passed to OG_Import_Image is: C:\home\smith\images\my_image.
*/
/*Assume the initial value of *file_path* is `disk$ic1[]'. On VMS systems, the value of
***file_path* that is passed to OG_Import_Image is: disk$ic1:[home.smith.images]my_image.
*/

OG_Center

Description This procedure redraws the display in the specified window such that the point in the display represented by *center_pt* appears at the center of the window.

Syntax

```
PROCEDURE OG_Center
  (window_hdl  OG_Window,
   center_pt   OG_Point);
```

Parameters

window_hdl	Is the handle to the window.
center_pt	Is the point in the display around which the window should be centered.

OG_Center Examples

```
/* Suppose you have a chart that you want to appear in the center of a window.
**To do this, you need to get the location and dimensions of the chart's outer
**bounding box, calculate its center point, then use center_pt to place this
**point in the center of the window.
*/

PROCEDURE center_chart (my_window IN og_window, my_chart IN
og_object) IS
   center_point   og_point;
   chart_record   og_chart_ca;
BEGIN
   chart_record.chart_caob.mask:=OG_OBBOX_GENERICA;
   chart_record.chart_caog_mask:=OG_NONE_GROUPA;
   chart_record.chart_caoc_mask:=OG_NONE_CHARTA;
   og_get_attr (my_chart, chart_record);
   center_pt.x:=chart_record.chart_caob.obbox.x +
(chart_record.chart_caob.obbox.width / 2);
   center_pt.y:=chart_record.chart_caob.obbox.y +
(chart_record.chart_caob.obbox.height / 2);
   og_center (my_window, center_point);
END;
```

OG_Damage (Region)

Description This procedure damages a rectangular region on the layout.

Syntax

```
PROCEDURE OG_Damage
 (region  OG_Rectangle);

PROCEDURE OG_Damage
 (region     OG_Rectangle,
  layer_hdl  OG_Layer);
```

Parameters

region	Is a rectangular region to damaged.
layer_hdl	Is the layer in which the rectangular region is damaged. If *layer_hdl* is not specified, the region on all layers are damaged.

Usage Notes For more information, see "Damage Flag" .

OG_Damage (Region) Examples

```
/* The following procedure damages a 3"x2"
** area in the upper-left corner of the layout:
*/

PROCEDURE example IS
 damage_region  OG_Rectangle;
BEGIN
 damage_region.x := 0;
 damage_region.y := 0;
 damage_region.width := 3 * OG_Inch;
 damage_region.height := 2 * OG_Inch;
```

```
  OG_Damage(Damage_Region);
END;
```

OG_Get_Attr (Application)

Description

Syntax

```
PROCEDURE OG_Get_Attr
 (attr  IN OUT  OG_App_Attr);
```

Parameters

attr	Is the attribute record to be filled with the attributes of the application.

Usage Notes This procedure gets the attribute values of the currently running Graphics Builder executable.

The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Application) Examples

```
/* The following function returns the number of screen resolution units
**(i.e., pixels) for the current display device:
*/

FUNCTION example RETURN NUMBER IS
  rec  OG_App_Attr;
BEGIN
  rec.mask := OG_Screen_Res_Appa;
  OG_Get_Attr(Rec);
  RETURN(rec.hscreen_res);
END;
```

OG_Get_Attr (Axis)

Description This procedure gets the attribute values of the specified axis.

Syntax

```
PROCEDURE OG_Get_Attr                             generic
  (axis_hdl  IN      OG_Axis,
   attr      IN OUT  OG_Axis_Attr);
PROCEDURE OG_Get_Attr                             continuous
  (axis_hdl  IN      OG_Axis,
   attr      IN OUT  OG_Contaxis_Ca);
PROCEDURE OG_Get_Attr                             date
  (axis_hdl  IN      OG_Axis,
   attr      IN OUT  OG_Dateaxis_Ca);
PROCEDURE OG_Get_Attr                             discrete
  (axis_hdl  IN      OG_Axis,
   attr      IN OUT  OG_Discaxis_Ca);
```

Parameters

axis_hdl	Is the handle to the axis whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the axis.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Axis) Examples

```
/* The following function returns the custom label for the specified axis:
*/

FUNCTION example(axis OG_Axis) RETURN CHAR IS
  rec  OG_Contaxis_Ca;
BEGIN
  rec.ca_axis.mask := OG_Custlabel_Axisa;
  rec.ca_cont.mask := OG_None_Contaxisa;
  OG_Get_Attr(Axis, rec);
  RETURN(rec.ca_axis.custlabel);
END;
```

OG_Get_Attr (Display)

Description This procedure gets the attribute values of the current display.

Syntax

```
PROCEDURE OG_Get_Attr
  (attr  IN OUT  OG_Display_Attr);
```

Parameters

attr	Is the attribute record to be filled with the attributes of the display.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Display) Examples

```
/* The following function returns the display width:
*/

FUNCTION example RETURN NUMBER IS
  rec  OG_Display_Attr;
BEGIN
  rec.mask := OG_Size_Displaya;
  OG_Get_Attr(Rec);
  RETURN(rec.width);
END;
```

OG_Get_Attr (Field Template)

Description This procedure gets the attribute values of the specified field template.

Syntax

```
PROCEDURE OG_Get_Attr                        generic
  (ftemp_hdl  IN      OG_Ftemp,
   attr       IN OUT  OG_Ftemp_Attr);
PROCEDURE OG_Get_Attr                        axis
  (ftemp_hdl  IN      OG_Ftemp,
   attr       IN OUT  OG_Axisftemp_Ca);
```

Parameters

ftemp_hdl	Is the handle to the field template whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the field template.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Field Template) Examples

```
/* The following function returns the number format mask for the specified field
template:
*/

FUNCTION example(ftemp OG_Ftemp) RETURN CHAR IS
  rec  OG_Axisftemp_Ca;
BEGIN
  rec.ca_ftemp.mask := OG_Numfmt_Ftempa;
  rec.ca_aftemp.mask := OG_None_Axisftempa;
  OG_Get_Attr(Ftemp, rec);
  RETURN(rec.ca_ftemp.numfmt);
END;
```

OG_Get_Attr (Frame Template)

Description This procedure gets the attribute values of the specified frame template.

Syntax

```
PROCEDURE OG_Get_Attr                          generic frame
  (template_hdl  IN      OG_Template,
   attr          IN OUT  OG_Frame_Attr);
PROCEDURE OG_Get_Attr                          axis frame
  (template_hdl  IN      OG_Template,
   attr          IN OUT  OG_Axisframe_Ca);
PROCEDURE OG_Get_Attr                          pie frame
  (template_hdl  IN      OG_Template,
   attr          IN OUT  OG_Pieframe_Ca);
PROCEDURE OG_Get_Attr                          table frame
  (template_hdl  IN      OG_Template,
   attr          IN OUT  OG_Tableframe_Ca);
```

Parameters

template_hdl	Is the handle to the chart template whose frame attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the frame template.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Frame Template) Examples

```
/* The following function returns the depth size for the specified template frame:
*/

FUNCTION example(temp OG_Template) RETURN NUMBER IS
  rec  OG_Axisframe_Ca;
BEGIN
```

```
   rec.ca_frame.mask := OG_Depthsize_Framea;
   rec.ca_axis.mask := OG_None_Framea;
   OG_Get_Attr(Temp, rec);
   RETURN(rec.ca_frame.depthsize);
 END;
```

OG_Get_Attr (Object)

Description This procedure gets the attribute values of the specified object.

Syntax

```
PROCEDURE OG_Get_Attr                          arc
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Arc_Ca);
PROCEDURE OG_Get_Attr                          chart
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Chart_Ca);
PROCEDURE OG_Get_Attr                          generic
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Generic_Attr);
PROCEDURE OG_Get_Attr                          graphic
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Graphic_Ca);
PROCEDURE OG_Get_Attr                          group
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Group_Ca);
PROCEDURE OG_Get_Attr                          image
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Image_Ca);
PROCEDURE OG_Get_Attr                          line
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Line_Ca);
PROCEDURE OG_Get_Attr                          polygon/polyline
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Poly_Ca);
PROCEDURE OG_Get_Attr                          rectangle
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Rect_Ca);
PROCEDURE OG_Get_Attr                          rounded rectangle
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Rrect_Ca);
PROCEDURE OG_Get_Attr                          symbol
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Symbol_Ca);
PROCEDURE OG_Get_Attr                          text/text field
  (object_hdl  IN      OG_Object,
   attr        IN OUT  OG_Text_Ca);
```

Parameters

object_hdl	Is the handle to the object whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the object.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Object) Examples

```
/* Suppose the user selects an object whose color determines what information
**the user is interested in viewing.  The following function will take an object
**as its **argument and return the name of its foreground fill color.  The color
```

```
**can then be determined, and the appropriate information displayed.
*/

FUNCTION get_color(the_object IN OG_Object) RETURN VARCHAR2
IS
   obj_record   OG_Graphic_Ca;
BEGIN
   obj_record.graphic_caoh.mask:=OG_Ffcolor_Graphica;
   obj_record.generic_caob.mask:=OG_None_Generica;
   OG_Get_Attr(The_Object, obj_record);
   RETURN(obj_record.graphic_caoh.ffcolor);
END;
```

OG_Get_Attr (Printer)

Description This procedure gets the attribute values of the current printer..

Syntax

```
PROCEDURE OG_Get_Attr
  (attr  IN OUT  OG_Printer_Attr);
```

Parameters

attr	Is the attribute record to be filled with the attributes of the printer.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Printer) Examples

```
/* The following function returns the name of the current printer:
*/

FUNCTION example RETURN CHAR IS
  rec  OG_Printer_Attr;
BEGIN
  rec.mask := OG_Name_Printera;
  OG_Get_Attr(Rec);
  RETURN(rec.name);
END;
```

OG_Get_Attr (Query)

Description This procedure gets the attribute values of the specified query.

Syntax

```
PROCEDURE OG_Get_Attr
  (query_hdl  IN      OG_Query,
   attr       IN OUT  OG_Query_Attr);
```

Parameters

query_hdl	Is the handle to the query whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the query.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Query) Examples

```
/* The following function returns the SQL statement that defines the specified query:
*/

FUNCTION example(query OG_Query) RETURN CHAR IS
  rec  OG_Query_Attr;
BEGIN
  rec.mask := OG_Querysource_Querya;
  OG_Get_Attr(Query, rec);
  RETURN(rec.querysource);
END;
```

OG_Get_Attr (Reference Line)

Description This procedure gets the attribute values of the specified reference line.

Syntax

```
PROCEDURE OG_Get_Attr
  (refline_hdl  IN      OG_Refline,
   attr         IN OUT  OG_Refline_Attr);
```

Parameters

refline_hdl	Is the handle to the reference line whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the reference line.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Reference Line) Examples

```
/* The following function returns the label for the specified reference line.
*/

FUNCTION example(refline OG_Refline) RETURN CHAR IS
  rec  OG_Refline_Attr;
BEGIN
  rec.mask := OG_Label_Reflinea;
  OG_Get_Attr(Refline, rec);
  RETURN(rec.label);
END;
```

OG_Get_Attr (Sound)

Description This procedure gets the attribute values of the specified sound.

Syntax

```
PROCEDURE OG_Get_Attr
  (sound_hdl  IN      OG_Sound,
   attr       IN OUT  OG_Sound_Attr);
```

Parameters

sound_hdl	Is the handle to the sound whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the sound.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Sound) Examples

```
/* The following function returns the name of the specified sound:
*/

FUNCTION example(sound OG_Sound) RETURN CHAR IS
  rec  OG_Sound_Attr;
BEGIN
  rec.mask := OG_Name_Sounda;
  OG_Get_Attr(Sound, rec);
  RETURN(rec.name);
END;
```

OG_Get_Attr (Timer)

Description This procedure gets the attribute values of the specified timer.

Syntax

```
PROCEDURE OG_Get_Attr
  (timer_hdl  IN      OG_Timer,
   attr       IN OUT  OG_Timer_Attr);
```

Parameters

timer_hdl	Is the handle to the timer whose attributes you want to get.
attr	Is the attribute record to be filled with the attributes of the timer.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Timer) Examples

```
/* The following function returns the procedure name assigned to the specified timer:
*/

FUNCTION example(timer OG_Timer) RETURN CHAR IS
  rec  OG_Timer_Attr;
BEGIN
  rec.mask := OG_Timerproc_Timera;
  OG_Get_Attr(Timer, rec);
  RETURN(rec.timerproc);
END;
```

OG_Get_Attr (Window)

Description This procedure gets the attribute values of the specified window.

Syntax

```
PROCEDURE OG_Get_Attr
  (window_hdl  IN      OG_Window,
   attr        IN OUT  OG_Window_Attr);
```

Parameters

window_hdl	Is the handle to the window whose attributes

	you want to get.
attr	Is the attribute record to be filled with the attributes of the window.

Usage Notes The only attribute values that will be retrieved are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Get_Attr (Window) Examples

```
/* The following function returns the specified window's position:
*/

FUNCTION example(window OG_Window) RETURN OG_Point IS
  rec  OG_Window_Attr;
BEGIN
  rec.mask := OG_Position_Windowa;
  OG_Get_Attr(Window, rec);
  RETURN(rec.position);
END;
```

OG_Get_Buttonproc

Description The button procedure must already be defined, and not exist in any PL/SQL package.
Syntax

```
FUNCTION OG_Get_Buttonproc
  (proc_name  VARCHAR2)
RETURN OG_Buttonproc;
```

Parameters

proc_name	Is the name of the PL/SQL button procedure whose handle should be returned.

Returns A handle to the specified button procedure. If the specified button procedure does not exist, this function will return a null handle.

OG_Get_Buttonproc Examples

```
/* Suppose you have written a button procedure named `show_sales_data'
**and you want to assign that procedure to an object; then, when the user
**selects the object, the procedure will be executed.
*/

PROCEDURE make_object_button(my_obj IN OG_Object) IS
   obj_rec         OG_Generic_Attr;
   my_buttonproc   OG_Buttonproc;
BEGIN
   my_buttonproc:=OG_Get_Buttonproc('Show_Sales_Data');
   obj_rec.mask:=OG_Button_Generica;
   obj_rec.button:=my_buttonproc;
   obj_rec.events:=OG_Mouse_Down;
   OG_Set_Attr(My_Obj, obj_rec);
END;
```

OG_Help

Description This procedure invokes the Help system and shows the runtime Help document at the specified hypertext target.

Syntax

```
PROCEDURE OG_Help
  (target  VARCHAR2);
```

Parameters

target	Is the hypertext target in the runtime Help document that is displayed.

OG_Help Examples

```
/* Suppose you want the user to be able to select a button and invoke
**the Help system.  You could write the following button procedure:
*/

PROCEDURE get_help (buttonobj IN OG_Object, hitobj IN
OG_Object, win IN OG_Window, eventinfo IN OG_Event) IS
BEGIN
   IF eventinfo.event_type=OG_Mouse_Up THEN
      OG_Help('Topic_1');
   END IF;
END;
```

OG_Host

Description This procedure passes the specified command to the operating system.

Syntax

```
PROCEDURE OG_Host
  (command  VARCHAR2);
```

Parameters

command	Is a text string containing the command to execute.

OG_Host Examples

```
/* Suppose you want to be notified via electronic mail when a user closes a display.
**You could create a script named `mail_me' in your file system that sends you mail,
**and then invoke it with the following Close Display trigger:
*/

PROCEDURE send_me_mail IS
BEGIN
   OG_Host('Mail_Me');
END;
```

OG_Pause

Description This procedure suspends the execution of the display for the specified number of seconds.

Syntax

```
PROCEDURE OG_Pause
  (secs  NUMBER);
```

Parameters

secs	Is the number of seconds to pause.

OG_Pause Examples

```
/* The following procedure suspends display execution for seven seconds:
*/

PROCEDURE example IS
BEGIN
  OG_Pause(7);
END;
```

OG_Print

Description This procedure prints the contents of the layout to the currently selected print device.

Syntax

```
PROCEDURE OG_Print;
PROCEDURE OG_Print
  (window_hdl  OG_Window);
```

Parameters

window_hdl	Is the handle to the window to be printed.

Usage Notes If a window handle is specified, only the layers showing in that window are printed; otherwise, all layers in the display are printed, regardless of what window they are in or whether they are showing or hidden.

OG_Print Examples

```
/* Suppose you want to print the contents of the main layout window.
*/

PROCEDURE print_main_window IS
   the_window   OG_Window;
BEGIN
   the_window:=OG_Get_Window('Main Layout');
   OG_Print(The_Window);
END;
```

OG_Quit

Description This procedure quits the current Graphics Builder session.

Syntax

```
PROCEDURE OG_Quit;
```

Parameters:
None.

OG_Quit Examples

```
/* Suppose you want to provide the user with a button that-when selected
**commits database changes and quits Graphics Builder.  You could write the
**following button procedure:
*/

PROCEDURE commit_and_quit (hitobj IN OG_Object, buttonobj
  IN OG_Object, win IN OG_Window, eventinfo IN OG_Event) IS
BEGIN
```

```
   COMMIT;
   OG_Quit;
END;
```

OG_Root_Object

Description This function returns a handle to the display's root object.

Syntax

```
FUNCTION OG_Root_Object
RETURN OG_Object;
```

Parameters:

None.

Returns A handle to the display's root object.

Usage Notes The root object is the topmost object in the display. Its immediate children are the display's layers.

OG_Root_Object Examples

```
/* The following procedure moves the topmost layer in the display to the bottom of the
layer list:
*/

PROCEDURE example IS
  root   OG_Object;
  layer  OG_Object;
BEGIN
  root := OG_Root_Object;
  layer := OG_Get_Child(Root, 0);
  OG_Insert_Child(Root, layer, OG_Last);
END;
```

OG_Set_Attr (Application)

Description This procedure sets the attributes of the currently running Graphics Builder executable.

Syntax

```
PROCEDURE OG_Set_Attr
  (attr  OG_App_Attr);
```

Parameters

attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Application) Examples

```
/* The following procedure sets the cursor to 'busy':
*/

PROCEDURE example IS
  attr  OG_App_Attr;
BEGIN
  attr.cursor := 'busy';
  attr.mask := OG_Cursor_Appa;
  OG_Set_Attr(Attr);
END;
```

OG_Set_Attr (Axis)

Description This procedure sets the attribute values of the specified axis.

Syntax

```
PROCEDURE OG_Set_Attr                          generic
  (axis_hdl  OG_Axis,
   attr      OG_Axis_Attr);
PROCEDURE OG_Set_Attr                          continuous
  (axis_hdl  OG_Axis,
   attr      OG_Contaxis_Ca);
PROCEDURE OG_Set_Attr                          date
  (axis_hdl  OG_Axis,
   attr      OG_Dateaxis_Ca);
PROCEDURE OG_Set_Attr                          discrete
  (axis_hdl  OG_Axis,
   attr      OG_Discaxis_Ca);
```

Parameters

axis_hdl	Is the handle to the axis whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Axis) Examples

```
/* The following procedure sets the custom label for the specified axis:
*/

PROCEDURE example(axis OG_Axis, label VARCHAR2) IS
  rec  OG_Contaxis_Ca;
BEGIN
  rec.ca_axis.custlabel := 'New Label';
  rec.ca_axis.mask := OG_Custlabel_Axisa;
  rec.ca_cont.mask := OG_None_Contaxisa;
  OG_Set_Attr(Axis, rec);
END;
```

OG_Set_Attr (Chart Element)

Description This procedure sets the attributes of a chart element, such as a bar or pie slice.

Syntax

```
PROCEDURE OG_Set_Attr
  (chart_hdl  OG_Object,
   row_num    NUMBER,
   col_name   VARCHAR2,
   attr       OG_Chelement_Ca);
```

Parameters

chart_hdl	Is the handle to the chart containing the data value whose attributes you want to set.
row_num	Is the row number of the data value whose attributes you want to set.
col_name	Is the column name of the data value whose attributes you want to set.
attr	Is the attribute record containing the new

	attribute values.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes You must specify the chart, row, and column of the corresponding data value, as well as a chart set attribute record. The attribute record contains graphical and other attributes that will be applied to the chart element that represents the specified data value. For example, you can set the color of a bar in a bar chart by specifying an attribute record for the data value that corresponds to that bar.
Note that any changes made to a chart element will not be applied until the chart is updated via a call to OG_Update_Chart.
The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Chart Element) Examples

```
/* The following procedure loops through all rows in a chart's query,
**and then sets the color of each bar in the chart based on its value:
*/

PROCEDURE OGTRIGGERPROC0 IS
chart og_object;
query og_query;
rec og_chelement_ca;
total number;
bar_val number;
BEGIN
  chart := og_get_object ('Employees');
  query:= og_get_query ('query0');
  og_execute_query (query);
  og_start_from(query, OG_NEWDATA);
  total := og_numrows (query, OG_NEWDATA);
for i in 0..total-1 loop
  bar_val:=og_get_numcell (query, OG_NEWDATA,'SAL');
   IF bar_val>2000 THEN^M
   rec.chelement_cagr.mask :=OG_BFCOLOR_GRAPHICA;
   rec.chelement_cace.mask :=OG_NONE_CHELEMENTA;
   rec.chelement_cagr.bfcolor := 'cyan';
og_set_attr (chart, i, 'SAL', rec);
   END; IF;
  og_next_row(query, OG_NEWDATA);
 END LOOP;
 og_update_chart (chart, OG_ALL_CHUPDA);
END;
```

OG_Set_Attr (Display)

Description This procedure sets the attributes of the current display.

Syntax

```
PROCEDURE OG_Set_Attr
  (attr  OG_Display_Attr);
```

Parameters

attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Display) Examples

```
/* The following procedure sets the display width:
*/

PROCEDURE example IS
  rec  OG_Display_Attr;
BEGIN
  rec.width := 4 * OG_Inch;
  rec.height := 5 * OG_Inch;
  rec.mask := OG_Size_Displaya;
  OG_Set_Attr(Rec);
END;
```

OG_Set_Attr (Field Template)

Description This procedure sets the attribute values of the specified field template.

Syntax

```
PROCEDURE OG_Set_Attr                            generic
  (ftemp_hdl  OG_Ftemp,
   attr       OG_Ftemp_Attr);
PROCEDURE OG_Set_Attr                            axis
  (ftemp_hdl  OG_Ftemp,
   attr       OG_Axisftemp_Ca);
```

Parameters

ftemp_hdl	Is the handle to the field template whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Field Template) Examples

```
/* The following procedure sets the number format mask for the specified field
template:
*/

PROCEDURE example(ftemp OG_Ftemp) IS
  rec  OG_Axisftemp_Ca;
BEGIN
  rec.ca_ftemp.numfmt := '9,990';
  rec.ca_ftemp.mask := OG_Numfmt_Ftempa;
  rec.ca_aftemp.mask := OG_None_Axisftempa;
  OG_Set_Attr(Ftemp, rec);
END;
```

OG_Set_Attr (Frame Template)

Description This procedure sets the attribute values of the specified frame template.

Syntax

```
PROCEDURE OG_Set_Attr                            generic frame
  (template_hdl  OG_Template,
   attr          OG_Frame_Attr);
PROCEDURE OG_Set_Attr                            axis frame
  (template_hdl  OG_Template,
   attr          OG_Axisframe_Ca);
PROCEDURE OG_Set_Attr                        pie frame
  (template_hdl  OG_Template,
   attr          OG_Pieframe_Ca);
PROCEDURE OG_Set_Attr                        table frame
  (template_hdl  OG_Template,
   attr          OG_Tableframe_Ca);
```

Parameters

template_hdl	Is the handle to the chart template whose frame attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Frame Template) Examples

```
/* The following procedure sets the depth size for the specified template frame:
*/

PROCEDURE example(temp OG_Template) IS
  rec  OG_Axisframe_Ca;
BEGIN
  rec.ca_frame.depthsize := OG_Large_Depthsize;
  rec.ca_frame.mask := OG_Depthsize_Framea;
  rec.ca_axis.mask := OG_None_Framea;
  OG_Set_Attr(Temp, rec);
END;
```

OG_Set_Attr (Object)

Description This procedure sets the attributes of the specified object.

Syntax

```
PROCEDURE OG_Set_Attr                        arc
  (object_hdl   OG_Object,
   attr         OG_Arc_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        chart
  (object_hdl   OG_Object,
   attr         OG_Chart_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        generic
  (object_hdl   OG_Object,
   attr         OG_Generic_Attr,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        graphic
  (object_hdl   OG_Object,
   attr         OG_Graphic_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        group
  (object_hdl   OG_Object,
   attr         OG_Group_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        image
  (object_hdl   OG_Object,
   attr         OG_Image_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        line
  (object_hdl   OG_Object,
   attr         OG_Line_Ca,
   damage       BOOLEAN          := TRUE,
   update_bbox  BOOLEAN          := TRUE);
PROCEDURE OG_Set_Attr                        polygon/polyline
  (object_hdl   OG_Object,
   attr         OG_Poly_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        rectangle
  (object_hdl   OG_Object,
   attr         OG_Rect_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        rounded rectangle
  (object_hdl   OG_Object,
   attr         OG_Rrect_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        symbol
  (object_hdl   OG_Object,
   attr         OG_Symbol_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
PROCEDURE OG_Set_Attr                        text/text field
  (object_hdl   OG_Object,
   attr         OG_Text_Ca,
   damage       BOOLEAN          :=  TRUE,
   update_bbox  BOOLEAN          :=  TRUE);
```

Parameters

object_hdl	Is the handle to the object whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Object) Examples

```
/* Suppose you have a map of the world and you want to change the
**color of one of the countries.  First, you would get the handle to the
**country object, then you would change its color.
*/

PROCEDURE color_country (country_name) IS
   my_object    OG_Object;
   obj_record   OG_Graphic_Ca;
BEGIN
   my_object:=OG_Get_Object(Country_Name);
   obj_record.graphic_caob.mask:=OG_None_Generica;
   obj_record.graphic_caoh.mask:=OG_Ffcolor_Graphica;
   obj_record.graphic_caoh.ffcolor:='red';
   OG_Set_Attr(My_Object, obj_record);
END;
```

OG_Set_Attr (Printer)

Description This procedure sets the attribute values of the current printer..

Syntax

```
PROCEDURE OG_Set_Attr
  (attr  OG_Printer_Attr);
```

Parameters

attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Printer) Examples

```
/* The following procedure tells sets the number of copies to print:
*/

PROCEDURE example IS
  rec  OG_Printer_Attr;
BEGIN
  rec.copies := 2;
  rec.mask := OG_Copies_Printera;
  OG_Set_Attr(Rec);
END;
```

OG_Set_Attr (Query)

Description This procedure sets the attributes of the specified query.

Syntax

```
PROCEDURE OG_Set_Attr
  (query_hdl  OG_Query,
   attr       OG_Query_Attr);
```

Parameters

query_hdl	Is the handle to the query whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Query) Examples

```
/* The following procedure sets the SQL statement that defines the specified query:
*/

PROCEDURE example(query OG_Query) IS
  rec  OG_Query_Attr;
BEGIN
  rec.querysource := 'select ename, sal from emp';
  rec.mask := OG_Querysource_Querya;
  OG_Set_Attr(Query, rec);
END;
```

OG_Set_Attr (Reference Line)

Description This procedure sets the attributes of the specified reference line.

Syntax

```
PROCEDURE OG_Set_Attr
  (refline_hdl  OG_Refline,
   attr         OG_Refline_Attr);
```

Parameters

refline_hdl	Is the handle to the reference line whose attributes you want to Set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Reference Line) Examples

```
/* The following procedure sets the label for the specified reference line.
*/

PROCEDURE example(refline OG_Refline) IS
  rec  OG_Refline_Attr;
BEGIN
  rec.label := 'Average';
  rec.mask := OG_Label_Reflinea;
  OG_Set_Attr(Refline, rec);
END;
```

OG_Set_Attr (Sound)

Description This procedure sets the attributes of the specified sound.

Syntax

```
PROCEDURE OG_Set_Attr
  (sound_hdl  OG_Sound,
   attr       OG_Sound_Attr);
```

Parameters

sound_hdl	Is the handle to the sound whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Sound) Examples

```
/* The following procedure sets the name of the specified sound:
*/

PROCEDURE example(sound OG_Sound) IS
  rec  OG_Sound_Attr;
BEGIN
  rec.name := 'Alert';
  rec.mask := OG_Name_Sounda;
  OG_Set_Attr(Sound, rec);
END;
```

OG_Set_Attr (Timer)

Description This procedure sets the attributes of the specified timer.

Syntax

```
PROCEDURE OG_Set_Attr
  (timer_hdl  OG_Timer,
   attr       OG_Timer_Attr);
```

Parameters

timer_hdl	Is the handle to the timer whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Timer) Examples

```
/* The following procedure sets the procedure name assigned to the specified timer:
*/

PROCEDURE example(timer OG_Timer) IS
  rec  OG_Timer_Attr;
BEGIN
  rec.timerproc := 'update_proc';
  rec.mask := OG_Timerproc_Timera;
  OG_Set_Attr(Timer, rec);
END;
```

OG_Set_Attr (Window)

Description This procedure sets the attributes of the specified window.

Syntax

```
PROCEDURE OG_Set_Attr
  (window_hdl  OG_Window,
   attr        OG_Window_Attr);
```

Parameters

window_hdl	Is the handle to the window whose attributes you want to set.
attr	Is the attribute record containing the new attribute values.

Usage Notes The only attribute values that will be set are those specified by the value of the *mask* attribute(s) in the attribute record. Fields in the attribute record for which the mask is not set will be unaffected by the call to this procedure.

OG_Set_Attr (Window) Examples

```
/* The following procedure sets the specified window's size:
*/
```

```
PROCEDURE example(window OG_Window) IS
  rec  OG_Window_Attr;
BEGIN
  rec.width := 4 * OG_Get_Ap_Hscreen_Res;
  rec.height := 5 * OG_Get_Ap_Vscreen_Res;
  rec.mask := OG_Size_Windowa;
  OG_Set_Attr(Window, rec);
END;
```

OG_Translate_Envvar

Description This function returns the value of the specified environment variable.

Syntax

```
FUNCTION OG_Translate_Envvar
  (envvar  VARCHAR2)
RETURN VARCHAR2;
```

Parameters

envvar	Is the environment variable to translate.

Returns A character string containing the value of the environment variable.

Usage Notes The treatment (and even the existence) of environment variables is system-specific. For more information, see the Graphics Builder documentation for your operating system.

When searching for the environment variable, Graphics Builder first checks your operating system to see if it defined. If not, it looks in your preferences file.

OG_Translate_Envvar Examples

```
/* Suppose your system has an environment variable named IMAGE_DIR
**that specifies the directory path of the image file `my_image'.  The following
**procedure imports that image:
*/

PROCEDURE import_my_image IS
   the_image   OG_Object;
   file_path   VARCHAR2(50);
BEGIN
   file_path:=OG_Translate_Envvar('Image_Dir');
   file_path:=OG_Append_File(File_Path, 'my_image');
   the_image:=OG_Import_Image(File_Path, OG_Filesystem,
     OG_Tiff_Iformat);
END;
```

OG_User_Exit

Description This procedure runs a user-defined executable.

Syntax

```
PROCEDURE OG_User_Exit
  (command  VARCHAR2);
```

Parameters

command	Is the name of the user exit, along with any arguments you wish to pass it.

OG_User_Exit Examples

```
/* Suppose your display controls the operation of hardware
**components connected to your system.  When the user selects
**a button, you may want to invoke the hardware controller
**routine, which you have linked in as a user exit.  In addition,
**you may want to pass an argument to this user exit.  The following
**procedure invokes the user exit `hw_ctrl' with the parameter `signal':
*/

PROCEDURE control_hw(buttonobj IN OG_Object, hitobj IN OG_Object, win IN OG_Window,
eventinfo IN OG_Event) IS
BEGIN
   OG_User_Exit('Hw_Ctrl' || :signal);
END;
```

Parameter Built-ins

OG_Delete_Param
OG_Get_Char_Param
OG_Get_Date_Param
OG_Get_Num_Param
OG_Get_Param_Type
OG_Param_Exists
OG_Set_Param

OG_Delete_Param

Description This procedure deletes the specified parameter.

Syntax

```
PROCEDURE OG_Delete_Param
  (param_name  VARCHAR2);
```

Parameters

param_name	Is the name of the parameter to delete.

OG_Delete_Param Examples

```
/* The following procedure deletes the parameter 'param0':
*/

PROCEDURE example IS
BEGIN
  OG_Delete_Parameter('Param0');
END;
```

OG_Get_Char_Param

Description This function gets the value of the specified CHAR parameter. It is equivalent to using a bind reference to the parameter.

Syntax

```
FUNCTION OG_Get_Char_Param
  (param_name  VARCHAR2)
RETURN VARCHAR2;
```

Parameters

param_name	Is the name of the parameter whose value you want to get.

Returns The value of the specified parameter.

Usage Notes This function is useful when you want to reference the parameter within a program unit that is in a library. Bind references do not compile within the context of a library; however, this function does.

OG_Get_Char_Param Examples

```
/* The following procedure gets the value of the parameter 'status',
**and changes the color of the specified object based on its value:
*/

PROCEDURE example(object OG_Object) IS
  stat  VARCHAR2(10);
BEGIN
  stat := OG_Get_Char_Param('Status');
  IF stat = 'obsolete' THEN
    OG_Set_Fillcolor(Object, 'red');
  END IF;
END;
```

OG_Get_Date_Param

Description This function gets the value of the specified DATE parameter. It is equivalent to using using a bind reference to the parameter.

Syntax

```
FUNCTION OG_Get_Date_Param
  (param_name  VARCHAR2,
   fmt         VARCHAR2)
RETURN DATE;
```

Parameters

param_name	Is the name of the parameter whose value you want to get.

Returns The value of the specified parameter.
Usage Notes This function is useful when you want to reference the parameter within a program unit that is in a library. Bind references do not compile within the context of a library; however, this function does.

OG_Get_Date_Param Examples

```
/* The following procedure gets the value of the parameter 'due_date',
**and changes the color of the specified object based on its value:
*/

PROCEDURE example(object OG_Object) IS
  due  DATE;
BEGIN
  due := OG_Get_Date_Param('Due_Date');
  IF due < sysdate THEN
    OG_Set_Fillcolor(Object, 'red');
  END IF;
END;
```

OG_Get_Num_Param

Description This function gets the value of the specified NUMBER parameter. It is equivalent to using using a bind reference to the parameter.
Syntax
```
FUNCTION OG_Get_Num_Param
  (param_name  VARCHAR2)
RETURN NUMBER;
```
Parameters

param_name	Is the name of the parameter whose value you want to get.

Returns The value of the specified parameter.
Usage Notes This function is useful when you want to reference the parameter within a program unit that is in a library. Bind references do not compile within the context of a library; however, this function does.

OG_Get_Num_Param Examples

```
/* The following procedure gets the value of the parameter 'priority', and increases it
by 1:
*/

PROCEDURE example IS
  val  NUMBER;
BEGIN
  val := OG_Get_Num_Param('Priority');
  OG_Set_Param('Priority', val + 1);
END;
```

OG_Get_Param_Type

Description This function returns the datatype of a parameter.
Syntax
```
FUNCTION OG_Get_Param_Type
  (param_name  VARCHAR2)
RETURN NUMBER;
```

Parameters

param_name	Is the name of the parameter.

Returns This function returns one of the following built-in constants:

- OG_Char_Paramtype
- OG_Date_Paramtype
- OG_Num_Paramtype

OG_Get_Param_Type Examples

```
/* The following procedure retrieves the datatype of the parameter 'param0',
**then increases it by one if the type is NUMBER:
*/

PROCEDURE example IS
  dtype  NUMBER;
BEGIN
  dtype := OG_Get_Param_Type('Param0');
  IF dtype = OG_Num_Paramtype THEN
    :param0 := :param0 + 1;
  END IF;
END;
```

OG_Param_Exists

Description This function determines whether a parameter has been created.

Syntax

```
FUNCTION OG_Param_Exists
  (param_name  VARCHAR2)
RETURN BOOLEAN;
```

Parameters

param_name	Is the name of the parameter.
Returns TRUE	If the parameter exists.
FALSE	If the parameter does not exist.

OG_Param_Exists Examples

```
/* The following procedure assigns drill-down behavior to a chart, but first verifies
**that the parameter it sets exists (and creates it if it doesn't exist):
*/

PROCEDURE example(chart OG_Object, param_name VARCHAR2) IS
  chelement_group  OG_Object;
BEGIN
  IF NOT OG_Param_Exists(Param_Name) THEN
    OG_Set_Param(Param_Name, 10);
  END IF;

  chelement_group := OG_Get_Object('Sal_Bars', chart);
  OG_Set_Setparam(Chelement_Group, param_name);
  OG_Set_Keycol(Chelement_Group, 'DEPTNO');
END;
```

OG_Set_Param

Description This procedure sets the value of the specified parameter. If the parameter does not exist, it will be created.

Syntax

```
PROCEDURE OG_Set_Param                       date
  (param_name    VARCHAR2,
   param_value   DATE,
   param_format  VARCHAR2  :=  'DD-MON-YY');
PROCEDURE OG_Set_Param                       number
  (param_name   VARCHAR2,
   param_value  NUMBER);
PROCEDURE OG_Set_Param                       char
  (param_name   VARCHAR2,
   param_value  VARCHAR2);
```

Parameters

param_name	Is the name of the parameter whose value you want to set.
param_value	Is the value to which the parameter will be set.
param_format	Is the format mask used to interpret *param_value* for date parameters.

Usage Notes This procedure is useful when you want to reference the parameter within a program unit that is in a library. Bind references do not compile within the context of a library; however, this procedure does.

OG_Set_Param Examples

```
/* The following procedure gets the value of the parameter
**'priority', and increases it by 1:
*/

PROCEDURE example IS
  val  NUMBER;
BEGIN
  val := OG_Get_Num_Param('Priority');
  OG_Set_Param('Priority', val + 1);
END;
```

Query Built-ins

OG_Append_Row
OG_Clear_Query
OG_Data_Changed
OG_Data_Queried
OG_Destroy (Query)
OG_Execute_Query
OG_Get_Charcell
OG_Get_Datecell
OG_Get_Newrows
OG_Get_Numcell
OG_Get_Query
OG_Get_Schema
OG_Insert_Column
OG_Make_Query
OG_Next_Row
OG_Numcols
OG_Numrows
OG_Set_Charcell
OG_Set_Datecell
OG_Set_Numcell
OG_Set_Schema
OG_Start_From

OG_Append_Row

Description This procedure adds the current row buffer to the bottom of a custom query.

Syntax

```
PROCEDURE OG_Append_Row
  (query_hdl  OG_Query);
```

Parameters

query_hdl	Is the handle to the query to which the row buffer is appended.

Usage Notes Specify the contents of the row buffer using OG_Set_Charcell, OG_Set_Datecell, and OG_Set_Numcell.

OG_Append_Row Examples

```
/* Suppose you want to create a custom query using the ENAME, SAL, and
**HIREDATE columns in the existing query 'query0' as a basis.  However, in the
**new query, you want to double every SAL value.  The following procedure
**is a custom query procedure you could use:
*/

PROCEDURE OGQUERYPROC0(query IN OG_Query) IS
  other_ename     VARCHAR2(10);
  other_sal       NUMBER(7,2);
  other_query     OG_Query;
  other_hiredate  DATE;
  row_count       NUMBER;
BEGIN
  OG_Clear_Query(Query);

  other_query := OG_Get_Query('Query0');
```

```
   row_count := OG_Numrows(Other_Query, OG_Newdata);
   OG_Start_From(Other_Query, OG_Newdata, 0);

   FOR i IN 0..row_count loop
     other_ename := OG_Get_Charcell(Other_Query, 'ENAME');
     other_sal := OG_Get_Numcell(Other_Query, 'SAL');
     other_hiredate := OG_Get_Numcell(Other_Query, 'HIREDATE');
     OG_Set_Charcell(Query, 'ENAME', other_ename);
     OG_Set_Numcell(Query, 'SAL', other_sal * 2);
     OG_Set_Datecell(Query, 'HIREDATE', other_hiredate);
     OG_Append_Row(Query);
     OG_Next_Row(Other_Query, OG_Newdata);
   END LOOP;

 END;
```

OG_Clear_Query

Description This procedure removes all rows of data from the specified query.

Syntax

```
 PROCEDURE OG_Clear_Query
   (query_hdl  OG_Query);
```

Parameters

query_hdl	Is the handle to the query to clear..

OG_Clear_Query Examples

```
/* Suppose you want to create a custom query using the ENAME, SAL, and
**HIREDATE columns in the existing query 'query0' as a basis.  However, in the
**new query, you want to double every SAL value.  The following procedure
**is a custom query procedure you could use:
*/

PROCEDURE OGQUERYPROC0(query IN OG_Query) IS
  other_ename     VARCHAR2(10);
  other_sal       NUMBER(7,2);
  other_query     OG_Query;
  other_hiredate  DATE;
  row_count       NUMBER;
BEGIN
  OG_Clear_Query(Query);

  other_query := OG_Get_Query('Query0');
  row_count := OG_Numrows(Other_Query, OG_Newdata);
  OG_Start_From(Other_Query, OG_Newdata, 0);

  FOR i IN 0..row_count loop
    other_ename := OG_Get_Charcell(Other_Query, 'ENAME');
    other_sal := OG_Get_Numcell(Other_Query, 'SAL');
    other_hiredate := OG_Get_Numcell(Other_Query, 'HIREDATE');
    OG_Set_Charcell(Query, 'ENAME', other_ename);
    OG_Set_Numcell(Query, 'SAL', other_sal * 2);
    OG_Set_Datecell(Query, 'HIREDATE', other_hiredate);
    OG_Append_Row(Query);
    OG_Next_Row(Other_Query, OG_Newdata);
  END LOOP;

END;
```

OG_Data_Changed

Description This function compares the old data and new data resulting from the most recent call to OG_Execute_Query for the specified query. It returns TRUE if the data sets differ, and FALSE if they do not.

Syntax

```
FUNCTION OG_Data_Changed
  (query_hdl  OG_Query)
RETURN BOOLEAN;
```

Parameters

query_hdl	Is the handle to the query.
Returns TRUE	If the data has changed.
FALSE	If the data has not changed.

Usage Notes This function compares the following for the old and new data, stopping as soon as a discrepancy is detected:

1 the number of rows returned

2 the query's schema

3 a cell-by-cell comparison of data (note that this comparison can be time-consuming for large sets of data)

OG_Data_Changed Examples

```
/* Suppose you want to update a chart periodically, but only if the
**data has changed.  You could write the following timer trigger:
```

```
*/

PROCEDURE my_timer IS
   my_query   OG_Query;
   my_chart   OG_Object;
BEGIN
   my_query:=OG_Get_Query('Emp_Query');
   OG_Execute_Query(My_Query);
   IF OG_Data_Changed(My_Query) THEN
      my_chart:=OG_Get_Object('Emp_Chart');
      OG_Update_Chart(My_Chart, OG_All_Chupda);
   END IF;
END;
```

OG_Data_Queried

Description This function determines if the specified data category was queried by the most recent call to OG_Execute_Query for the specified query.

Syntax

```
FUNCTION OG_Data_Queried
  (query_hdl   OG_Query,
   which_data  NUMBER)
RETURN BOOLEAN;
```

Parameters

query_hdl	Is the handle to the query.
which_data	Specifies whether the status of the old data or the new data should be checked. Graphics Builder provides two built-in numeric constants that may be used as values for this argument: OG_Newdata and OG_Olddata.

Returns TRUE	If the data has queried.
FALSE	If the data has not been queried.

Usage Notes If the query has not been executed by OG_Execute_Query, neither data category will have been queried, and this function will return FALSE. If the query has been executed exactly once, this function will return TRUE for the new data and FALSE for the old. If the query has been executed more than once, this function will always return TRUE.

OG_Data_Queried Examples

```
/* Suppose you want to use OG_Data_Changed to check if a query's
**data has changed, and then update a chart that uses that query.
**Before you do so, you may want to make sure that both the old and
**new data for the query have been queried.
*/

PROCEDURE check_and_update IS
   my_query   OG_Query;
   my_chart   OG_Object;
BEGIN
   my_query:=OG_Get_Query('Sales Query');
   IF OG_Data_Queried(My_Query, OG_Olddata) AND
OG_Data_Queried(My_Query, OG_Newdata) THEN
      IF data_changed(my_query) THEN
         my_chart:=OG_Get_Object('Sales Chart');
         OG_Update_Chart(My_Chart, OG_All_Chupda);
      END IF;
   END IF;
END;
```

OG_Destroy (Query)

Description This procedure destroys the specified query.

Syntax

```
PROCEDURE OG_Destroy
  (query_hdl  OG_Query);
```

Parameters

query_hdl	Is the handle to the query to destroy.

OG_Destroy (Query) Examples

```
/* The following procedure destroys the specified query:
*/

PROCEDURE destroy_query(query_name VARCHAR2) IS
  query  OG_Query;
BEGIN
  query := OG_Get_Query(Query_Name);
  OG_Destroy(Query);
END;
```

OG_Execute_Query

Description This procedure executes the specified query and stores the results internally.

Syntax

```
PROCEDURE OG_Execute_Query
  (query_hdl  OG_Query);
```

Parameters

query_hdl	Is the handle to the query to execute.

Usage Notes The query must be defined in the Builder. If the query requires database access and the database is not connected, the OG_No_Database_Connection exception is raised. Note that this procedure only retrieves data; it does not apply the data to a chart, nor does it manipulate data in any other way.
For each query, two sets of the data are stored: the current results of the query ("new" data), and the previous results of the query ("old" data). This makes it possible to perform operations that depend on changing data, such as updating a chart only if the data have changed since the last time the query was executed. Other built-in procedures and functions that allow you to manipulate and examine data let you specify which data set you want to use.
If a query has not been executed via a call to this procedure, neither the old data nor the new data for the query will exist. The first time the query is executed, the results are stored as new data, but the old data still does not exist. Subsequently, each time the query is executed the old data is discarded, the existing new data becomes the old data, and the latest results of the query are stored as new data.
Each time a query is executed, an implicit cursor is created for the new data. (Several other procedures and functions allow you to manipulate this cursor and examine the data.) When new data is reclassified as old data, its cursor (and the cursor's current position in the data list) remains with it. Note, however, that a new cursor does not automatically point to a valid row of data. To prepare the cursor for use, use OG_Start_From.

OG_Execute_Query Examples

```
/* Suppose you want to update a chart periodically.
**You could write the following timer trigger:
*/

PROCEDURE every_30_secs IS
   the_query   OG_Query;
   the_chart   OG_Object;
BEGIN
   the_query:=OG_Get_Query('Emp_Query');
   the_chart:=OG_Get_Object('Emp_Chart');
   OG_Execute_Query(The_Query);
   OG_Update_Chart(The_Chart, OG_All_Chupda);
END;
```

OG_Get_Charcell

Description This function returns a character data value for the specified query, in the current row of data, for the specified column.

Syntax

```
FUNCTION OG_Get_Charcell
  (query_hdl   OG_Query,
   col_name    VARCHAR2)
RETURN VARCHAR2;
```

```
FUNCTION OG_Get_Charcell
  (query_hdl   OG_Query,
   which_data  NUMBER,
   col_name    VARCHAR2)
RETURN VARCHAR2;
```

Parameters

query_hdl	Is the handle to the query containing the data to return.
col_name	Is the column name containing the data to return.
which_data	Specifies whether the cell value is retrieved from the old data or the new data. If not specified, the value of this argument defaults to OG_Newdata. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means the cell value is retrieved from the new data. **OG_Olddata** Means the cell value is retrieved from the old data.

Returns The contents of the specified data cell.

Usage Notes The current row is determined by the query's implicit cursor, which is initially created when OG_Execute_Query is used to execute the query.

If you picture the query results displayed in the Data table, this function returns the data contained in the cell at the intersection of the current row and the specified column.

You may use this function to return data values from columns that are of type CHAR, VARCHAR2, or RAW.

OG_Get_Charcell Examples

```
/*Suppose that you have a chart that
**displays employee salaries. The following
**procedure uses a format trigger to paint
**a specific employee's salary column
**yellow (in this case, "Scott’s").
*/
PROCEDURE CharCell (elem IN og_object,
                        query IN og_query) IS
  ename varchar2(10);
BEGIN
  ename := og_get_charcell(query, OG_NEWDATA, 'ename');
  if ename = 'SCOTT' then
     og_set_bfcolor(elem, 'yellow');
  end if;
END;
```

OG_Get_Datecell

Description This function returns the date data value for the specified query, in the current row of data, for the specified column.

Syntax

```
FUNCTION OG_Get_Datecell
  (query_hdl   OG_Query,
   which_data  NUMBER,
   col_name    VARCHAR2)
RETURN DATE;
```

Parameters

query_hdl	Is the handle to the query containing the data to return.
col_name	Is the column name containing the data to return.
which_data	Specifies whether the cell value is retrieved from the old data or the new data. If not specified, the value of this argument defaults to OG_Newdata. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means the cell value is retrieved from the new data. **OG_Olddata** Means the cell value is retrieved from the old data.

Returns The contents of the specified data cell.

Usage Notes The current row is determined by the query's implicit cursor, which is initially created when OG_Execute_Query is used to execute the query.

If you picture the query results displayed in a data table, this function will return the data contained in the cell at the intersection of the current row and the specified column.

OG_Get_Datecell Examples

```
/* Suppose you have a bar chart that
**shows the hire dates of several employees.
**The following format trigger changes the
**column color for employees with
**a HireDate prior to '28-SEP-81'
**to green.
*/
 PROCEDURE DateCell (elem IN og_object,
                        query IN og_query) IS
  HireDate date;
BEGIN
  HireDate := og_get_datecell(query, OG_NEWDATA, 'HIREDATE');
  if HireDate < '28-SEP-81' then
     og_set_bfcolor(elem, 'green');
  end if;
END;
```

OG_Get_Newrows

Description This function determines the number of new rows of data appended to a query.

Syntax

```
FUNCTION OG_Get_Newrows
  (query  OG_Query)
RETURN NUMBER;
```

Parameters

query	Is a handle the query.

Returns The number of rows appended to the query the last time it was executed.
Usage Notes This function is useful only if the query properties specify that new data is appended to old data. If the new data replaces old data, this function returns the same result as OG_Numrows.

OG_Get_Newrows Examples

```
/* Suppose you have a query that appends new data old data, but you want to
**know a cell value for the first new row returned.  The following function sets the
**query's cursor to start at the first new row returned:
*/

FUNCTION example(query OG_Query) RETURN CHAR IS
  total_rows  NUMBER;
  new_rows    NUMBER;
  new_name    VARCHAR2(10);
BEGIN
  OG_Execute_Query(Query);
  total_rows := OG_Numrows(Query, OG_Newdata);
  new_rows := OG_Get_Newrows(Query);
  OG_Start_From(Query, OG_Newdata, total_rows - new_rows);

  new_name := OG_Get_Charcell(Query, 'ENAME');
  RETURN(new_name);
END;
```

OG_Get_Numcell

Description This function returns the numeric data value for the specified query, in the current row of data, for the specified column.
Syntax
```
FUNCTION OG_Get_Numcell
  (query_hdl   OG_Query,
   col_name    VARCHAR2)
RETURN NUMBER;
FUNCTION OG_Get_Numcell
  (query_hdl   OG_Query,
   which_data  NUMBER,
   col_name    VARCHAR2)
RETURN NUMBER;
```
Parameters

query_hdl	Is the handle to the query containing the data to return.
col_name	Is the column name containing the data to return.
which_data	Specifies whether the cell value is retrieved from the old data or the new data. If not specified, the value of this argument defaults to OG_Newdata. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means the cell value is retrieved from the new data. **OG_Olddata** Means the cell value is retrieved from the old data.

Returns The contents of the specified data cell.
Usage Notes The current row is determined by the query's implicit cursor, which is initially created when OG_Execute_Query is used to execute the query.

If you picture the query results displayed in a data table, this function will return the data contained in the cell at the intersection of the current row and the specified column.

OG_Get_Numcell Examples

```
/* The following is an example of a
**format trigger that changes the color
**of the chosen chart element (e.g. pie
**slice or bar) to red if its value is
**greater than 200.
/*
PROCEDURE format_point (elem IN og_object, query IN
    og_query) IS
  st_price NUMBER;
BEGIN
  st_price:=OG_GET_NUMCELL (query, OG_NEWDATA, 'sell_pr');
IF st_price > 200 THEN
  OG_SET_BFCOLOR (elem,'red');
END IF;
END;
```

OG_Get_Query

Description This function returns a handle to the specified query.

Syntax

```
FUNCTION OG_Get_Query
  (query_name  VARCHAR2)
RETURN OG_Query;
```

Parameters

query_name	Is the name of the query whose handle is returned. **Note:** QUERY_NAME is case-sensitive.

Returns A handle to the specified query.

Usage Notes If the query does not exist, this function returns a null handle.

OG_Get_Query Examples

```
/* Suppose you want to update a chart periodically.
**You could write the following timer trigger:
*/

PROCEDURE every_30_secs IS
   the_query   OG_Query;
   the_chart   OG_Object;
BEGIN
   the_query:=OG_Get_Query('Emp_Query');
   the_chart:=OG_Get_Object('Emp_Chart');
   OG_Execute_Query(The_Query);
   OG_Update_Chart(The_Chart, OG_All_Chupda);
```

OG_Get_Schema

Description This function returns information about the schema of a particular column in a query.

Syntax

```
FUNCTION OG_Get_Schema
  (query_hdl   OG_Query,
   which_data  NUMBER,
   col_num     NUMBER)
RETURN OG_Colschema;
```

Parameters

query_hdl	Is the handle to the query that contains the column.
which_data	Specifies whether the column whose schema is retrieved exists in the query's old data or new data. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means the column exists in the query's new data. **OG_Olddata** Means the column exists in the query's old data.
col_num	Specifies which column's schema is retrieved. The first column's number is 0, the second is 1, etc.

Returns The schema of the column in the specified query.

OG_Get_Schema Examples

```
/* Suppose you want to query a database table, and then use the name
**of the first column elsewhere in your application.  Assume you have
**defined a parameter named `my_query' that is of type CHAR,
**and that you have defined the following SQL query named `query0':
*/

&my_query
/* The following function takes a table name as an argument and
**returns the name of the table's first column:
*/
FUNCTION get_col_name (table_name IN VARCHAR2) RETURN VARCHAR2 IS
   my_schema    OG_Colschema;
   star_query   OG_Query;
BEGIN
   :my_query:='select * from ' || table_name;
   star_query:=OG_Get_Query('Query0');
   OG_Execute_Query(Star_Query);
   my_schema:=OG_Get_Schema(Star_Query, OG_Newdata, 0);
   RETURN(my_schema.colname);
END;
```

OG_Insert_Column

Description This procedure inserts a column into a custom query.

Syntax

```
PROCEDURE OG_Insert_Column
  (query_hdl  OG_Query,
   indx       NUMBER,
   schema     OG_Colschema);
```

Parameters

query_hdl	Is the handle to the query in which to insert the column.
indx	Is the index at which to insert the new column in the query's column list. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of columns in the query prior to the insertion. The value

	of this argument may also be one of the following built-in constants: **OG_First** Means insert the new column at the beginning of the query's column list (index = 0). **OG_Last** Means insert the new column at the end of the query's column list (index = the number of columns in the query prior to the insertion).
schema	Is the schema of the column to insert.

OG_Insert_Column Examples

```
/* The following procedure creates 'query0', containing the columns ENAME and SAL:
*/

PROCEDURE example IS
  query  OG_Query;
  col    OG_Colschema;
BEGIN
  query := OG_Make_Query('Query0', NULL);
  OG_Set_Querytype(Query, OG_Custom_Qtype);

  col.colname := 'ENAME';
  col.coltype := OG_Char_Coltype;
  col.maxlen := 10;

  OG_Insert_Column(Query, OG_Last, col);

  col.colname := 'SAL';
  col.coltype := OG_Number_Coltype;
  col.precision := 7;
  col.scale := 2;

  OG_Insert_Column(Query, OG_Last, col);

END;
```

OG_Make_Query

Description This function creates a query.

Syntax

```
FUNCTION OG_Make_Query
  (querytype    NUMBER,
   querysource  VARCHAR2
RETURN OG_Query;
```

Parameters

querytype	Is the query type. This value may be one of the following built-in constants: **OG_Custom_Qtype** Means the query is a Custom query. **OG_Exsql_Qtype** Means the query retrieves its data from a text file that contains a SQL SELECT statement. **OG_Prn_Qtype** Means the query is based on a PRN file. **OG_Sql_Qtype** Means the query is a SQL SE.LECT statement. **OG_Sylk_Qtype** Means the query is based on a SYLK file. **OG_Wks_Qtype** Means the query is based on a WKS file.
querysource	Is the source of the query's data. If the data comes from a database, this property should contain the text of the query's SQL SELECT statement. If the data is stored in the

filesystem, this property should contain the path and name of the data file.

Returns A handle to the newly created query.

OG_Make_Query Examples

```
/* The following function creates a SQL query:
*/

FUNCTION example(query_name VARCHAR2) RETURN OG_Query IS
  query   OG_Query;
  qtype   NUMBER;
  qsource VARCHAR2(2000);
BEGIN
  qtype := OG_Sql_Qtype;
  qsource := 'select ename, sal from emp';

  query := OG_Make_Query(Qtype, qsource);

  OG_Set_Name(Query, query_name);
  OG_Execute_Query(Query);
  RETURN(query);
END;
```

OG_Next_Row

Description

Syntax This procedure advances the implicit cursor associated with the specified query ahead to the next row of data.

```
PROCEDURE OG_Next_Row
  (query_hdl   OG_Query,
   which_data  NUMBER);
```

Parameters

query_hdl	Is the handle to the query.
which_data	Specifies whether the old data or the new data should be processed. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means advance the cursor for the query's new data. **OG_Olddata** Means advance the cursor for the query's old data.

Usage Notes If the cursor is pointing to the last row of data in the query, the next call to OG_Next_Row will leave the cursor where it is. The cursor will *not* advance to a non-existent row, and *no* error or exception will be raised. To recognize that the cursor is pointing to the last row of data, you must use OG_Numrows to determine the exact number of rows, then keep track of how many times you use OG_Next_Row to advance the cursor.

OG_Next_Row Examples

```
/* Suppose you want to name each bar in a bar chart so that when
**the user selects one of the bars you can determine which one it is
**by checking its name.  For this example, assume the query for the chart is:
*/

SELECT ENAME, SAL FROM EMP
 /*The following procedure gives each bar the name of its category,
**which in this case is its associated ENAME:
```

```
*/
PROCEDURE name_the_bars (my_chart IN OG_Object, my_query IN
  OG_Query) IS
   bar_rec    OG_Chelement_Ca;
   curr_row   NUMBER;
   total      NUMBER;
   bar_name   VARCHAR2(15);
BEGIN
   OG_Execute_Query(My_Query);
   OG_Start_From(My_Query, OG_Newdata, 0);
   total:=OG_Numrows(My_Query, OG_Newdata);
   FOR curr_row IN 0..total-1 LOOP
      bar_name:=OG_Get_Charcell(My_Query,
        OG_Newdata, 'ENAME');
      bar_rec.chelement_cagr.mask:=OG_None_Graphica;
      bar_rec.chelement_cace.mask:=OG_Name_Chelementa;
      bar_rec.chelement_cace.name:=bar_name;
      OG_Set_Attr(My_Chart, curr_row, 'ENAME', bar_rec);
      OG_Next_Row(My_Query, OG_Newdata);
   END LOOP;
   OG_Update_Chart(My_Chart, OG_All_Chupda);
END;
```

OG_Numcols

Description This function returns the number of columns that exist in a query.

Syntax

```
FUNCTION OG_Numcols
  (query_hdl   OG_Query,
   which_data  NUMBER)
RETURN NUMBER;
```

Parameters

query_hdl	Is the handle to the query.
which_data	Specifies whether the old data or the new data should be checked. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means return the number of columns in the query's new data. **OG_Olddata** Means return the number of columns in the query's old data.

Returns The number of columns in the specified query.

OG_Numcols Examples

```
/* Suppose Reports will pass data to your display, and you want to chart it.
**Since you may not be sure what columns your display will receive,
**you can make your charting procedure generic.  You can write one
**procedure that creates an chart, then pass the query and chart to
**another procedure that inserts the query's columns as fields.  The following
**procedure inserts the columns (it assumes the first column is the independent
**field, and the rest are dependent fields):
*/

PROCEDURE add_columns(the_query OG_Query, the_chart OG_Object) IS
   num_of_cols   NUMBER(1);
   the_field     OG_Field;
   the_column    OG_Colschema;
BEGIN
   OG_Execute_Query(The_Query);
   num_of_cols:=OG_Numcols(The_Query, OG_Newdata);
   FOR i IN 0..num_of_cols-1 LOOP
      the_column:=OG_Get_Schema(The_Query, OG_Newdata, i);
      the_field.colname:=the_column.colname;
      IF i=0 THEN
         the_field.field_type:=OG_Independent;
      ELSE
         the_field.field_type:=OG_Dependent;
         the_field.ftname:='line';
      END IF;
      OG_Insert_Field(The_Chart, the_field, i);
   END LOOP;
   OG_Update_Chart(The_Chart, OG_All_Chupda);
END;
```

OG_Numrows

Description This function returns the number of rows that exist in a query.

Syntax

```
FUNCTION OG_Numrows
  (query_hdl   OG_Query,
   which_data  NUMBER)
RETURN NUMBER;
```

Parameters

query_hdl	Is the handle to the query.
which_data	Specifies whether the old data or the new data should be checked. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means return the number of rows in the query's new data. **OG_Olddata** Means return the number of rows in the query's old data.

Returns The number of rows of data in the specified query.

OG_Numrows Examples

```
/* Suppose you want to name each bar in a bar chart so that when
**the user selects one of the bars you can determine which one it is
**by checking its name.  For this example, assume the query for the chart is:
*/

SELECT ENAME, SAL FROM EMP
 /*The following procedure gives each bar the name of its category,
**which in this case is its associated ENAME:
*/
PROCEDURE name_the_bars(my_chart IN OG_Object, my_query IN
  OG_Query) IS
   bar_rec    OG_Chelement_Ca;
   curr_row   NUMBER;
   total      NUMBER;
   bar_name   VARCHAR2(15);
BEGIN
   OG_Execute_Query(My_Query);
   OG_Start_From(My_Query, OG_Newdata, 0);
   total:=OG_Numrows(My_Query, OG_Newdata);
   FOR curr_row IN 0..total-1 LOOP
      bar_name:=OG_Get_Charcell(My_Query, OG_Newdata,
        'ENAME');
      bar_rec.chelement_cagr.mask:=OG_None_Graphica;
      bar_rec.chelement_cace.mask:=OG_Name_Chelementa;
      bar_rec.chelement_cace.name:=bar_name;
      OG_Set_Attr(My_Chart, curr_row, 'ENAME', bar_rec);
      OG_Next_Row(My_Query, OG_Newdata);
   END LOOP;
   OG_Update_Chart(My_Chart, OG_All_Chupda);
```

```
END;
```

OG_Set_Charcell

Description This procedure sets the value of a CHAR cell in the row buffer.

Syntax

```
PROCEDURE OG_Set_Charcell
  (query_hdl   OG_Query,
   col_name    VARCHAR2
   cell_value  VARCHAR2);
```

Parameters

query_hdl	Is the handle to the query in which to set the cell value.
col_name	Is the name of the column containing the cell to set.
cell_value	Is the value that the cell will contain.

Usage Notes Once you set the values for all the cells in the buffer, use OG_Append_Row to add the buffer as a new row at the end of a custom query.

OG_Set_Charcell Examples

```
/* Suppose you want to create a custom query using the ENAME, SAL,
**and HIREDATE columns in the existing query 'query0' as a basis.
**However, in the new query, you want to double every SAL value.
**The following procedure is a custom query procedure you could use:
*/

PROCEDURE OGQUERYPROC0(query IN OG_Query) IS
  other_ename     VARCHAR2(10);
  other_sal       NUMBER(7,2);
  other_query     OG_Query;
  other_hiredate  DATE;
  row_count       NUMBER;
BEGIN
  OG_Clear_Query(Query);

  other_query := OG_Get_Query('Query0');
  row_count := OG_Numrows(Other_Query, OG_Newdata);
  OG_Start_From(Other_Query, OG_Newdata, 0);

  FOR i IN 0..row_count-1 loop
    other_ename := OG_Get_Charcell(Other_Query, 'ENAME');
    other_sal := OG_Get_Numcell(Other_Query, 'SAL');
    other_hiredate := OG_Get_Numcell(Other_Query, 'HIREDATE');
    OG_Set_Charcell(Query, 'ENAME', other_ename);
    OG_Set_Numcell(Query, 'SAL', other_sal * 2);
    OG_Set_Datecell(Query, 'HIREDATE', other_hiredate);
    OG_Append_Row(Query);
    OG_Next_Row(Other_Query, OG_Newdata);
  END LOOP;

END;
```

OG_Set_Datecell

Description This procedure sets the value of a DATE cell in the row buffer.

Syntax

```
PROCEDURE OG_Set_Datecell
  (query_hdl   OG_Query,
   col_name    VARCHAR2
   cell_value  DATE);
```

Parameters

query_hdl	Is the handle to the query in which to set the cell value.
col_name	Is the name of the column containing the cell to set.
cell_value	Is the value that the cell will contain.

Usage Notes Once you set the values for all the cells in the buffer, use OG_Append_Row to add the buffer as a new row at the end of a custom query.

OG_Set_Datecell Examples

```
/* Suppose you want to create a custom query using the ENAME, SAL,
**and HIREDATE columns in the existing query 'query0' as a basis.
**However, in the new query, you want to double every SAL value.
**The following procedure is a custom query procedure you could use:
*/

PROCEDURE OGQUERYPROC0(query IN OG_Query) IS
  other_ename     VARCHAR2(10);
  other_sal       NUMBER(7,2);
  other_query     OG_Query;
  other_hiredate  DATE;
  row_count       NUMBER;
BEGIN
  OG_Clear_Query(Query);

  other_query := OG_Get_Query('Query0');
  row_count := OG_Numrows(Other_Query, OG_Newdata);
  OG_Start_From(Other_Query, OG_Newdata, 0);

  FOR i IN 0..row_count-1 loop
    other_ename := OG_Get_Charcell(Other_Query, 'ENAME');
    other_sal := OG_Get_Numcell(Other_Query, 'SAL');
    other_hiredate := OG_Get_Numcell(Other_Query, 'HIREDATE');
    OG_Set_Charcell(Query, 'ENAME', other_ename);
    OG_Set_Numcell(Query, 'SAL', other_sal * 2);
    OG_Set_Datecell(Query, 'HIREDATE', other_hiredate);
    OG_Append_Row(Query);
    OG_Next_Row(Other_Query, OG_Newdata);
  END LOOP;

END;
```

OG_Set_Numcell

Description This procedure sets the value of a NUMBER cell in the row buffer.

Syntax

```
PROCEDURE OG_Set_Numcell
  (query_hdl   OG_Query,
   col_name    VARCHAR2,
   cell_value  NUMBER);
```

Parameters

query_hdl	Is the handle to the query in which to set the cell value.
col_name	Is the name of the column containing the cell to set.
cell_value	Is the value that the cell will contain.

Usage Notes Once you set the values for all the cells in the buffer, use OG_Append_Row to add the buffer as a new row at the end of a custom query.

OG_Set_Numcell Examples

```
/* Suppose you want to create a custom query using the ENAME, SAL,
**and HIREDATE columns in the existing query 'query0' as a basis.
**However, in the new query, you want to double every SAL value.
**The following procedure is a custom query procedure you could use:
*/

PROCEDURE OGQUERYPROC0(query IN OG_Query) IS
  other_ename     VARCHAR2(10);
  other_sal       NUMBER(7,2);
  other_query     OG_Query;
  other_hiredate  DATE;
  row_count       NUMBER;
BEGIN
  OG_Clear_Query(Query);

  other_query := OG_Get_Query('Query0');
  row_count := OG_Numrows(Other_Query, OG_Newdata);
  OG_Start_From(Other_Query, OG_Newdata, 0);
  FOR i IN 0..row_count-1 loop
    other_ename := OG_Get_Charcell(Other_Query, 'ENAME');
    other_sal := OG_Get_Numcell(Other_Query, 'SAL');
    other_hiredate := OG_Get_Numcell(Other_Query, 'HIREDATE');
    OG_Set_Charcell(Query, 'ENAME', other_ename);
    OG_Set_Numcell(Query, 'SAL', other_sal * 2);
    OG_Set_Datecell(Query, 'HIREDATE', other_hiredate);
    OG_Append_Row(Query);
    OG_Next_Row(Other_Query, OG_Newdata);
  END LOOP;

END;
```

OG_Set_Schema

Description This procedure sets the schema of a column in a custom query.

Syntax

```
PROCEDURE OG_Set_Schema
  (query_hdl  OG_Query,
   col_num    NUMBER,
   schema     OG_Colschema);
```

Parameters

query_hdl	Is the handle to the query whose schema you want to set.
col_num	Is the index of the column that you want to set.
schema	Is the new schema to which you want to set the column.

OG_Set_Schema Examples

```
/* The following procedure changes the name of the fist column
**in a custom query from ENAME to EMPLOYEE:
*/

PROCEDURE example(query OG_Query) IS
  schema  OG_Colschema;
BEGIN
  schema.colname := 'EMPLOYEE';
  schema.coltype := OG_Char_Coltype;
  schema.maxlen := 10;

  OG_Set_Schema(Query, 0, schema);

END;
```

OG_Start_From

Description This procedure causes the implicit cursor associated with the specified query to point to the specified row of data.

Syntax

```
PROCEDURE OG_Start_From
  (query_hdl   OG_Query,
   which_data  NUMBER,
   start_row   NUMBER);
```

Parameters

query_hdl	Is the handle to the query.
which_data	Specifies whether the old data or the new data should be processed. The value of this argument may be one of the following built-in constants: **OG_Newdata** Means set the cursor for the query's new data. **OG_Olddata** Means set the cursor for the query's old data.
start_row	Is the row number at which to position the cursor.

Usage Notes To point to the very first row of data, use an offset value of 0. The last row of data will have an offset equal to the value returned by OG_Numrows minus one. Note that each time OG_Execute_Query is used to execute the query, the cursor position will be discarded and must be set again. Be aware that the number of rows retrieved by a query may change each time the query is executed.

OG_Start_From Examples

```
/* Suppose you want to name each bar in a bar chart so that
**when the user selects one of the bars you can determine
**which one it is by checking its name.  For this example,
**assume the query for the chart is:
*/

SELECT ENAME, SAL FROM EMP
 /*The following procedure gives each bar the name of its
**category, which in this case is its associated ENAME:
```

```
*/
PROCEDURE name_the_bars (my_chart IN OG_Object, my_query IN
  OG_Query) IS
   bar_rec    OG_Chelement_Ca;
   curr_row   NUMBER;
   total      NUMBER;
   bar_name   VARCHAR2(15);
BEGIN
   OG_Execute_Query(My_Query);
   OG_Start_From(My_Query, OG_Newdata, 0);
   total:=OG_Numrows(My_Query, OG_Newdata);
   FOR curr_row IN 0..total-1 LOOP
      bar_name:=OG_Get_Charcell(My_Query, OG_Newdata,
        'ENAME');
      bar_rec.chelement_cagr.mask:=OG_None_Graphica;
      bar_rec.chelement_cace.mask:=OG_Name_Chelementa;
      bar_rec.chelement_cace.name:=bar_name;
      OG_Set_Attr(My_Chart, curr_row, 'ENAME', bar_rec);
      OG_Next_Row(My_Query, OG_Newdata);
   END LOOP;
   OG_Update_Chart(My_Chart, OG_All_Chupda);
END;
```

Sound Built-ins

OG_Destroy (Sound)
OG_Export_Sound
OG_Get_Sound
OG_Import_Sound
OG_Make_Sound
OG_Play_Sound
OG_Record_Sound
OG_Stop_Sound

OG_Destroy (Sound)

Description This procedure destroys the specified sound.

Syntax

```
PROCEDURE OG_Destroy
  (sound_hdl  OG_Sound);
```

Parameters

sound_hdl	Is the handle to the sound to destroy.

OG_Destroy (Sound) Examples

```
/* The following procedure destroys the specified sound:
*/

PROCEDURE destroy_sound(sound_name VARCHAR2) IS
  sound  OG_Sound;
BEGIN
  sound := OG_Get_Sound(Sound_Name);
  OG_Destroy(Sound);
END;
```

OG_Export_Sound

Description This procedure exports a sound.

Syntax

```
PROCEDURE OG_Export_Sound
  (name        VARCHAR2,
   repository  NUMBER,
   format      NUMBER,
   sound_hdl   OG_Sound);
```

Parameters

name	Is the name to which the sound will be exported. If the sound is to be stored in the database, this argument should contain only the name of the sound module. If the sound is to be stored in the file system, this argument should contain the absolute or relative pathname of the sound file.
repository	Specifies whether the sound is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the sound is to be stored in the database. **OG_Filesystem** Means the sound is to be stored in the file system.
format	Specifies the format in which the sound is exported. The value of this argument may be one of the following built-in constants: **OG_Aiff_Sformat** Means the sound is saved in the AIFF format. **OG_Aiffc_Sformat** Means the sound is saved in the Audio Interchange File Format-c. **OG_Au_Sformat** Means the sound is saved in the SUN au format. **OG_Wave_Sformat** Means the sound is saved in the PCM WAVE format.
sound_hdl	Is the handle to the sound that will be exported.

OG_Export_Sound Examples

```
/* Suppose you want to export the sound named `sound0' to the AIFF-c file
**`my_sound' so that you can later import it into some other application.
**The following procedure does this:
*/

PROCEDURE export_the_sound IS
   the_sound   OG_Sound;
BEGIN
   the_sound:=OG_Get_Sound('Sound0');
   OG_Export_Sound('My_Sound', OG_Filesystem,
OG_Aiffc_Sformat, the_sound);
END;
```

OG_Get_Sound

Description This function returns a handle to the specified sound.

Syntax

```
FUNCTION OG_Get_Sound
   (sound_name  VARCHAR2)
RETURN OG_Sound;
```

Parameters

sound_name	Is the name of the sound whose handle should be returned.

Returns A handle to the specified sound. If the sound does not exist, this function will return a null handle.

OG_Get_Sound Examples

```
/* Suppose you want to play a warning sound, to indicate
**low inventory or an illegal action by the user.
*/

PROCEDURE warning IS
   the_sound   OG_Sound;
BEGIN
   the_sound:=OG_Get_Sound('Warning_Snd');
   OG_Play_Sound(The_Sound);
END;
```

OG_Import_Sound

Description This procedure imports a sound from the database or a file.

Syntax

```
FUNCTION OG_Import_Sound
  (name         VARCHAR2,
   repository  NUMBER,
   format      NUMBER,
   sound_name  VARCHAR2)
RETURN OG_Sound;
```

Parameters

name	Is the name of the sound as it is stored. If the sound is stored in the database, this argument

	should contain only the name of the sound. If the sound is stored in the file system, this argument should contain the absolute or relative pathname of the sound file.
repository	Specifies whether the sound is stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the sound is stored in the database. **OG_Filesystem** Means the sound is stored in the file system.
format	Specifies the format in which the sound is saved. The value of this argument may be one of the following built-in constants: **OG_Aiff_Sformat** Means the sound is saved in the AIFF format. **OG_Aiffc_Sformat** Means the sound is saved in the AIFF-c format. **OG_Any_Sformat** Means Graphics Builder automatically determines the sound's format. **Note:** Specify this format if your sound was exported in the Oracle Format (now obsolete). **OG_Au_Sformat** Means the sound is saved in the SUN AU format. **OG_Oracle_Sformat** Means the sound is saved in the Oracle Format, used by other Oracle products. **OG_Wave_Sformat** Means the sound is saved in the WAV format.
sound_name	Is the name that Graphics Builder will assign to the sound. If another sound already has this name, Graphics Builder replaces it with the imported sound.

Returns A handle to the imported sound.

OG_Import_Sound Examples

```
/* Suppose you want to import the contents of the AIFF-c file
**`my_sound' into your display as the sound named `sound0'.
**The following procedure does this:
*/

PROCEDURE import_the_sound IS
   the_sound   OG_Sound;
BEGIN
   the_sound:=OG_Import_Sound('My_Sound', OG_Filesystem,
OG_Aiffc_Sformat, 'sound0');
END;
```

OG_Make_Sound

Description This function creates a sound from data stored in a database table.

Syntax

```
FUNCTION OG_Make_Sound
  (query       OG_Query,
   which_data  NUMBER,
   colname     VARCHAR2)
RETURN OG_Sound;
```

Parameters

query	Is the handle to the query that retrieves the sound from a table in a database. Note that this table must be a user table, and not one the private tables used by Graphics Builder when you save or export a module to the database.
which_data	Specifies whether the sound to be created is contained in a query's new or old data set. Graphics Builder provides two built-in numeric constants that may be used as values for this attribute: **OG_Newdata** Means the sound is contained in the query's new data set. **OG_Olddata** Means the sound is contained in the query's old data set.
colname	Is the name of the query column that contains the sound data. The sound that is created is the one contained in the query cell at the intersection of the column specified by this attribute and the row pointed to by the query's cursor.

Returns A handle to the newly created sound.

OG_Make_Sound Examples

```
/* The following function creates a sound from data in the sixth
**row of the query 'sound_query' in the column SOUND_COLUMN:
*/

FUNCTION example(sound_name VARCHAR2) RETURN OG_Sound IS
  query  OG_Query;
  sound  OG_Sound;
BEGIN
  query := OG_Get_Query('Sound_Query');
  OG_Execute_Query(Query);
  OG_Start_From(Query, OG_Newdata, 5);
  sound := OG_Make_Sound(Query, OG_Newdata, 'SOUND_COLUMN');

  OG_Set_Name(Sound, sound_name);
  RETURN(sound);
END;
```

OG_Play_Sound

Description This procedure plays the specified sound through the sound output device specified in your preferences.

Syntax

```
PROCEDURE OG_Play_Sound
  (sound_hdl  OG_Sound);
```

Parameters

sound_hdl	Is the handle to the sound to be played.

OG_Play_Sound Examples

```
/* Suppose you want to play a warning sound, to indicate
**low inventory or an illegal action by the user.
*/

PROCEDURE warning IS
   the_sound   OG_Sound;
BEGIN
   the_sound:=OG_Get_Sound('Warning_Snd');
   OG_Play_Sound(The_Sound);
END;
```

OG_Record_Sound

Description This procedure shows the sound dialog box and allows the user to record a new sound.

Syntax

```
PROCEDURE OG_Record_Sound
  (sound_hdl  OG_Sound);
```

Parameters

sound_hdl	Is the handle to the sound.

Usage Notes The new sound overwrites the sound pointed to by *sound_hdl*. In addition, *sound_hdl* must point to a sound previously created either in the Builder, or by the built-in functions OG_Import_Sound and OG_Make.

OG_Record_Sound Examples

```
/* Suppose you want the user to record a sound to be played as a warning
**when data changes.  You could write the following procedure:
*/

PROCEDURE record_warning IS
   warn_sound   OG_Sound;
BEGIN
   warn_sound:=OG_Get_Sound('Warning');
   IF not OG_Isnull(Warn_Sound) THEN
      OG_Record_Sound(Warn_Sound);
   END IF;
END;
```

OG_Stop_Sound

Description This procedure cancels the playback of a sound that is in the process of playing.

Syntax

```
PROCEDURE OG_Stop_Sound
  (sound_hdl   OG_Sound);
```

Parameters

sound_hdl	Is the handle to the sound whose playback you want to stop.

OG_Stop_Sound Examples

```
/* Suppose you want to create a button that the user
**can select to cancel a sound that is currently playing.
**The following button procedure does this:
*/

PROCEDURE OGBUTTONPROC0 (buttonobj IN OG_Object, hitobj IN OG_Object, win IN OG_Window,
eventinfo IN OG_Event) IS
  sound  OG_Sound;
BEGIN
  sound := OG_Get_Sound('Alarm');
  OG_Stop_Sound(Sound);
END;
```

Template Built-ins

OG_Clone (Template)
OG_Delete_Ftemp
OG_Delete_Refline
OG_Destroy (Template)
OG_Export_Template
OG_Get_Axis
OG_Get_Ftemp
OG_Get_Refline
OG_Get_Template
OG_Import_Template
OG_Insert_Ftemp
OG_Insert_Refline
OG_Make_Template

OG_Clone (Template)

Description This function creates a new chart template that is identical to the specified template.

Syntax

```
FUNCTION OG_Clone
  (template_hdl  OG_Template,
   name          VARCHAR2)
RETURN OG_Template;
```

Parameters

template_hdl	Is the handle to the chart template to be cloned.
name	Is the name to be given to the new template.

Returns The handle to the newly created template.

OG_Clone (Template) Examples

```
/* Suppose you have created  atemplate, and you want to create another
**identical template without having to again specify the same properties.
*/
```

```
PROCEDURE dup_template(old_template IN OG_Template) IS
   new_template   OG_Template;
BEGIN
   new_template:=OG_Clone(Old_Template);
END;
```

OG_Delete_Ftemp

Description This procedure deletes one or more field template from the specified chart template.

Syntax

```
PROCEDURE OG_Delete_Ftemp
  (template_hdl  OG_Template,
   indx          NUMBER,
   total         NUMBER);
```

Parameters

template_hdl	Is the handle to the chart template.
indx	Is the index of the first field template to delete from the chart template's list of field templates..
total	Is the total number of field template to delete.

OG_Delete_Ftemp Examples

```
 /* The following procedure deletes a column from the template 'template0':
*/

PROCEDURE example(ft_num  NUMBER) IS
  template  OG_Template;
BEGIN
  template := OG_Get_Template('Template0');
  OG_Delete_Ftemp(Template, ft_num, 1);
END;
```

OG_Delete_Refline

Description This procedure deletes one or more reference lines from the specified chart template.

Syntax

```
PROCEDURE OG_Delete_Refline
  (template_hdl  OG_Template,
   indx          NUMBER,
   total         NUMBER);
```

Parameters

template_hdl	Is the handle to the chart template.
indx	Is the index of the first reference line to delete from the chart template's list of reference lines.
total	Is the total number of reference lines to delete.

OG_Delete_Refline Examples

```
 /* The following procedure deletes a reference line template 'template0':
*/

PROCEDURE example(rl_num  NUMBER) IS
  template  OG_Template;
BEGIN
  template := OG_Get_Template('Template0');
  OG_Delete_Refline(Template, rl_num, 1);
END;
```

OG_Destroy (Template)

Description This procedure destroys the specified chart template.

Syntax

```
PROCEDURE OG_Destroy
  (template_hdl  OG_Template);
```

Parameters

template_hdl	Is the handle to the chart template to destroy.

OG_Destroy (Template) Examples

```
 /* The following procedure destroys the specified template:
*/

PROCEDURE destroy_template(template_name VARCHAR2) IS
  template  OG_Template;
BEGIN
  template := OG_Get_Template(Template_Name);
  OG_Destroy(Template);
END;
```

OG_Export_Template

Description This procedure exports a chart template.

Syntax

```
PROCEDURE OG_Export_Template
  (name          VARCHAR2,
   repository    NUMBER,
   template_hdl  OG_Template);
```

Parameters

name	Is the name to which the template will be exported. If the template is to be stored in the database, this argument should contain only the name of the template. If the template is to be stored in the file system, this argument should contain the absolute or relative pathname of the template file.
repository	Specifies whether the template is to be stored in the file system or database. The value of this argument may be one of the following built-in constants: **OG_Db** Means the template is to be stored in the database. **OG_Filesystem** Means the template is to be stored in the file system.
tamplate_hdl	Is the handle to the template that will be exported.

OG_Export_Template Examples

```
 /* Suppose you want to export the chart template named `template0'
**to the file `my_temp' so that you can later import it into some other
**Graphics Builder display.  The following procedure does this:
*/

PROCEDURE export_the_template IS
   the_temp   OG_Template;
BEGIN
   the_temp:=OG_Get_Template('Template0');
   OG_Export_Template('My_Temp', OG_Filesystem, the_temp);
END;
```

OG_Get_Axis

Description This function returns a handle to an axis in a chart template.

Syntax

```
FUNCTION OG_Get_Axis
  (template_hdl  OG_Template,
   which_axis    NUMBER)
RETURN OG_Axis;
```

Parameters

template_hdl	Is the handle to the chart template containing the axis whose handle should be returned.
which_axis	Specifies which axis will be returned. The value of this argument may be one of the following built-in constants: **OG_X_Axis** **OG_Y1_Axis** **OG_Y2_Axis**

Returns A handle to the specified axis. If the specified button procedure does not exist, this function will return a null handle.

OG_Get_Axis Examples

```
 /* The following function returns a handle to the specified template's X axis:
*/

FUNCTION example(template_name VARCHAR2) RETURN OG_Axis IS
  template  OG_Template;
  axis      OG_Axis;
BEGIN
  template := OG_Get_Template(Template_Name);
  axis := OG_Get_Axis(Template, OG_X_Axis);
  RETURN(axis);
END;
```

OG_Get_Ftemp

Description This function returns a handle to a field template within a chart template.

Syntax

```
FUNCTION OG_Get_Ftemp
```

```
  (template_hdl  OG_Template,
   indx          NUMBER)
RETURN OG_Ftemp;
```

Parameters

template_hdl	Is the handle to the chart template containing the field template that should be returned.
indx	Is the index of the field template in the chart's field template list to be returned.

Returns The attributes of the specified field template.

OG_Get_Ftemp Examples

```
 /* The following function returns the handle to the
**first field template in the specified chart template:
*/

FUNCTION example(temp_name VARCHAR2) RETURN OG_Ftemp IS
  template  OG_Template;
  ftemp   OG_Ftemp;
BEGIN
  template := OG_Get_Template(Temp_Name);
  ftemp := OG_Get_Ftemp(Template, 0);
  RETURN(ftemp);
END;
```

OG_Get_Refline

Description This function returns a handle to a reference line in a chart template:

Syntax

```
FUNCTION OG_Get_Refline
  (template_hdl  OG_Template,
   indx          NUMBER)
RETURN OG_Refline;
```

Parameters

template_hdl	Is the handle to the chart template containing the reference line that should be returned.
indx	Is the index of the reference line in the chart's reference line list to be returned.

Returns A handle to the specified reference line.

OG_Get_Refline Examples

```
 /* The following function returns the handle to the
**first reference line in the specified chart template:
*/

FUNCTION example(temp_name VARCHAR2) RETURN OG_Refline IS
  template  OG_Template;
  refline   OG_Refline;
BEGIN
  template := OG_Get_Template(Temp_Name);
  refline := OG_Get_Refline(Template, 0);
  RETURN(refline);
END;
```

OG_Get_Template

Description This function returns a handle to the specified template.

Syntax

```
FUNCTION OG_Get_Template
  (template_name  VARCHAR2)
RETURN OG_Template;
```

Parameters

template_name	Is the name of the chart template whose handle should be returned.

Returns A handle to the specified chart template. If the template does not exist, this function will return a null handle.

OG_Get_Template Examples

```
 /* Suppose you want to create a chart programmatically.  You would need to assign
attribute values (including a template) to a chart combined attribute record, then pass
that record to OG_Make.
*/

PROCEDURE create_emp_chart IS
   chart_rec      OG_Chart_Ca;
   the_template   OG_Template;
   the_query      OG_Query;
   the_chart      OG_Object;
BEGIN
   the_template:=OG_Get_Template('Emp_Template');
   the_query:=OG_Get_Query('Emp_Query');
   chart_rec.chart_caoc.template:=the_template;
   chart_rec.chart_caoc.query:=the_query;
   chart_rec.chart_caob.mask:=OG_None_Generica;
   chart_rec.chart_caog.mask:=OG_None_Groupa;
   chart_rec.chart_caoc.mask:=OG_Template_Charta+
                              OG_Query_Charta;
   the_chart:=OG_Make(Chart_Rec);
END;
```

OG_Import_Template

Description This procedure imports a chart template.

Syntax

```
FUNCTION OG_Import_Template
  (name           VARCHAR2,
   repository     NUMBER,
   template_name  VARCHAR2)
RETURN OG_Template;
```

Parameters

name	Is the name of the template as it is stored. If the template is stored in the database, this argument should contain only the name of the template. If the template is stored in the file system, this argument should contain the absolute or relative pathname of the template file.
repository	Specifies whether the template is stored in the file system or database. The value of this argument may be one of the following built-in constants:

	OG_Db Means the template is stored in the database. **OG_Filesystem** Means the template is stored in the file system.
template_name	Is the name that Graphics Builder will assign to the template. If another template already has this name, Graphics Builder replaces it with the imported template.

Returns A handle to the imported template.

OG_Import_Template Examples

```
 /* Suppose you want to import the chart template file `my_temp'
**into your display as the template named `template0'.
**The following procedure does this:
*/

PROCEDURE import_the_template IS
   the_temp   OG_Template;
BEGIN
   the_temp:=OG_Import_Template('My_Temp', OG_Filesystem,
'template0');
END;
```

OG_Insert_Ftemp

Description This procedure inserts a new field template into the specified chart template.

Syntax

```
PROCEDURE OG_Insert_Ftemp                         pie/table chart
  (template_hdl  OG_Template,
   attr          OG_Ftemp_Attr,
   indx          NUMBER);
PROCEDURE OG_Insert_Ftemp                         axis chart
  (template_hdl  OG_Template,
   attr          OG_Axisftemp_Ca,
   indx          NUMBER);
```

Parameters

template_hdl	Is the handle to the chart template.
attr	Is the record containing the field template's attributes.
indx	Is the index at which to insert the new field template in the chart template's list of field templates. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of field templates in the chart template prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new field template at the beginning of the chart template's list of field templates (index = 0). **OG_Last** Means insert the new field at the end of the chart template's list of field templates (index = the number of field

templates in the chart template prior to the insertion).

OG_Insert_Ftemp Examples

```
 /* The following procedure inserts a new field
**template into the specified chart template.
*/

PROCEDURE example(template OG_Template) IS
  attr  OG_Axisftemp_Ca;
BEGIN
  attr.ca_aftemp.plottype := OG_Bar_Plottype;
  attr.ca_ftemp.name := 'column';
  attr.ca_aftemp.mask:= OG_Plottype_Axisftempa;
  attr.ca_ftemp.mask := OG_Name_Ftempa;

  OG_Insert_Ftemp(Template, attr, 0);
END;
```

OG_Insert_Refline

Description This procedure inserts a new reference line into the specified chart template.

Syntax

```
PROCEDURE OG_Insert_Refline
  (template_hdl  OG_Template,
   attr          OG_Refline_Attr,
   indx          NUMBER);
```

Parameters

template_hdl	Is the handle to the chart template.
attr	Is the record containing the reference line's attributes.
indx	Is the index at which to insert the new reference line in the chart template's list of reference lines. This argument must be an integer between 0 and *n* (inclusive), where *n* is the number of reference lines in the chart template prior to the insertion. The value of this argument may also be one of the following built-in constants: **OG_First** Means insert the new reference line at the beginning of the chart template's list of reference lines (index = 0). **OG_Last** Means insert the new reference line at the end of the chart template's list of reference lines (index = the number of reference lines in the chart template prior to the insertion).

OG_Insert_Refline Examples

```
 /* The following procedure inserts a new refernce
**line into the specified chart template.
*/

PROCEDURE example(template OG_Template) IS
  attr  OG_Refline_Attr;
BEGIN
  attr.numvalue := 1000;
  attr.label := 'Average';
  attr.mask:= OG_Value_Reflinea+
              OG_Label_Reflinea;

  OG_Insert_Refline(Template, attr, 0);
END;
```

OG_Make_Template

Description This function creates a chart template.

Syntax

```
FUNCTION OG_Make_Template
  (name        VARCHAR2,
   chart_type  NUMBER
RETURN OG_Template;
```

Parameters

name	Is the template name.
chart_type	Is the chart type for the template. This value may be one of the following built-in constants: **OG_Column** **OG_Column_Stacked** **OG_Column_Overlap** **OG_Column_Pct** **OG_Column_Zero** **OG_Column_Shadow** **OG_Column_3d** **OG_Column_Line** **OG_Bar** **OG_Bar_Stacked** **OG_Bar_Overlap** **OG_Bar_Pct** **OG_Bar_Zero** **OG_Bar_Shadow** **OG_Bar_3d** **OG_Bar_Line** **OG_Line** **OG_Line_Symbol** **OG_Line_Stacked** **OG_Line_Fill** **OG_Step** **OG_Step_Symbol** **OG_Step_Stacked** **OG_Step_Fill**

OG_Spline
OG_Spline_Symbol
OG_Spline_Stacked
OG_Spline_Fill
OG_Mixed_Line
OG_Mixed_Fill
OG_Mixed_Spline
OG_Mixed_Spfill
OG_Doubley_Column
OG_Doubley_Overlap
OG_Doubley_Line
OG_Doubley_Symbol
OG_Highlow_Symbol
OG_Highlow_Spike
OG_Highlow_Both
OG_Highlow_Line
OG_Highlow_Fill
OG_Scatter_Symbol
OG_Scatter_Curvefit
OG_Scatter_Linear
OG_Scatter_Log
OG_Scatter_Loglog
OG_Scatter_Connect
OG_Gantt
OG_Gantt_Shadow
OG_Gantt_Depth
OG_Pie
OG_Pie_Shadow
OG_Pie_Depth
OG_Table
OG_Table_Shadow
OG_Table_Depth

Returns A handle to the newly created template.

OG_Make_Template Examples

```
 /* The following function creates a column
**chart template with shadows on the bars:
*/

FUNCTION example RETURN OG_Template IS
  template  OG_Template;
BEGIN
  template := OG_Make_Template('Template0', OG_Column_Shadow);
  RETURN(template);
END;
```

Timer Built-ins

OG_Activate_Timer
OG_Deactivate_Timer
OG_Destroy (Timer)
OG_Get_Timer

OG_Activate_Timer

Description This procedure activates the specified timer.

Syntax

```
PROCEDURE OG_Activate_Timer
  (timer_hdl  OG_Timer);
```

Parameters

timer_hdl	Is the handle to the timer to be activated.

Usage Notes Note that all timers are activated by default when a display is opened. You do not need to activate a timer unless you have explicitly deactivated it by using the OG_Deactivate_Timer procedure.

OG_Activate_Timer Examples

```
/* Suppose you have created timers to update all of your charts
**every 30 seconds, and that you have deactivated the timers
**for charts that are not displayed.  When you display a chart,
**however, you want to re-activate its timer.
*/

PROCEDURE activate_emp_timer IS
   my_timer   OG_Timer;
BEGIN
   my_timer:=OG_Get_Timer('Emp_Timer');
   OG_Activate_Timer(My_Timer);
END;
```

OG_Deactivate_Timer

Description This procedure deactivates the specified timer.

Syntax

```
PROCEDURE OG_Deactivate_Timer
  (timer_hdl  OG_Timer);
```

Parameters

timer_hdl	Is the handle to the timer.

Usage Notes Note that when you open a display, all timers are activated automatically. If you want a timer to be inactive, you must deactivate it explicitly by using this procedure. It is often useful to deactivate timers when they are not required for the portion of the display being viewed. Your system will then not waste time processing irrelevant timers.

OG_Deactivate_Timer Examples

```
 /* Suppose you have created timers to update all of your
**charts every 30 seconds, and that you have deactivated
**the timers for charts that are not displayed.
*/

PROCEDURE deactivate_emp_timer IS
   my_timer   OG_Timer;
BEGIN
   my_timer:=OG_Get_Timer('Emp_Timer');
   OG_Deactivate_Timer(My_Timer);
END;
```

OG_Destroy (Timer)

Description This procedure destroys the specified timer.

Syntax

```
PROCEDURE OG_Destroy
  (timer_hdl  OG_Timer);
```

Parameters

timer_hdl	Is the handle to the timer to destroy.

OG_Destroy (Timer) Examples

```
 /* The following procedure destroys the specified timer:
*/

PROCEDURE destroy_timer(timer_name VARCHAR2) IS
  timer  OG_Timer;
BEGIN
  timer := OG_Get_Timer(Timer_Name);
  OG_Destroy(Timer);
END;
```

OG_Get_Timer

Description This function returns a handle to the specified timer.

Syntax

```
FUNCTION OG_Get_Timer
  (timer_name  VARCHAR2)
RETURN OG_Timer;
```

Parameters

timer_name	Is the name of the timer whose handle is returned.

Returns A handle to the specified timer. If the timer does not exist, this function will return a null handle.

OG_Get_Timer Examples

```
 /* Suppose you have created timers to update all of your
**charts every 30 seconds, and that you have deactivated
**the timers for charts that are not displayed.  When you
**display a chart, however, you want to re-activate its timer.
*/

PROCEDURE activate_emp_timer IS
   my_timer   OG_Timer;
BEGIN
   my_timer:=OG_Get_Timer('Emp_Timer');
   OG_Activate_Timer(My_Timer);
END;
```

OG_Make_Timer

Description This function creates a timer.

Syntax

```
FUNCTION OG_Make_Timer
  (interval   OG_Point,
   timerproc  VARCHAR2
RETURN OG_Timer;
```

Parameters

interval	Is the interval (in seconds) at which the timer is fired.
timerproc	Is the name of the procedure to execute at the timed interval.

Returns A handle to the newly created timer.

OG_Make_Timer Examples

```
 /* The following procedure creates a timer that executes
**the procedure 'update_proc' every 30 seconds.
*/

PROCEDURE example(timer_name VARCHAR2) IS
  timer  OG_Timer;
BEGIN
  timer := OG_Make_Timer(30, 'update_proc');
  OG_Set_Name(Timer, timer_name);
END;
```

TOOLS_INT Built-ins

Tool_Int.Add_Parameter
Tool_Int.Create_Parameter_List
Tool_Int.Delete_Parameter
Tool_Int.Destroy_Parameter_List
Tool_Int.Get_Parameter_Attr
Tool_Int.Get_Parameter_List
Tool_Int.Isnull
Tool_Int.Run_Product
Tool_Int.Set_Parameter_Attr

TOOL_INT.ADD_PARAMETER

Description This procedure adds a parameter to the specified parameter list.

Syntax

```
PROCEDURE TOOL_INT.add_parameter
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   attr        TOOL_INT.PARAM_ATTR);
PROCEDURE TOOL_INT.add_parameter
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   attr        TOOL_INT.PARAM_ATTR);
PROCEDURE TOOL_INT.add_parameter
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   param_type  NUMBER,
   value       CHAR);
```

Parameters

list_hdl	Is the handle to the parameter list.
param_name	Is the name of the parameter to add.
attr	Is the parameter attribute record that contains the type and value of the parameter to add.
param_type	Is the type of the parameter to add. The value of this argument may be one of the following built-in constants: **TOOL_INT.DATA_PARAMETER** Means the parameter represents a mapping of a query

	from one product to the other.
	TOOL_INT.TEXT_PARAMETER Means the parameter is a single value.
value	Is the value of the parameter to add.

Usage Notes You can provide either a parameter attribute record that contains the parameter's type and value, or you can specify the type and value directly in this procedure call.

Tool_Int.Add_Parameter Examples

```
/* The following procedure creates a parameter list and
**adds several parameters to it:
*/

PROCEDURE create_plist IS
  the_list  tool_int.paramlist;
BEGIN
  the_list:=tool_int.create_parameter_list('my_plist');
  tool_int.add_parameter(the_list, 'userid',
    TOOL_INT.TEXT_PARAMETER, 'scott/tiger');
  tool_int.add_parameter(the_list, 'destype',
    TOOL_INT.TEXT_PARAMETER, 'printer');
  tool_int.add_parameter(the_list, 'copies',
    TOOL_INT.TEXT_PARAMETER, '2');
  tool_int.add_parameter(the_list, 'my_param',
    TOOL_INT.TEXT_PARAMETER, '67');
  tool_int.add_parameter(the_list, 'Q_1',
    TOOL_INT.DATA_PARAMETER, 'query0');
END;
```

TOOL_INT.CREATE_PARAMETER_LIST

Description This function creates a new parameter list with the specified name.

Syntax

```
FUNCTION TOOL_INT.create_parameter_list
  (name  CHAR)
RETURN TOOL_INT.PARAMLIST;
```

Parameters

name	Is the name of the parameter list to create.

Returns A handle to the newly created parameter list.

Tool_Int.Create_Parameter_List Examples

```
 /* The following procedure creates a parameter list and
**adds several parameters to it:
*/

PROCEDURE create_plist IS
  the_list  tool_int.paramlist;
BEGIN
  the_list:=tool_int.create_parameter_list('my_plist');
  tool_int.add_parameter(the_list, 'userid',
    TOOL_INT.TEXT_PARAMETER, 'scott/tiger');
  tool_int.add_parameter(the_list, 'destype',
    TOOL_INT.TEXT_PARAMETER, 'printer');
  tool_int.add_parameter(the_list, 'copies',
    TOOL_INT.TEXT_PARAMETER, '2');
  tool_int.add_parameter(the_list, 'my_param',
    TOOL_INT.TEXT_PARAMETER, '67');
  tool_int.add_parameter(the_list, 'Q_1',
    TOOL_INT.DATA_PARAMETER, 'query0');
END;
```

TOOL_INT.DELETE_PARAMETER

Description This procedure deletes the specified parameter from the specified parameter list.

Syntax

```
PROCEDURE TOOL_INT.delete_parameter
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR);
```

Parameters

list_hdl	Is the handle to the parameter list from which to delete the parameter.
param_name	Is the name of the parameter to delete.

Tool_Int.Delete_Parameter Examples

```
 /* The following procedure deletes the parameter 'username' from a parameter list:
*/

PROCEDURE example IS
  list  tool_int.paramlist;
BEGIN
  list := tool_int.get_parameter_list('list0');
  tool_int.delete_parameter(list, 'username');
END;
```

TOOL_INT.DESTROY_PARAMETER_LIST

Description This procedure destroys the specified parameter list.

Syntax

```
PROCEDURE TOOL_INT.destroy_parameter_list
  (list_hdl  TOOL_INT.PARAMLIST);
```

Parameters

list_hdl	Is the handle to the parameter list to destroy.

Tool_Int.Destroy_Parameter_List Examples

```
 /* The following procedure creates a parameter list,
**first destroying an existing list (if any):
*/

PROCEDURE example IS
  list  tool_int.paramlist;
BEGIN
  list := tool_int.get_parameter_list('list0');

  IF NOT tool_int.isnull(list) THEN
    tool_int.destroy_parameter_list(list);
  END IF;

  list := tool_int.create_parameter_list('list0');
END;
```

TOOL_INT.GET_PARAMETER_ATTR

Description This procedure gets the attributes of the specified parameter in the specified parameter list.

Syntax

```
PROCEDURE TOOL_INT.get_parameter_attr
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   attr        TOOL_INT.PARAM_ATTR);
PROCEDURE TOOL_INT.get_parameter_attr
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   param_type  NUMBER,
   value       CHAR);
```

Parameters

list_hdl	Is the handle to the parameter list from which to get the parameter.
param_name	Is the name of the parameter to get.
attr	Is the attribute record to be set to the parameter's attributes.
param_type	Is a variable that this procedure will set to be the parameter's type when it is called. The value of this argument may be one of the following built-in constants: **TOOL_INT.DATA_PARAMETER** Means the parameter represents a mapping of a query from one product to the other. **TOOL_INT.TEXT_PARAMETER** Means the parameter is a single value.
value	Is a variable that this procedure will set to be the parameter's value when it is called.

Usage Notes You can provide either a parameter attribute record that this procedure will set to the parameter's attributes, or you can provide separate variables that will be set.

Tool_Int.Get_Parameter_Attr Examples

```
 /* The following procedure gets the value of the 'priority' parameter,
**and increases its value by one:
```

```
*/

PROCEDURE example IS
  list    tool_int.paramlist;
  ptype   NUMBER;
  pvalue  VARCHAR2;
BEGIN
  list := tool_int.get_parameter_list('list0');
  tool_int.get_parameter_attr(list, 'priority', ptype,
    pvalue);

  pvalue := to_char(to_number(pvalue) + 1);

  tool_int.set_parameter_attr(list, 'priority',
    tool_int.TEXT_PARAMETER, pvalue);

END;
```

TOOL_INT.GET_PARAMETER_LIST

Description This function returns the handle to the parameter list with the specified name.

Syntax

```
FUNCTION TOOL_INT.get_parameter_list
  (list_name  CHAR)
RETURN TOOL_INT.paramlist;
```

Parameters

list_name	Is the name of the parameter list to get.

Returns A handle to the named parameter list.

Tool_Int.Get_Parameter_List Examples

```
 /* The following procedure creates a parameter list,
**first destroying an existing list (if any):
*/

PROCEDURE example IS
  list  tool_int.paramlist;
BEGIN
  list := tool_int.get_parameter_list('list0');

  IF NOT tool_int.isnull(list) THEN
    tool_int.destroy_parameter_list(list);
  END IF;

  list := tool_int.create_parameter_list('list0');
END;
```

TOOL_INT.ISNULL

Description This function determines if the specified parameter list handle is a null handle.

Syntax

```
FUNCTION TOOL_INT.isnull
  (list_hdl  TOOL_INT.PARAMLIST)
RETURN BOOLEAN;
```

Parameters

list_hdl	Is the handle to be evaluated.
Returns TRUE	If the handle is null.
FALSE	If the handle is not null.

Usage Notes TOOL_INT.GET_PARAMETER_LIST returns a null handle if the parameter list it attempts to get does not exist.

Tool_Int.Isnull Examples

```
 /* The following procedure creates a parameter list,
**first destroying an existing list (if any):
*/

PROCEDURE example IS
  list  tool_int.paramlist;
BEGIN
  list := tool_int.get_parameter_list('list0');

  IF NOT tool_int.isnull(list) THEN
    tool_int.destroy_parameter_list(list);
  END IF;

  list := tool_int.create_parameter_list('list0');
END;
```

TOOL_INT.RUN_PRODUCT

Description This procedure invokes another supported Oracle product.

Syntax

```
PROCEDURE TOOL_INT.run_product
  (product     NUMBER,
   module      CHAR,
   comm_mode   NUMBER,
   exec_mode   NUMBER,
   repository  NUMBER,
   list_hdl    TOOL_INT.PARAMLIST);
PROCEDURE TOOL_INT.run_product
  (product     NUMBER,
   module      CHAR,
   comm_mode   NUMBER,
   exec_mode   NUMBER,
   repository  NUMBER,
   list_name   CHAR);
```

Parameters

product	Is the Oracle product that will be invoked. The value of this argument may be one of the following built-in constants: **TOOL_INT.BOOK** Means invoke Oracle Book. **TOOL_INT.FORMS** Means invoke Forms. **TOOL_INT.REPORTS** Means invoke Reports.
module	Is the name of the module to be executed by the invoked product. If the module is stored in the database, this argument should contain only the name of the module . If the module is stored in the file system, this argument should contain the absolute or relative pathname of the module file.
comm_mode	Is the communication mode in which the product will be invoked. The value of this argument may be one of the following built-in

	constants: **TOOL_INT.SYNCHRONOUS** Means the product is invoked synchronously. **TOOL_INT.ASYNCHRONOUS** Means the product is invoked asynchronously.
exec_mode	Is the execution mode for the invoked product. The value of this argument may be one of the following built-in constants: **TOOL_INT.BATCH** Means the product is invoked in batch mode. **TOOL_INT.RUNTIME** Means the product is invoked in runtime mode.
repository	Specifies whether the module is stored in the file system or database. The value of this argument may be one of the following built-in constants: **TOOL_INT.DB** Means the module is stored in the database. **TOOL_INT.FILESYSTEM** Means the module is stored in the filesystem.
list_hdl	Is the handle to the parameter list to be passed to the invoked product.
list_name	Is the name of the parameter list to be passed to the invoked product.

Usage Notes For more information, refer to your Oracle product documentation.

Tool_Int.Run_Product Examples

```
 /* The following procedure opens the Oracle Book document named
**`catalog' and jumps to the hypertext target sailboard:
*/

PROCEDURE call_book is
  list  tool_int.paramlist;
BEGIN
  list:=tool_int.create_parameter_list('plist');

  tool_int.add_parameter(list, 'target',
    tool_int.TEXT_PARAMETER, 'sailboard');

  tool_int.RUN_PRODUCT(tool_int.BOOK, 'catalog',
    tool_int.ASYNCHRONOUS, tool_int.RUNTIME,
    tool_int.FILESYSTEM, list);

END;
```

TOOL_INT.SET_PARAMETER_ATTR

Description This procedure sets the attributes of the specified parameter in the specified parameter list.

Syntax

```
PROCEDURE TOOL_INT.set_parameter_attr
  (list_hdl    TOOL_INT.PARAMLIST,
   param_name  CHAR,
   attr        TOOL_INT.PARAM_ATTR);
PROCEDURE TOOL_INT.set_parameter_attr
```

```
(list_hdl    TOOL_INT.PARAMLIST,
 param_name  CHAR,
 param_type  NUMBER,
 value       CHAR);
```

Parameters

list_hdl	Is the handle to the parameter list that contains the parameter.
param_name	Is the name of the parameter to set.
attr	Is the attribute record that contains the parameter's attributes to set.
param_type	Is the type to which to set the parameter. The value of this argument may be one of the following built-in constants: **TOOL_INT.DATA_PARAMETER** Means the parameter represents a mapping of a query from one product to the other. **TOOL_INT.TEXT_PARAMETER** Means the parameter is a single value.
value	Is the value to which to set the parameter.

Usage Notes You can provide either a parameter attribute record that this procedure will use to set the parameter's attributes, or you can provide separate variables that contain the attributes to be set.

Tool_Int.Set_Parameter_Attr Examples

```
 /* The following procedure gets the value of the 'priority'
**parameter, and increases its value by one:
*/

PROCEDURE example IS
  list    tool_int.paramlist;
  ptype   NUMBER;
  pvalue  VARCHAR2;
BEGIN
  list := tool_int.get_parameter_list('list0');
  tool_int.get_parameter_attr(list, 'priority', ptype,
    pvalue);

  pvalue := to_char(to_number(pvalue) + 1);

  tool_int.set_parameter_attr(list, 'priority',
    tool_int.TEXT_PARAMETER, pvalue);

END;
```

Window Built-ins

OG_Destroy (Window)
OG_Get_Window
OG_Hide_Window
OG_Make_Window
OG_Show_Window

OG_Destroy (Window)

Description This procedure destroys the specified window. Destroying a window closes it, but does not affect its contents.

Syntax

```
PROCEDURE OG_Destroy
  (window_hdl  OG_Window);
```

Parameters

window_hdl	Is the handle to the window to destroy.

OG_Destroy (Window) Examples

```
/* Suppose a user selects a country in a map of the world,
**and the application creates a new window to display
**information about sales in that country.  When the user selects
**another country, you may want to destroy the old window
**and create another one containing information about the new country.
*/

PROCEDURE destroy_USA IS
   the_window   OG_Window;
BEGIN
   the_window:=OG_Get_Window('Usa_Window');
   OG_Destroy(The_Window);
END;
```

OG_Get_Window

Description This function returns a handle to the specified window.

Syntax

```
FUNCTION OG_Get_Window
  (window_name  VARCHAR2)
RETURN OG_Window;
```

Parameters

window_name	Is the name of the window whose handle should be returned.

Returns A handle to the specified window. If the window does not exist, this function will return a null handle.

Usage Notes The window may be the main window for the display (named "Main Layout") or one that has been created programmatically.

OG_Get_Window Examples

```
 /* Suppose the main window-which was previously hidden-contains information
**that is now useful to view.  The following procedure will show it:
*/

PROCEDURE show_main_window IS
   the_main_window   OG_Window;
BEGIN
   the_main_window:=OG_Get_Window('Main Layout');
   OG_Show_Window(The_Main_Window);
END;
```

OG_Hide_Window

Description This procedure hides the specified window.

Syntax

```
PROCEDURE OG_Hide_Window
  (window_hdl  OG_Window);
```

Parameters

window_hdl	Is the handle to the window that should be hidden.

Usage Notes Note that the window will *not* be destroyed. As a result, you can hide the window when its contents are not useful, then show it again when they are.

OG_Hide_Window Examples

```
 /* Suppose the main layout window contains a chart that the user
**no longer needs to see.  The following procedure will hide it temporarily.
**Remember that this does not destroy the window; it will still exist
**and be available to be shown again when needed.
*/

PROCEDURE hide_main_window IS
   the_main_window   OG_Window;
BEGIN
   the_main_window:=OG_Get_Window('Main Layout');
   OG_Hide_Window(The_Main_Window);
END;
```

OG_Make_Window

Description This function creates a window.

Syntax

```
FUNCTION OG_Make_Window
  (position  OG_Point,
   height    NUMBER,
   width     NUMBER)
RETURN OG_Window;
```

Parameters

position	Is the x- and y-coordinates of the window's upper left corner (in screen resolution units).
height	Is the height of the window (in screen resolution units)
width	Is the width of the window (in screen resolution units).

Returns A handle to the newly created window.

OG_Make_Window Examples

```
 /* The following function creates a 5"x4" window
**in the upper left corner of the screen:
*/

FUNCTION example(window_name VARCHAR2) RETURN OG_Window IS
  window  OG_Window;
  pos     OG_Point;
  height  NUMBER;
  width   NUMBER;
BEGIN
  pos.x := 0;
  pos.y := 0;
  width := 5 * OG_Get_Ap_Hscreen_Res;
  height := 4 * OG_Get_Ap_Vscreen_Res;

  window := OG_Make_Window(Pos, height, width);
  OG_Set_Name(Window, window_name);
  RETURN(window);
END;
```

OG_Show_Window

Description This procedure shows the specified window.

Syntax

```
PROCEDURE OG_Show_Window
  (window_hdl  OG_Window);
```

Parameters

window_hdl	Is the handle to the window that should be shown.

OG_Show_Window Examples

```
 /* Suppose the main window-which was previously hidden
**contains information that is now useful to view.
**The following procedure will show it.
*/

PROCEDURE show_main_window IS
   the_main_window   OG_Window;
BEGIN
   the_main_window:=OG_Get_Window('Main Layout');
   OG_Show_Window(The_Main_Window);
END;
```

Properties

Application Properties

Connection String Property
Cursor Property
Horizontal Layout Resolution Property
Horizontal Screen Resolution Property
Password Property
Platform Property
Username Property
Vertical Layout Resolution Property
Vertical Screen Resolution Property

Connection String Property

Description Is the database connection string for the current database connection. If the user is not connected, this property is NULL.

Syntax

```
FUNCTION OG_Get_Ap_Connection
RETURN VARCHAR2;
```

Parameters

None.

Connection String Property Examples

```
/*The following open trigger procedure
**displays the connection string for
**the current database connection
**to a text object.
*/
PROCEDURE connection IS
  connstr     varchar2(10);
BEGIN
  connstr := og_get_ap_connection;
  if connstr = NULL then
    og_set_str(og_get_object('text object'), 'NULL', true, true);
  else
    og_set_str(og_get_object('text object'), connstr, true, true);
  end if;
END;
```

Cursor Property

Description Is the name of the mouse cursor to use. The value of this property may be one of the following strings:

default
insertion
crosshair
help
busy

The appearance of each cursor is system-specific. For more information, refer to your system documentation. If you set this property to an invalid value, it assumes the value `default.'

Syntax

```
PROCEDURE OG_Set_Ap_Cursor
  (cursor  VARCHAR2);
FUNCTION OG_Get_Ap_Cursor
RETURN VARCHAR2;
```

Parameters

cursor	Is the mouse cursor to use.

Cursor Property Examples

```
/*The following procedure changes
**the shape of the cursor depending on
**which layer the user selects.
*/
PROCEDURE ChangeCursor (buttonobj IN og_object,
  hitobj IN og_object, win IN og_window,
  eventinfo IN og_event) IS
  cur        varchar2(10);
BEGIN
  mycur := og_get_ap_cursor;
  if cur = 'default' then
     og_set_ap_cursor('insertion');
  elsif cur = 'insertion' then
     og_set_ap_cursor('crosshair');
  elsif cur = 'crosshair' then
     og_set_ap_cursor('help');
  elsif cur = 'help' then
     og_set_ap_cursor('busy');
  elsif cur = 'busy' then
     og_set_ap_cursor('default');
  end if;
END;
```

Horizontal Layout Resolution Property

Description Is the horizontal resolution of the layout. This value is the number of layout units in one horizontal inch of the layout.

Syntax

```
FUNCTION OG_Get_Ap_Hlayout_Res
RETURN NUMBER;
```

Parameters

None.

Horizontal Layout Resolution Property Examples

```
/*The following procedure displays
**current horizontal layout resolution
**to a text object.
*/
PROCEDURE h_layout IS
  h_layout    number;
BEGIN
  h_layout := og_get_ap_hlayout_res;
  og_set_str(og_get_object('text object'), h_layout, true, true);
END;
```

Horizontal Screen Resolution Property

Description Is the horizontal resolution of the screen. This value is the number of screen resolution units (i.e., pixels) in one horizontal inch of the screen.

Syntax

```
FUNCTION OG_Get_Ap_Hscreen_Res
RETURN NUMBER;
```

Parameters

None.

Horizontal Screen Resolution Property Examples

```
/*The following procedure displays
**current horizontal screen
**to a text object.
*/
PROCEDURE HRES IS
  h_res       number;
BEGIN
  h_res := og_get_ap_vscreen_res;
  og_set_str(og_get_object('text object'), h_res, true, true);
END;
```

Password Property

Description Is the password for the current database connection. If the user is not connected, or the *Keep_Password* preference setting is set to No, this property is NULL.

Syntax

```
FUNCTION OG_Get_Ap_Password
RETURN VARCHAR2;
```

Parameters

None.

Password Property Examples

```
/*The following open trigger procedure
** displays the password for the current
**database connection to a text object.
*/

PROCEDURE password IS
  pw          varchar2(10);
BEGIN
  pw := og_get_ap_password;
  if pw = NULL then
    og_set_str(og_get_object('text object'), 'NULL', true, true);
  else
    og_set_str(og_get_object('text object'), pw, true, true);
  end if;
END;
```

Platform Property

Description Is the platform on which Graphics Builder is running. The value of this property may be one of the following built-in constants:

OG_Macintosh_Platform Means the platform is the Apple Macintosh.

OG_Motif_Platform Means the platform is OSF/MOTIF.

OG_Mswindows_Platform Means the platform is Microsoft Windows.

OG_Pm_Platform Means the platform is Presentation Manager.

OG_X_Platform Means the platform is the X Window System.

Syntax

```
FUNCTION OG_Get_Ap_Platform
RETURN NUMBER;
```

Parameters

None.

Platform Property Examples

```
/*The following procedure displays
**the platform type on which Oracle
**Graphics is currently running to
**a text object.
*/
PROCEDURE platform IS
  ptform     number;
BEGIN
  ptform := og_get_ap_platform;
  if ptform = og_macintosh_platform then
    og_set_str(og_get_object('text object'), 'og_macintosh_platform', true, true);
  elsif ptform = og_motif_platform then
    og_set_str(og_get_object('text object'), 'og_motif_platform', true, true);
  elsif ptform = og_mswindows_platform then
    og_set_str(og_get_object('text object'), 'og_mswindows_platform', true, true);
  elsif ptform = og_pm_platform  then
    og_set_str(og_get_object('text object'), 'og_pm_platform', true, true);
  elsif ptform = og_x_platform then
    og_set_str(og_get_object('text object'), 'og_x_platform', true, true);
  end if;
END;
```

Username Property

Description Is the username for the current database connection. If the user is not connected, this property is NULL.

Syntax

```
FUNCTION OG_Get_Ap_Username
RETURN VARCHAR2;
```

Parameters

None.

Username Property Examples

```
/*The following open trigger procedure
**displays the username for the current
**database connection to a text object.
*/

PROCEDURE username IS
  usr        varchar2(10);
BEGIN
  usr := og_get_ap_username;
  if usr = NULL then
    og_set_str(og_get_object('text object'), 'NULL', true, true);
  else
    og_set_str(og_get_object('text object'), usr, true, true);
  end if;
END;
```

Vertical Layout Resolution Property

Description Is the vertical resolution of the layout. This value is the number of layout units in one vertical inch of the layout.

Syntax

```
FUNCTION OG_Get_Ap_Vlayout_Res
RETURN NUMBER;
```

Parameters

None.

Vertical Layout Resolution Property Examples

```
/*The following procedure displays current
**vertical layout resolution to a text object.
*/
PROCEDURE v_layout IS
  v_layout    number;
BEGIN
  v_layout := og_get_ap_vlayout_res;
  og_set_str(og_get_object('text object'), v_layout, true, true);
END;
```

Vertical Screen Resolution Property

Description Is the vertical resolution of the screen. This value is the number of screen resolution units (i.e., pixels) in one vertical inch of the screen.

Syntax

```
FUNCTION OG_Get_Ap_Vscreen_Res
RETURN NUMBER;
```

Parameters

None.

Vertical Screen Resolution Property Examples

```
/*The following procedure displays current
**vertical screen resolution to a text object:
/*
PROCEDURE VRES IS
  v_res      number;
BEGIN
  v_res := og_get_ap_vscreen_res;
  og_set_str(og_get_object('text object'), v_res, true, true);
END;
```

Arc Properties

Base Arc Property
Closure Property
Fill Property

Base Arc Property

Description Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the ellipse from which the arc is cut.

Syntax

```
PROCEDURE OG_Set_Basearc
  (arc         OG_Object,
   basearc     OG_Arc,
   damage      BOOLEAN    :=  TRUE,
   update_bbox BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Basearc
  (arc  OG_Object)
RETURN OG_Arc;
```

Parameters

arc	Is the arc object being described.
basearc	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the ellipse from which the arc is cut.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Base Arc Property Examples

```
/*The following procedure reads
**information from an existing arc,
**reduces all data by half, and
**updates the arc object.
*/

PROCEDURE base_arc IS
  arc        og_arc;
BEGIN
  arc := og_get_basearc(og_get_object('arc'));
  arc.x := arc.x/2;
  arc.y := arc.y/2;
  arc.height :=arc.height/2;
  arc.width := arc.width/20;
  arc.sangle := arc.sangle/2;
  arc.eangle := arc.eangle/2;
  og_set_basearc(og_get_object('arc'), arc);
END;
```

Closure Property

Description Is the closure of the arc. The value of this property may be one of the following:

TRUE Means the arc is closed.

FALSE Means the arc is open.

Syntax

```
PROCEDURE OG_Set_Arc_Closed
  (arc         OG_Object,
   closed      BOOLEAN,
   damage      BOOLEAN    :=  TRUE,
   update_bbox BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Arc_Closed
  (arc  OG_Object)
RETURN BOOLEAN;
```

Parameters

arc	Is the arc object being described.
closed	Is the closure of the arc.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Closure Property Examples

```
/*The following procedure reads the
**closure of an existing arc.  If closure
**equals TRUE, it fills the arc with red
**and sets the closure value to FALSE;
**if closure equals FALSE, it fills the
**arc with blue and sets the value to TRUE.
*/
PROCEDURE closure IS
  cls        BOOLEAN;
  arc        og_object;
BEGIN
  arc := og_get_object('arc');
  cls := og_get_arc_closed(arc);
  if cls = TRUE  then
     og_set_bfcolor(arc, 'red');
     og_set_arc_closed(arc, FALSE);
  else
     og_set_bfcolor(arc, 'blue');
     og_set_arc_closed(arc, TRUE);
  end if;
END;
```

Fill Property

Description Is the fill shape of the arc. The value of this property may be one of the following built-in constants:

OG_Chord_Arcfill Means the fill shape of the arc is that of a chord.

OG_Pie_Arcfill Means the fill shape of the arc is that of a full pie slice.

Syntax

```
PROCEDURE OG_Set_Arcfill
  (arc          OG_Object,
   arcfill      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Arcfill
  (arc  OG_Object)
RETURN NUMBER;
```

Parameters

arc	Is the arc object being described.
arcfill	Is the fill shape of the arc.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Fill Property Examples

```
/*The following procedure reads the
**arcfill from an arc, prints the value to a
**text object, assigns a different value
** to the arcfill value.
*/
PROCEDURE fill IS
  text       og_object;
  arc        og_object;
  num        NUMBER;
BEGIN
  text := og_get_object('text object');
  arc := og_get_object('arc');
  num := og_get_arcfill(arc);
  og_set_str(text, num);
  og_set_arcfill(arc, og_chord_arcfill);
END;
```

Axis (Date) Properties

Auto Maximum Property
Auto Minimum Property
Auto Step Property
Custom Format Property
Day Format Property
First Month Property
Labels Property
Maximum Property
Minimum Property
Month Format Property
Quarter Format Property
Skip Weekends Property
Step Property
Year Format Property

Auto Maximum Property

Description Specifies whether the axis maximum is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Date_Automax
  (axis     OG_Axis,
   automax  BOOLEAN,
   maximun  DATE);
FUNCTION OG_Get_Date_Automax
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
automax	Specifies whether the axis maximum is set to *Auto*.

maximun	Specifies the maximum axis value (if *automax* is FALSE).

Auto Maximum Property Examples

```
/*The following procedure checks if axis
**Y1's date maximum is set to auto. If
**the return value is TRUE, it resets the
**value to FALSE with default_max;
**if the return value is FALSE, it resets
**the value to TRUE after reading the
**specified maximum axis value.
/*
PROCEDURE datemax IS
  template    og_template;
  axis        og_axis;
  val1        date;
  val2        boolean;
  default_max date := '06-dec-99';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val2 := og_get_date_automax(axis);
  if val2 = true then
    og_set_date_automax(axis, false, default_max);
    val1 := og_get_date_maximum(axis);
  elsif val2 = false then
   og_set_date_automax(axis, true, default_max);
  end if;
END;
```

Auto Minimum Property

Description Specifies whether the axis minimum is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Date_Automin
  (axis     OG_Axis,
   automin  BOOLEAN,
   minimun  DATE);
FUNCTION OG_Get_Date_Automin
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
automin	Specifies whether the axis minimum is set to *Auto*.
minimun	Specifies the minimum axis value (if *automin* is FALSE).

Auto Minimum Property Examples

```
/*The following procedure checks if axis
**Y1's date minimum is set to auto. If the
**return value is TRUE, it resets the value
**to FALSE with default_min; if the return
**value is FALSE, it resets the value to
**TRUE after reading the specified minimum
**axis value.
*/
PROCEDURE datemin IS
  template    og_template;
  axis        og_axis;
  val1        date;
  val2        boolean;
 default_min  date := '01-dec-79';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val2 := og_get_date_automin(axis);
  if val2 = true then
    og_set_date_automin(axis, false, default_min);
    val1 := og_get_date_minimum(axis);
  elsif val2 = false then
    og_set_date_automin(axis, true, default_min);
  end if;
END;
```

Auto Step Property

Description Specifies whether the axis step value is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Date_Autostep
  (axis      OG_Axis,
   autostep  BOOLEAN,
   step      NUMBER);
FUNCTION OG_Get_Date_Autostep
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
autostep	Specifies whether the axis step value is set to *Auto*.
step	Specifies the axis step value (if *autostep* is FALSE).

Auto Step Property Examples

```
/*The following procedure checks if axis
**Y1's date step is set to auto. If the
**return value is TRUE, it resets the value
**to FALSE with default_step; if the return
**value is FALSE, it resets the value
**to TRUE after reading the specified step
**value.
*/
PROCEDURE datestep IS
  template    og_template;
  axis        og_axis;
  val         boolean;
  num         number;
  default_step number := og_second_step;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val:= og_get_date_autostep(axis);
  if val = true then
    og_set_date_autostep(axis, false, default_step);
    num := og_get_date_step(axis);
  elsif val = false then
    og_set_date_autostep(axis, true, default_step);
  end if;
END;
```

Custom Format Property

Description Is the custom date format for the axis tick labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Custfmt
  (axis     OG_Axis,
   custfmt  VARCHAR2);
FUNCTION OG_Get_Custfmt
  (axis  OG_Axis)
RETURN VARCHAR2;
```

Parameters

axis	Is the axis object being described.
custfmt	Is the custom date format for the axis tick labels.

Custom Format Property Examples

```
/*The following procedure reads the
**Custom format value and compares it
**with the variable 'default_format';
**if the two value are not equal,
**it resets the current format to the
**value of the 'default_format'.
*/
PROCEDURE customfmt IS
  template    og_template;
  axis        og_axis;
  val         varchar2(10);
  default_format varchar2(10) := 'DD_YY_MM';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_custfmt(axis);
  if val != default_format then
    og_set_custfmt(axis, default_format);
  end if;
END;
```

Day Format Property

Description Determines the appearance of day-of-the-week labels along the axis. The value of this property may be one of the following built-in constants:

OG_Firstletter_Fmt

OG_Threeletter_Fmt

Syntax

```
PROCEDURE OG_Set_Dayfmt
  (axis    OG_Axis,
   dayfmt  NUMBER);
FUNCTION OG_Get_Dayfmt
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
dayfmt	Determines the appearance of day-of-the-week labels along the axis.

Day Format Property Examples

```
/*The following procedure checks the
**day-of-week format.  If the current format
**is First-Letter format, it resets the value
**to Three-Letter format, and vice versa.
*/
PROCEDURE dayfmt IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  num:= og_get_dayfmt(axis);
  if num = og_firstletter_fmt then
    og_set_dayfmt(axis, og_threeletter_fmt);
  elsif num = og_threeletter_fmt then
    og_set_dayfmt(axis, og_firstletter_fmt);
  end if;
END;
```

First Month Property

Description Is the month that is considered to begin a new year. The value of this property may be one of the following built-in constants:

OG_Jan_Month
OG_Feb_Month
OG_Mar_Month
OG_Apr_Month
OG_May_Month
OG_Jun_Month
OG_Jul_Month
OG_Aug_Month
OG_Sep_Month
OG_Oct_Month
OG_Nov_Month
OG_Dec_Month

Syntax

```
PROCEDURE OG_Set_Firstmon
  (axis      OG_Axis,
   firstmon  NUMBER);
FUNCTION OG_Get_Firstmon
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
firstmon	Is the month that is considered to begin a new year.

First Month Property Examples

```
/*The following reads the first month
**value and resets the value to the next
**acceptable value.
*/
PROCEDURE firstmonth IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  num:= og_get_firstmon(axis);
  if num = og_jan_month then
    og_set_firstmon(axis, og_feb_month);
  elsif num = og_feb_month then
    og_set_firstmon(axis, og_mar_month);
  elsif num = og_mar_month then
    og_set_firstmon(axis, og_apr_month);
  elsif num = og_apr_month then
    og_set_firstmon(axis, og_may_month);
  elsif num = og_may_month then
    og_set_firstmon(axis, og_jun_month);
  elsif num = og_jun_month then
    og_set_firstmon(axis, og_jul_month);
  elsif num = og_jul_month then
    og_set_firstmon(axis, og_aug_month);
  elsif num = og_aug_month then
    og_set_firstmon(axis, og_sep_month);
  elsif num = og_sep_month then
    og_set_firstmon(axis, og_oct_month);
  elsif num = og_oct_month then
    og_set_firstmon(axis, og_nov_month);
  elsif num = og_nov_month then
    og_set_firstmon(axis, og_dec_month);
  else og_set_firstmon(axis, og_jan_month);
  end if;
END;
```

Labels Property

Description Specifies the major interval along the axis at which major tick marks and tick labels appear, as well as the appearance of the tick labels. The value of this property may be one of the following built-in constants:
OG_No_Labels
OG_Second_Labels
OG_Minute_Labels
OG_Hour_Labels
OG_Ampm_Labels
OG_Day_Labels
OG_Dayofweek_Labels
OG_Week_Labels
OG_Month_Labels
OG_Quarter_Labels
OG_Year_Labels
OG_Custom_Labels (If *labels* is set to this value, you must specify the custom date format in the *Custom Format* property.)
Syntax

```
PROCEDURE OG_Set_Labels
  (axis    OG_Axis,
   labels  NUMBER);
FUNCTION OG_Get_Labels
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
labels	Specifies the major interval along the axis at which major tick marks and tick labels appear, as well as the appearance of the tick labels.

Labels Property Examples

```
/*The following procedure determines
**if any label boxes are checked.
**If checked label boxes are found,
**it unchecks all labels; if no checked
**labels are found, it checks the 'Year'
**check box.
*/
PROCEDURE labels IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);

  num:= og_get_labels(axis);
  if num != og_no_labels then
    og_set_labels(axis, og_no_labels);
  else og_set_labels(axis, og_year_labels);
  end if;
END;
```

Maximum Property

Description Specifies the maximum axis value (if *Auto Maximum* is FALSE).

Syntax

(See OG_Set_Date_Automax, above.)

```
FUNCTION OG_Get_Date_Maximum
  (axis  OG_Axis)
RETURN DATE;
```

Parameters

axis	Is the axis object being described.

Maximum Property Examples

```
/*The following procedure checks if
**axis Y1's date maximum is set to auto.
**If the return value is TRUE,
**it resets the value to FALSE with
**default_max; if the return value is
**FALSE, it resets the value to
**TRUE after reading the specified
**maximum axis value.
/*
PROCEDURE datemax IS
  template    og_template;
  axis        og_axis;
  val1        date;
  val2        boolean;
  default_max date := '06-dec-99';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val2 := og_get_date_automax(axis);
  if val2 = true then
    og_set_date_automax(axis, false, default_max);
    val1 := og_get_date_maximum(axis);
  elsif val2 = false then
   og_set_date_automax(axis, true, default_max);
  end if;
END;
```

Minimum Property

Description Specifies the minimum axis value (if *Auto Minimum* is FALSE).

Syntax

(See OG_Set_Date_Automin.)

```
FUNCTION OG_Get_Date_Minimum
  (axis  OG_Axis)
RETURN DATE;
```

Parameters

axis	Is the axis object being described.

Minimum Property Examples

```
/*The following procedure checks if
**axis Y1's date minimum is set to auto.
**If the return value is TRUE, it resets
**the value to FALSE with default_min;
**if the return value is FALSE, it resets
**the value to TRUE after reading the
**specified minimum axis value.
*/
PROCEDURE datemin IS
  template    og_template;
  axis        og_axis;
  val1        date;
  val2        boolean;
 default_min  date := '01-dec-79';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val2 := og_get_date_automin(axis);
  if val2 = true then
    og_set_date_automin(axis, false, default_min);
    val1 := og_get_date_minimum(axis);
  elsif val2 = false then
    og_set_date_automin(axis, true, default_min);
  end if;
END;
```

Month Format Property

Description Determines the appearance of month labels along the axis. The value of this property may be one of the following built-in constants:

OG_Firstletter_Fmt

OG_Threeletter_Fmt

Syntax

```
PROCEDURE OG_Set_Monthfmt
  (axis      OG_Axis,
   monthfmt  NUMBER);
FUNCTION OG_Get_Monthfmt
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
monthfmt	Determines the appearance of month labels along the axis.

Month Format Property Examples

```
/*The following procedure checks the
**Month format. If the current format
**is First-Letter format, it
**resets the value to Three-Letter
**format, and vice versa.
*/
PROCEDURE monthfmt IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  num:= og_get_monthfmt(axis);
  if num = og_firstletter_fmt then
    og_set_monthfmt(axis, og_threeletter_fmt);
  elsif num = og_threeletter_fmt then
    og_set_monthfmt(axis, og_firstletter_fmt);
  end if;
END;
```

Quarter Format Property

Description Determines the appearance of quarter labels along the axis. The value of this property may be one of the following built-in constants:

OG_Arabic_Fmt

OG_Roman_Fmt

Syntax

```
PROCEDURE OG_Set_Qtrfmt
  (axis    OG_Axis,
   qtrfmt  NUMBER);
FUNCTION OG_Get_Qtrfmt
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
qtrfmt	Determines the appearance of quarter labels along the axis.

Quarter Format Property Examples

```
/*The following procedure checks the
**Quarter format. If the current
**format is Arabic format, it resets
**the value to Roman format, and vice versa.
*/
PROCEDURE qtrfmt IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  num:= og_get_qtrfmt(axis);
  if num = og_arabic_fmt then
    og_set_qtrfmt(axis, og_roman_fmt);
  elsif num = og_roman_fmt then
    og_set_qtrfmt(axis, og_arabic_fmt);
  end if;
END;
```

Skip Weekends Property

Description Specifies whether weekends are ignored when calculating axis values.

Syntax

```
PROCEDURE OG_Set_Skipwknds
  (axis       OG_Axis,
   skipwknds  BOOLEAN);
FUNCTION OG_Get_Skipwknds
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
skipwknds	Specifies whether weekends are ignored when calculating axis values.

Skip Weekends Property Examples

```
/*The following procedure checks whether
**weekends are ignored when calculating
**axis values. If the value of weekend
**is set to TRUE (ignored), the procedure
**resets the value to FALSE (include
**in the calculation) and vice versa.
*/
PROCEDURE skipwknds IS
  template    og_template;
  axis        og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val:= og_get_skipwknds(axis);
  if val = true then
    og_set_skipwknds(axis, false);
  elsif val = false then
    og_set_skipwknds(axis, true);
  end if;
END;
```

Step Property

Description Specifies the axis step value (if *Auto Step* is FALSE). The value of this property may be one of the following built-in constants:

OG_Second_Step
OG_Minute_Step
OG_Hour_Step
OG_Day_Step
OG_Week_Step
OG_Month_Step
OG_Quarter_Step
OG_Year_Step

Syntax

(See OG_Set_Date_Autostep.)

```
FUNCTION OG_Get_Date_Step
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.

Step Property Examples

```
/*The following procedure checks if
**axis Y1's date step is set to auto.
**If the return value is TRUE, it resets
**the value to FALSE with default_step;
**if the return value is FALSE,
**it resets the value to TRUE
**after reading the specified step
**value.
*/

PROCEDURE datestep IS
  template    og_template;
  axis        og_axis;
  val         boolean;
  num         number;
  default_step number := og_second_step;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val:= og_get_date_autostep(axis);
  if val = true then
    og_set_date_autostep(axis, false, default_step);
    num := og_get_date_step(axis);
  elsif val = false then
    og_set_date_autostep(axis, true, default_step);
  end if;
END
```

Year Format Property

Description Determines the appearance of year labels along the axis. The value of this property may be one of the following built-in constants:

OG_Fourdigit_Fmt

OG_Twodigit_Fmt

Syntax

```
PROCEDURE OG_Set_Yearfmt
  (axis     OG_Axis,
   yearfmt  NUMBER);
FUNCTION OG_Get_Yearfmt
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
yearfmt	Determines the appearance of year labels along the axis.

Year Format Property Examples

```
/*The following procedure checks the Year
**format.  If the current format is Two-Digit
**format, it resets the value to
**Four-Digit format, and vice versa.
*/
PROCEDURE yearfmt IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  num:= og_get_yearfmt(axis);
  if num = og_fourdigit_fmt then
    og_set_yearfmt(axis, og_twodigit_fmt);
  elsif num = og_twodigit_fmt then
    og_set_yearfmt(axis, og_fourdigit_fmt);
  end if;
END;
```

Axis (Generic) Properties

Axis Label Property
Axis Type Property
Custom Label Property
Direction Property
Major Grid Property
Major Ticks Property
Minor Grid Property
Minor Ticks Property
Minor Ticks Per Interval Property
Position Property
Tick Label Rotation Property
Tick Labels Property
Tick Position Property

Axis Label Property

Description Specifies whether labels that identify values along the axis appear.

Syntax

```
PROCEDURE OG_Set_Axislabel
  (axis       OG_Axis,
   axislabel  BOOLEAN);
FUNCTION OG_Get_Axislabel
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
axislabel	Specifies whether labels that identify values along the axis appear.

Axis Label Property Examples

```
/*The following procedure determines if
**the Axis Label checkbox is checked.
**If the box is checked, it unchecks
**it, and vice versa.
*/
PROCEDURE GenAxisLbl IS
  template    og_template;
  x_axis      og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  val := og_get_axislabel(x_axis);
  if val = true then
     og_set_axislabel(x_axis, false);
  else
     og_set_axislabel(x_axis, true);
  end if;
END;
```

Axis Type Property

Description Specifies the type of axis to use. The value of this property may be one of the following built-in constants:

OG_Continuous_Axistype

OG_Date_Axistype

OG_Discrete_Axistype

Syntax

```
PROCEDURE OG_Set_Axistype
  (axis      OG_Axis,
   axistype  NUMBER);
FUNCTION OG_Get_Axistype
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
axistype	Specifies the type of axis to use.

Axis Type Property Examples

```
/*The following procedure reads the
**current axis type.  If the current type
**is CONTINUOUS, it resets the type
**to DISCRETE, or vice versa.  If the
**current type is DATE, it changes the
**year format.
*/
PROCEDURE GenAxisType IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  num := og_get_axistype(axis);
  if num = og_discrete_axistype then
     og_set_axistype(axis,og_continuous_axistype);
  elsif num = og_continuous_axistype then
     og_set_axistype(axis, og_discrete_axistype);
  elsif num = og_date_axistype then
     og_set_yearfmt(axis, og_twodigit_fmt);
  end if;
END;
```

Custom Label Property

Description Specifies the text of the label that appears along the axis.

Syntax

```
PROCEDURE OG_Set_Custlabel
  (axis        OG_Axis,
   custlabel  VARCHAR2);
FUNCTION OG_Get_Custlabel
  (axis  OG_Axis)
RETURN VARCHAR2;
```

Parameters

axis	Is the axis object being described.
custlabel	Specifies the text of the label that appears along the axis.

Custom Label Property Examples

```
/*The following procedure reads the current
**label of the specific axis, and changes
**the name of that label.
*/
PROCEDURE CustLabel IS
  template    og_template;
  axis        og_axis;
  label       varchar2(20);
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  label := og_get_custlabel(axis);
  og_set_custlabel(axis, 'Employee Number');
END;
```

Direction Property

Description Specifies in which direction increasing values, or successive categories, are placed along the axis. The value of this property may be one of the following built-in constants:

OG_Down_Direction
OG_Left_Direction
OG_Right_Direction
OG_Up_Direction

Syntax

```
PROCEDURE OG_Set_Direction
  (axis       OG_Axis,
   direction  NUMBER);
FUNCTION OG_Get_Direction
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
direction	Specifies in which direction increasing values, or successive categories, are placed along the axis.

Direction Property Examples

```
/*The following procedure reads the
**directions of the x and y axis and sets
**them to the opposite directions.
*/
PROCEDURE GenDirection IS
  template    og_template;
  x_axis      og_axis;
  y_axis      og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);
  y_axis := og_get_axis(template, og_y1_axis);
  num := og_get_direction(x_axis);
  if num = og_left_direction then
     og_set_direction(x_axis, og_right_direction);
  elsif num = og_right_direction then
     og_set_direction(x_axis, og_left_direction);
  end if;
  num := og_get_direction(y_axis);
  if num = og_up_direction then
     og_set_direction(y_axis, og_down_direction);
  elsif num = og_down_direction then
     og_set_direction(y_axis, og_up_direction);
  end if;
END;
```

Major Grid Property

Description Specifies whether a grid line appears at each major tick mark.

Syntax

```
PROCEDURE OG_Set_Majorgrid
  (axis       OG_Axis,
   majorgrid  BOOLEAN);
FUNCTION OG_Get_Majorgrid
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
majorgrid	Specifies whether a grid line appears at each major tick mark.

Major Grid Property Examples

```
/*The following procedure checks if the
**Major Grid checkbox is checked.  If the
**box is checked, it unchecks it, and vice
**versa.
*/
PROCEDURE GenMajorGrids IS
  template    og_template;
  x_axis      og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  val := og_get_majorgrid(x_axis);
  if val = true then
     og_set_majorgrid(x_axis, false);
  else
     og_set_majorgrid(x_axis, true);
  end if;
END;
```

Major Ticks Property

Description Specifies whether major tick marks appear at each major interval.

Syntax

```
PROCEDURE OG_Set_Majorticks
  (axis        OG_Axis,
   majorticks  BOOLEAN);
FUNCTION OG_Get_Majorticks
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
majorticks	Specifies whether major tick marks appear at each major interval.

Major Ticks Property Examples

```
/*The following procedure checks if the
**Major Ticks checkbox is checked.  If
**the box is checked, it unchecks it,
**and vice versa.
*/
PROCEDURE GenMajorTicks IS
  template    og_template;
  x_axis      og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  val := og_get_majorticks(x_axis);
  if val = true then
    og_set_majorticks(x_axis, false);
  else
     og_set_majorticks(x_axis, true);
  end if;
END;
```

Minor Grid Property

Description Specifies whether a grid line appears at each minor tick mark.

Syntax

```
PROCEDURE OG_Set_Minorgrid
  (axis       OG_Axis,
   minorgrid  BOOLEAN);
FUNCTION OG_Get_Minorgrid
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
minorgrid	Specifies whether a grid line appears at each minor tick mark

Minor Grid Property Examples

```
/*The following procedure checks if
**the Minor Grid checkbox is checked.
**If the box is checked, it unchecks it,
**and vice versa.
*/
PROCEDURE GenMinorGrids IS
  template    og_template;
  x_axis      og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  val := og_get_minorgrid(x_axis);
  if val = true then
     og_set_minorgrid(x_axis, false);
  else
     og_set_minorgrid(x_axis, true);
  end if;
END;
```

Minor Ticks Property

Description Specifies whether minor tick marks appear, as specified by the value set for Minor Ticks per Interval.

Syntax

```
PROCEDURE OG_Set_Minorticks
  (axis        OG_Axis,
   minorticks  BOOLEAN);
FUNCTION OG_Get_Minorticks
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
minorticks	Specifies whether minor tick marks appear as set by the value of Minor Ticks per Interval.

Minor Ticks Property Examples

```
/*The following procedure checks if the
**Minor Ticks checkbox is checked.  If
**the box is checked, it unchecks it, and
**vice versa.
*/
PROCEDURE GenMinorTicks IS
  template    og_template;
  x_axis      og_axis;
  val         boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  val := og_get_minorticks(x_axis);
  if val = true then
     og_set_minorticks(x_axis, false);
  else
     og_set_minorticks(x_axis, true);
  end if;
END;
```

Minor Ticks Per Interval Property

Description Is the number of minor ticks defined within each major tick interval.

Syntax

```
PROCEDURE OG_Set_Minorct
  (axis     OG_Axis,
   minorct  NUMBER);
FUNCTION OG_Get_Minorct
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
minorct	Is the number of minor ticks defined within each major tick interval.

Minor Ticks Per Interval Property Examples

```
/*The following procedure reads the number
**of minor ticks per interval and resets
**the value to triple the original value.
*/
PROCEDURE GenMinorCt IS
  template    og_template;
  x_axis      og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);
  num := og_get_ticklabelrot(x_axis);
  og_set_minorct(x_axis, 3*num);
END;
```

Position Property

Description Specifies along which edge of the chart the axis appears. The value of this property may be one of the following built-in constants:

OG_Bottom_Position
OG_Left_Position
OG_Right_Position
OG_Top_Position

Syntax

```
PROCEDURE OG_Set_Position
  (axis      OG_Axis,
   position  NUMBER);
FUNCTION OG_Get_Position
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
position	Specifies along which edge of the chart the axis appears.

Position Property Examples

```
/*The following procedure determines
**which edge of the chart the axis
**appears on, and resets the axis to
**the opposite edge.
*/
PROCEDURE GenPosition IS
  template    og_template;
  axis        og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  num := og_get_position(axis);
  if num = og_bottom_position then
     og_set_position(axis, og_top_position);
  elsif num = og_left_position then
     og_set_position(axis, og_right_position);
  elsif num = og_right_position then
     og_set_position(axis, og_left_position);
  elsif num = og_top_position then
     og_set_position(axis, og_bottom_position);
  end if;
END;
```

Tick Label Rotation Property

Description Specifies the direction in which the tick labels are rotated. The value of this property may be one of the following built-in constants:

OG_Ccw_Rotation Means counter-clockwise rotation.

OG_Cw_Rotation Means clockwise rotation.

OG_No_Rotation Means no rotation.

Syntax

```
PROCEDURE OG_Set_Ticklabelrot
  (axis          OG_Axis,
   ticklabelrot  NUMBER);
FUNCTION OG_Get_Ticklabelrot
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
ticklabelrot	Specifies the direction in which the tick labels are rotated.

Tick Label Rotation Property Examples

```
/*The following procedure reads the
**tick label rotation and changes it
**to a different value.
*/
PROCEDURE GenTickLbl IS
  template    og_template;
  x_axis      og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);

  num := og_get_ticklabelrot(x_axis);
  if num = og_ccw_rotation then
     og_set_ticklabelrot(x_axis, og_cw_rotation);
  elsif num = og_cw_rotation then
     og_set_ticklabelrot(x_axis, og_no_rotation);
  elsif num = og_no_rotation then
     og_set_ticklabelrot(x_axis, og_ccw_rotation);
  end if;
END;
```

Tick Labels Property

Description Specifies whether labels that identify values along the axis appear.

Syntax

```
PROCEDURE OG_Set_Ticklabels
  (axis        OG_Axis,
   ticklabels  BOOLEAN);
FUNCTION OG_Get_Ticklabels
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
ticklabels	Specifies whether labels that identify values along the axis appear.

Tick Labels Property Examples

```
/*The following procedure checks if
**Tick Label checkbox is checked.
**If the box is checked, it unchecks it,
**and vice versa.
*/
PROCEDURE GenTickLbl IS
  template   og_template;
  x_axis     og_axis;
  val        boolean;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);
  val := og_get_ticklabels(x_axis);
  if val = true then
     og_set_ticklabels(x_axis, false);
  else
     og_set_ticklabels(x_axis, true);
  end if;
END;
```

Tick Position Property

Description Specifies how the major and minor tick marks appear. The value of this property may be one of the following built-in constants:

OG_Cross_Tickpos
OG_Inside_Tickpos
OG_Outside_Tickpos

Syntax

```
PROCEDURE OG_Set_Tickpos
  (axis     OG_Axis,
   tickpos  NUMBER);
FUNCTION OG_Get_Tickpos
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.
tickpos	Specifies how the major and minor tick marks appear.

Tick Position Property Examples

```
/*The following procedure reads the tick
**position of the x-axis, and sets it to
**a different value.
*/
PROCEDURE GenTickPos IS
  template    og_template;
  x_axis      og_axis;
  num         number;
BEGIN
  template := og_get_template('template0');
  x_axis := og_get_axis(template, og_x_axis);
  num := og_get_tickpos(x_axis);
  if num = og_cross_tickpos then
     og_set_tickpos(x_axis, og_inside_tickpos);
  elsif num = og_inside_tickpos then
     og_set_tickpos(x_axis, og_outside_tickpos);
  elsif num = og_outside_tickpos then
     og_set_tickpos(x_axis, og_cross_tickpos);
  end if;
END;
```

Axis (Discrete) Properties

Auto Maximum Property

Description Specifies whether the maximum number of categories that appear on the axis is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Disc_Automax
  (axis     OG_Axis,
   automax  BOOLEAN,
   maxcat   NUMBER);
FUNCTION OG_Get_Disc_Automx
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
automax	Specifies whether the maximum number of categories that appear on the axis is set to *Auto*.
maxcat	Specifies the maximum number of categories that appear on the axis (if *automax* is FALSE).

Auto Maximum Property Examples

```
/*The following procedure checks if the
**X-axis's maximum is set to auto.  If
**true, it resets the value to false with
**default_maxcat; if false, it reads the
**current value and resets it to true.
*/
PROCEDURE datemax IS
  template   og_template;
  axis       og_axis;
  val        boolean;
  maxcat     number;
  default_maxcat     number := 3;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_automax(axis);
  if val = true then
     og_set_disc_automax(axis, false, default_maxcat);
  elsif val = false then
    maxcat := og_get_disc_maxcat(axis);
    og_set_disc_automax(axis,true,default_maxcat);
  end if;
END;
```

Auto Minimum Property

Description Specifies whether the minimum number of categories that appear on the axis is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Disc_Automin
  (axis     OG_Axis,
   automin  BOOLEAN,
   mincat   NUMBER);
FUNCTION OG_Get_Disc_Automin
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	Is the axis object being described.
automin	Specifies whether the minimum number of categories that appear on the axis is set to *Auto*.
mincat	Specifies the minimum number of categories that appear on the axis (if *automin* is FALSE.)

Auto Minimum Property Examples

```
/*The following procedure checks if the
**X-axis's minimum is set to auto.  If
**true, it resets the value to false with
*default_mincat;  if false, it reads the
**current value and resets the value to
**true.
*/
PROCEDURE datemin IS
  template    og_template;
  axis        og_axis;
  val         boolean;
  mincat      number;
  default_mincat     number := 50;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_automin(axis);
  if val = true then
    og_set_disc_automin(axis,false,default_mincat);
  elsif val = false then
    mincat := og_get_disc_mincat(axis);
    og_set_disc_automin(axis,true,default_mincat);
  end if;
END;
```

Date Format Property

Description Specifies the date format for the axis tick labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Disc_Datefmt
  (axis      OG_Axis,
   date_fmt  VARCHAR2);
FUNCTION OG_Get_Disc_Datefmt
  (axis  OG_Axis)
RETURN VARCHAR2;
```

Parameters

axis	Is the axis object being described.
date_fmt	Specifies the date format for the axis tick labels.

Date Format Property Examples

```
/*The following procedure reads the current
**date format of the axis.  If the current
**format is not equal to variable
**'default_date', it resets the value to
**'default_date.'
*/
PROCEDURE datefmt IS
  template    og_template;
  axis        og_axis;
  val varchar2(10);
  default_date       varchar2(10) := 'DD_YY_MM';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_datefmt(axis);
  if val != default_date then
     og_set_disc_datefmt(axis, default_date);
  end if;
END;
```

Maximum Number of Categories Property

Description Specifies the maximum number of categories that appear on the axis (if *automax* is FALSE).

Syntax

(See OG_Set_Disc_Automax, above.)

```
FUNCTION OG_Get_Disc_Maxcat
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.

Maximum Number of Categories Property Examples

```
/*
** The following procedure checks if the
**X-axis's maximum is set to auto.  If
**true, it resets the value to false with
**default_maxcat; if false, it reads the
**current value and resets
**it to true.
*/
PROCEDURE datemax IS
  template    og_template;
  axis        og_axis;
  val         boolean;
  maxcat      number;
  default_maxcat     number := 3;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_automax(axis);
  if val = true then
     og_set_disc_automax(axis, false, default_maxcat);
  elsif val = false then
    maxcat := og_get_disc_maxcat(axis);
    og_set_disc_automax(axis,true,default_maxcat);
  end if;
END;
```

Minimum Number of Categories Property

Description Specifies the minimum number of categories that appear on the axis (if *automin* is FALSE).

Syntax

(See OG_Set_Disc_Automin, above.)

```
FUNCTION OG_Get_Disc_Mincat
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	Is the axis object being described.

Minimum Number of Categories Property Examples

```
/*The following procedure checks if the
**X-axis's minimum is set to auto.  If
**true, it resets the value to false with
**default_mincat;  if false, it reads the
**current value and resets the value to
**true.
*/
PROCEDURE datemin IS
  template    og_template;
  axis        og_axis;
  val         boolean;
  mincat      number;
  default_mincat      number := 50;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_automin(axis);
  if val = true then
    og_set_disc_automin(axis,false,default_mincat);
  elsif val = false then
    mincat := og_get_disc_mincat(axis);
    og_set_disc_automin(axis,true,default_mincat);
  end if;
END;
```

Number Format Property

Description Specifies the number format for the axis tick labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Disc_Numfmt
  (axis     OG_Axis,
   num_fmt  VARCHAR2);
FUNCTION OG_Get_Disc_Numfmt
  (axis  OG_Axis)
RETURN VARCHAR2;
```

Parameters

axis	Is the axis object being described.
num_fmt	Specifies the number format for the axis tick labels.

Number Format Property Examples

```
/*The following procedure reads the current
**number format of the axis.  If the current
**format is not equal to variable
**'default_format', it resets the value to
**'default_format'
*/
PROCEDURE discnumfmt IS
  template    og_template;
  axis        og_axis;
  val         varchar2(10);
  default_format     varchar2(10) := '9,9,9,9';
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_x_axis);
  val := og_get_disc_numfmt(axis);
  if val != default_format then
     og_set_disc_numfmt(axis, default_format);
  end if;
END;
```

Axis (Continuous) Properties

Auto Maximum Property

Description Specifies whether the axis maximum is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Cont_Automax
  (axis     OG_Axis,
   automax  BOOLEAN,
   maximun  NUMBER);
FUNCTION OG_Get_Cont_Automax
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	The axis object being described.
automax	Specifies whether the axis maximum is set to *Auto*.
maximun	Specifies the maximum axis value (if *automax* is FALSE).

Auto Maximum Property Examples

```
/*The following procedure checks if axis
**Y1's maximum is set to auto. If return
**value is TRUE, reset the value to FALSE
**with default_max; if return value is
**FALSE, it resets the value to TRUE
**after reading the specified maximum
**axis value.
*/
PROCEDURE automin IS
  axis        og_axis;
  template    og_template;
  val         boolean;
  num         number;
  default_max        number := 3000;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_autostep(axis);
  if val = TRUE then
    og_set_cont_autostep(axis, FALSE, default_max);
  else
    num := og_get_cont_step(axis);
    og_set_cont_autostep(axis, TRUE, default_max);
  end if;
END;
```

Auto Minimum Property

Description Specifies whether the axis minimum is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Cont_Automin
  (axis     OG_Axis,
   automin  BOOLEAN,
   minimun  NUMBER);
FUNCTION OG_Get_Cont_Automin
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	The axis object being described.
automin	Specifies whether the axis minimum is set to *Auto*.
minimun	Specifies the minimum axis value (if *automin* is FALSE).

Auto Minimum Property Examples

```
/*The following procedure checks if axis
**Y1's minimum is set to auto. If the
**value is TRUE, it resets the value to
**FALSE with default_min; if the return
**value is FALSE, it resets the value to
**TRUE after reading the specified minimum
**axis value.
*/
PROCEDURE automin IS
  axis        og_axis;
  template    og_template;
  val         boolean;
  num         number;
  default_min        number := 500;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_automin(axis);
  if val = TRUE then
    og_set_cont_automin(axis, FALSE, default_min);
  elsif val = FALSE then
    num := og_get_cont_minimum(axis);
    og_set_cont_automin(axis, TRUE, default_min);
  end if;
END;
```

Auto Step Property

Description Specifies whether the axis step value is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Cont_Autostep
  (axis      OG_Axis,
   autostep  BOOLEAN,
   step      NUMBER);
FUNCTION OG_Get_Cont_Autostep
  (axis  OG_Axis)
RETURN BOOLEAN;
```

Parameters

axis	The axis object being described.
autostep	Specifies whether the axis step value is set to *Auto*.
step	Specifies the axis step value (if *autostep* is FALSE).

Auto Step Property Examples

```
/*The following procedure checks if axis
**Y1's step is set to auto. If the return
**value is TRUE, it resets the value to
**FALSE with default step value; if
**return value is FALSE, it resets
**the value to TRUE after reading
**the specified step value.
*/

PROCEDURE autostep IS
  axis        og_axis;
  template    og_template;
  val         boolean;
  num         number;
  step        number := 500;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_autostep(axis);
  if val = TRUE then
    og_set_cont_autostep(axis, FALSE, step);
  else
    num := og_get_cont_step(axis);
    og_set_cont_autostep(axis, TRUE, step);
  end if;
END;
```

Maximum Property

Description Specifies the maximum axis value (if *Auto Maximum* is FALSE).

Syntax

(See OG_Set_Cont_Automax, above.)

```
FUNCTION OG_Get_Cont_Maximum
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.

Maximum Property Examples

```
/*The following procedure checks if axis
**Y1's maximum is set to auto. If return
**value is TRUE, reset the value to
**FALSE with default_max; if return value
**is FALSE, it resets the value to
**TRUE after reading the specified
**maximum axis value.
*/
PROCEDURE automin IS
  axis        og_axis;
  template    og_template;
  val         boolean;
  num         number;
  default_max        number := 3000;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_autostep(axis);
  if val = TRUE then
    og_set_cont_autostep(axis, FALSE, default_max);
  else
    num := og_get_cont_step(axis);
    og_set_cont_autostep(axis, TRUE, default_max);
  end if;
END;
```

Minimum Property

Description Specifies the minimum axis value (if *Auto Minimum* is FALSE).

Syntax

(See OG_Set_Cont_Automin, above.)

```
FUNCTION OG_Get_Cont_Minimum
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.

Minimum Property Examples

```
/*The following procedure checks if axis
**Y1's minimum is set to auto. If the
**return value is TRUE, it resets the
**value to FALSE with default_min;
**if the return value is FALSE, it resets
**the value to TRUE after reading the
**specified minimum axis value.
*/
PROCEDURE automin IS
  axis        og_axis;
  template    og_template;
  val         boolean;
  num         number;
  default_min        number := 500;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_automin(axis);
  if val = TRUE then
    og_set_cont_automin(axis, FALSE, default_min);
  elsif val = FALSE then
    num := og_get_cont_minimum(axis);
    og_set_cont_automin(axis, TRUE, default_min);
  end if;
END;
```

Number Format Property

Description Specifies the number format for the axis tick labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Cont_Numfmt
  (axis     OG_Axis,
   num_fmt  VARCHAR2);
FUNCTION OG_Get_Cont_Numfmt
  (axis  OG_Axis)
RETURN VARCHAR2;
```

Parameters

axis	The axis object being described.
num_fmt	Specifies the number format for the axis tick labels.

Number Format Property Examples

```
/*The following procedure reads the current
**number format of the axis and resets it to
**a different value.
*/
PROCEDURE numFormat IS
  axis       og_axis;
  template   og_template;
  val        varchar2(10);
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_numfmt(axis);
  og_set_cont_numfmt(axis,'9,9,9,9,9');
END;
```

Percent by Property

Description Specifies how the *Percent Of* scaling values are calculated. The value of this property may be one of the following built-in constants:

OG_Category_Pctby Means the percentage for each data value is calculated relative to data values for the same field in other categories.

OG_Field_Pctby Means the percentage for each data value is calculated relative to data values in the same category for other fields.

Syntax

```
PROCEDURE OG_Set_Pct_By
  (axis    OG_Axis,
   pct_of  NUMBER);
FUNCTION OG_Get_Pct_By
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.
pct_of	Specifies how the *Percent Of* scaling values are calculated.

Percent by Property Examples

```
*/The following procedure reads the
**calculating method for the
**Percent Of scaling values
**(with Scale is set for OG_PCT_SCALE)
**from the axis and resets the value to
**the next available value.
/*
PROCEDURE pctby IS
  axis       og_axis;
  template   og_template;
  val        number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_pct_by(axis);
  if val = OG_category_pctby then
    og_set_pct_by(axis, og_field_pctby);
  elsif val = og_field_pctby then
    og_set_pct_by(axis, og_category_pctby);
  end if;
END;
```

Percent of Property

Description Specifies the relative scaling factor (if *Scale* is set to OG_Pct_Scale). The value of this property may be one of the following built-in constants:

OG_Maximum_Pctof Meanseach data value is plotted as a percentage of the largest data value.

OG_Minimum_Pctof Means each data value is plotted as a percentage of the smallest data value.

OG_Sum_Pctof Means each data value is plotted as a percentage of the sum of all data values.

Syntax

```
PROCEDURE OG_Set_Pct_Of
  (axis    OG_Axis,
   pct_of  NUMBER);
FUNCTION OG_Get_Pct_Of
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.
pct_of	Specifies the relative scaling factor (if *Scale* is set to OG_Pct_Scale).

Percent of Property Examples

```
/*The following procedure reads the
**relative scaling factor (with Scale
**set to OG_PCT_SCALE)from the axis
**and resets the value to the next
**available value.
*/
PROCEDURE pctof  IS
  axis       og_axis;
  template   og_template;
  val        number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_pct_of(axis);
  if val = OG_maximum_pctof then
     og_set_pct_of(axis, og_minimum_pctof);
  elsif val = og_minimum_pctof then
    og_set_pct_of(axis, og_sum_pctof);
  elsif val = og_sum_pctof then
    og_set_pct_of(axis, og_maximum_pctof);
  end if;
END;
```

Scale Property

Description Specifies the algorithm used for scaling the axis. The value of this property may be one of the following built-in constants:

OG_Linear_Scale Means the axis is scaled using a fixed interval between the minimum and maximum axis values.

OG_LOG_Scale Means the axis is scaled using a logarithmic algorithm (based on powers of 10) to determine the intervals between the minimum and maximum axis values.

OG_Pct_Scale Means the axis is scaled so that data values will be plotted relative to the amount specified by *Percent Of*.

Syntax

```
PROCEDURE OG_Set_Scale
  (axis   OG_Axis,
   scale  NUMBER);
FUNCTION OG_Get_Scale
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.
scale	Specifies the algorithm used for scaling the axis.

Scale Property Examples

```
/*The following procedure reads
**the method used for scaling from
**the axis and resets the value
**to the next available value.
*/

PROCEDURE  scale IS
  axis       og_axis;
  template   og_template;
  val        number;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_scale(axis);
  if val = OG_linear_scale then
    og_set_scale(axis, og_log_scale);
  elsif val = og_log_scale then
    og_set_scale(axis, og_pct_scale);
  elsif val = og_pct_scale then
    og_set_scale(axis, og_linear_scale);
  end if;
END;
```

Step Property

Description Specifies the axis step value (if *Auto Step* is FALSE).

Syntax

(See OG_Set_Cont_Autostep, above.)

```
FUNCTION OG_Get_Cont_Step
  (axis  OG_Axis)
RETURN NUMBER;
```

Parameters

axis	The axis object being described.

Step Property Examples

```
/*The following procedure checks if axis
**Y1's step is set to auto. If the return
**value is TRUE, it resets the value to
**FALSE with default step value; if return
**value is FALSE, it resets the value to
**TRUE after reading the specified step value.
*/

PROCEDURE autostep IS
  axis       og_axis;
  template   og_template;
  val        boolean;
  num        number;
  step       number := 500;
BEGIN
  template := og_get_template('template0');
  axis := og_get_axis(template, og_y1_axis);
  val := og_get_cont_autostep(axis);
  if val = TRUE then
    og_set_cont_autostep(axis, FALSE, step);
  else
    num := og_get_cont_step(axis);
    og_set_cont_autostep(axis, TRUE, step);
  end if;
END;
```

Chart Element Properties

Button Procedure Property
Events Property
Explosion Property
Name Property

Button Procedure Property

Description Is the handle to the button procedure that should be associated with this chart element. Note that the *Events* property must be set properly in order to ensure that this procedure receives the desired mouse events. The Events property may be one of the following built-in constants:
OG_No_Events
OG_Mouse_Up
OG_Mouse_Down
OG_Mouse_Move_Down
To enable the procedure to receive multiple event types, set Events to be the sum of the constants for the desired events.
Syntax
```
PROCEDURE OG_Set_Button
  (chart        OG_Object,
   row_num      NUMBER,
   col_name     VARCHAR2,
   button_proc  OG_Buttonproc,
   events       NUMBER);
```
Parameters

chart	Is the chart object being described.
row_num	Is the query row number represented by the chart element.
col_name	Is the query column represented by the chart element.
button_proc	Is the handle to the button procedure that should be associated with this chart element.
events	Is the type of mouse events that the button procedure should receive.

Button Procedure Property Examples

```
/*The following procedure assigns
**a button procedure to chart
**element MGR_bars.
*/
PROCEDURE  AssignButtonProc IS
  chart og_object;
  mgrbar og_object;
  button og_buttonproc;
BEGIN
  chart := og_get_object('chart');
  mgrbar := og_get_object('MGR_bars');
  button := og_get_buttonproc('button');
  og_set_button(chart, og_get_row(mgrbar), 'MGR', button, og_mouse_down);
END;
```

Events Property

Description Is the type of mouse events that the button procedure should receive. The value of this property may be one of the built-in constants:
OG_No_Events
OG_Mouse_Up
OG_Mouse_Down
OG_Mouse_Move_Down
To enable the procedure to receive multiple event types, set Events to be the sum of the constants for the desired events.
Syntax
(See OG_Set_Button.)
Parameters
None

Events Property Examples

```
/*The following procedure assigns
**a button procedure to
**chart element MGR_bars.
*/
PROCEDURE  AssignButtonProc IS
  chart og_object;
  mgrbar og_object;
  button og_buttonproc;
BEGIN
  chart := og_get_object('chart');
  mgrbar := og_get_object('MGR_bars');
  button := og_get_buttonproc('button');
  og_set_button(chart, og_get_row(mgrbar), 'MGR', button, og_mouse_down);
END;
```

Explosion Property

Description Is the distance that the chart element (i.e., pie slice) should be exploded, in terms of the percentage of the chart's x- and y-radii (e.g., 25). This property is meaningful only when used with a pie chart. In addition, all of the pie slices for a given category will be exploded the same amount. Therefore, the specified column name should be for a value column, not a category column.

Syntax

```
 PROCEDURE OG_Set_Explosion
   (chart        OG_Object,
    row_num      NUMBER,
    col_name     VARCHAR2,
    explode_pct  NUMBER);
```

Parameters

chart	Is the chart object being described.
row_num	Is the query row number represented by the chart element.
col_name	Is the query column represented by the chart element. It should be the name of a value column
explode_pct	Is the distance that the chart element (i.e., pie slice) should be exploded, in terms of the percentage of the chart's x- and y-radii (e.g., 25).

Explosion Property Examples

```
/*The following procedure assigns the
**distance the chart element should be
** exploded to to 50.
*/
PROCEDURE Explosion IS
  pie og_object;
  mgr_slice og_object;
BEGIN
  pie := og_get_object('pie');
  mgr_slice := og_get_object('MGR_slices');
  og_set_explosion(pie, og_get_row(mgr_slice), 'MGR', 50);
END;
```

Name Property

Description Is the name of the chart element.

Syntax

```
PROCEDURE OG_Set_Name
  (chart     OG_Object,
   row_num   NUMBER,
   col_name  VARCHAR2,
   name      VARCHAR2);
```

Parameters

chart	Is the chart object being described.
row_num	Is the query row number represented by the chart element.
col_name	Is the query column represented by the chart element.
name	Is the name of the chart element.

Name Property Examples

```
/*The following procedure sets
**the name of the chart element.
*/
PROCEDURE Name IS
  chart og_object;
  mgr_bar og_object;
BEGIN
  chart := og_get_object('chart');
  mgr_bar := og_get_object('Mgr_bars');
  og_set_name(chart, og_get_row(mgr_bar), 'MGR', 'NewName');
END;
```

Chart Properties

Auto Update Property
End Row Property
Filter Property
Query Property
Range Property
Size And Position Property
Start Row Property
Template Property
Title Property

Auto Update Property

Description Specifies that the chart is automatically be updated when the query is executed.

Syntax

```
PROCEDURE OG_Set_Autoupdate
  (chart       OG_Object,
   autoupdate  BOOLEAN);
FUNCTION OG_Get_Autoupdate
  (chart  OG_Object)
RETURN BOOLEAN;
```

Parameters

chart	Is the chart being described.
autoupdate	Specifies that the chart is automatically be updated when the query is executed.

Auto Update Property Examples

```
/*The following reads the value of
**autoupdate in Chart properties, and
**resets the value to its opposite value
*/
PROCEDURE ChartAutoUpdate IS
  chart og_object;
  autoupdate boolean;
BEGIN
  chart := og_get_object('chart');
  autoupdate := og_get_autoupdate(chart);
  if autoupdate = true then
     og_set_autoupdate(chart, false);
  else
    og_set_autoupdate(chart, true);
   end if;
END;
```

End Row Property

Description Is the last row from the query that appears on the chart.

Syntax

(See OG_Set_Rows.)

```
FUNCTION OG_Get_Endrow
  (chart  OG_Object)
RETURN NUMBER;
```

Parameters

chart	Is the chart object being described.

End Row Property Examples

```
/*The following procedure reads the
**startrow and endrow value from chart
**(provided the Plot rows box is checked),
**and resets the range to startrow -1 and
**endrow -1.)
*/
PROCEDURE ChartStartEnd IS
  chart og_object;
 startrow number;
  endrow number;
BEGIN
  chart := og_get_object('chart');
  startrow := og_get_startrow(chart);
  endrow := og_get_endrow(chart);
  og_set_rows(chart,true, startrow-1, endrow-1);
END;
```

Filter Property

Description Is the name of the query's filter trigger procedure.

Syntax

```
PROCEDURE OG_Set_Filter
  (chart   OG_Object,
   filter  VARCHAR2);
FUNCTION OG_Get_Filter
  (chart  OG_Object)
RETURN VARCHAR2;
```

Parameters

chart	Is the chart object being described.
filter	Is the name of the query's filter trigger procedure.

Filter Property Examples

```
/*The following procedure reads
**the name of the current filter trigger
**of the chart, and assigns a different
**filter trigger to the chart
*/
PROCEDURE ChartFilter IS
  chart og_object;
  current_filter varchar2(30);
  new_filter varchar2(30):='MyFilter';
BEGIN
  chart := og_get_object('chart');
  current_filter := og_get_filter(chart);
  og_set_filter(chart, new_filter);
END;
```

Query Property

Description Is the handle to the query to be used for the chart.

Syntax

```
PROCEDURE OG_Set_Query
  (chart        OG_Object,
   query        OG_Query,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Query
  (chart  OG_Object)
RETURN OG_Query;
```

Parameters

chart	Is the chart object being described.
query	Is the handle to the query to be used for the chart.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Query Property Examples

```
/*The following procedure reads the
**query handle from the current chart
**(qry0)and resets the handle value to
**qry1.
*/
PROCEDURE ChartQuery IS
  chart og_object;
  qry0 og_query;
  qry1 og_query;
BEGIN
  chart := og_get_object('chart');
  qry0 := og_get_query(chart);
  qry1 := og_get_query('query1');
  og_set_query(chart, qry1);
END;
```

Range Property

Description Specifies whether the number of query rows that appear on the chart is restricted to the range specified by *startrow* and *endrow.*

Syntax

```
PROCEDURE OG_Set_Rows
  (chart      OG_Object,
   rangeflag  BOOLEAN,
   startrow   NUMBER,
   endrow     NUMBER);
FUNCTION OG_Get_Rangeflag
  (chart  OG_Object)
RETURN BOOLEAN;
```

Parameters

chart	Is the chart object being described.
rangeflag	Specifies whether the number of query rows that appear on the chart is restricted to the range specified by *startrow* and *endrow.*
startrow	Is the first row from the query that appears on the chart. The first query row is 0, the second row is 1, and so on.
endrow	Is the last row from the query that appears on the chart.

Range Property Examples

```
/*The following procedure checks if
**the number of query rows that appear
**on the chart is range restricted.
**If true, it resets the value to false
**(i.e. plots all rows); if false, it
**resets the value to true with a
**restricted range specified by
**startrow and endrow.
*/
PROCEDURE ChartRange IS
  chart og_object;
  rangeflag boolean;
  startrow number := 3;
  endrow number := 9;
BEGIN
  chart := og_get_object('chart');
  rangeflag := og_get_rangeflag(chart);
  if rangeflag = true then
     og_set_rows(chart,false, startrow, endrow);
  else
    og_set_rows(chart, true, startrow, endrow);
  end if;
END;
```

Size and Position Property

Description Is the x- and y-coordinates, height, and width of the chart's frame (in layout units).

Syntax

```
PROCEDURE OG_Set_Frame
  (chart        OG_Object,
   frame        OG_Rectangle,
   damage       BOOLEAN       :=  TRUE,
   update_bbox  BOOLEAN       :=  TRUE);
FUNCTION OG_Get_Frame
  (chart  OG_Object)
RETURN OG_Rectangle;
```

Parameters

chart	Is the chart object being described.
frame	Is the x- and y-coordinates, height, and width of the chart's frame (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Size and Position Property Examples

```
/*The following procedure reads the frame
**size of the chart, and reduces it by half.
*/
PROCEDURE SizeAndPos IS
  chart og_object;
  rect og_rectangle;
BEGIN
  chart := og_get_object('chart');
    rect := og_get_frame(chart);
  rect.x := rect.x/2;
  rect.y := rect.y/2;
  rect.height := rect.height/2;
  rect.width := rect.width/2;
  og_set_frame(chart, rect);
END;
```

Start Row Property

Description Is the first row from the query that appears on the chart. The first query row is 0, the second row is 1, and so on.

Syntax

(See OG_Set_Rows, above.)

```
FUNCTION OG_Get_Startrow
  (chart  OG_Object)
RETURN NUMBER;
```

Parameters

chart	Is the chart object being described.

Start Row Property Examples

```
/*The following procedure reads the
**startrow and endrow value from chart
**(provided the Plot rows box is checked),
**and resets the range to startrow -1 and
**endrow -1.)
*/
PROCEDURE ChartStartEnd IS
  chart og_object;
 startrow number;
  endrow number;
BEGIN
  chart := og_get_object('chart');
  startrow := og_get_startrow(chart);
  endrow := og_get_endrow(chart);
  og_set_rows(chart,true, startrow-1, endrow-1);
END;
```

Template Property

Description Is the handle to the template to be used for the chart.

Syntax

```
PROCEDURE OG_Set_Template
  (chart       OG_Object,
   template    OG_Template,
   damage      BOOLEAN      :=  TRUE,
   update_bbox BOOLEAN      :=  TRUE);
FUNCTION OG_Get_Template
  (chart  OG_Object)
RETURN OG_Template;
```

Parameters

chart	Is the chart object being described.
template	Is the handle to the template to be used for the chart.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Template Property Examples

```
/*The following procedure reads the
**template handles template1 and template2
**from chart1 and chart2 respectively, and
**assigns template1 to chart2, template2
**to chart1.
*/
PROCEDURE ChartTemplate IS
  chart1 og_object;
  chart2 og_object;
  template1 og_template;
  template2 og_template;
BEGIN
  chart1 := og_get_object('chart1');
  chart2 := og_get_object('chart2');
  template1 := og_get_template(chart1);
  template2 := og_get_template(chart2);
  og_set_template(chart1, template2);
  og_set_template(chart2, template1);
END;
```

Title Property

Description Is the title of the chart.

Syntax

```
PROCEDURE OG_Set_Title
  (chart  OG_Object,
   title  VARCHAR2);
FUNCTION OG_Get_Title
  (chart  OG_Object)
RETURN VARCHAR2;
```

Parameters

chart	Is the chart object being described.
title	Is the title of the chart.

Title Property Examples

```
/*The following procedure reads
**the title of a chart; compare
**the value with new_title. If
**they are not equal, change the
**title to new_title.
*/
PROCEDURE ChartTitle IS
  chart og_object;
  title varchar2(30);
  new_title varchar2(30) := 'New title';
BEGIN
  chart := og_get_object('chart');
  title := og_get_title(chart);
  if title != new_title then
    og_set_title(chart, new_title);
  end if;
END;
```

Compound Text Properties

Simple Text Count Property
Compound Text Count Property

Simple Text Count Property

Description Is the number of simple text elements that compose the compound text element.

Syntax

```
FUNCTION OG_Get_Stcount
  (text          OG_Object,
   cmptext_index  NUMBER)
RETURN NUMBER;
```

Parameters

text	Is the text element being described.
cmptext_index	Is the index number of the compound text element being described.

Simple Text Count Property Examples

```
*/The following procedure reads the count of
**simple text of the first compound
**text in a text object, and prints the count
**back to the text object.
*/
 PROCEDURE simpleText IS
  num number;
  text og_object;
BEGIN
  text := og_get_object('text');
  num := og_get_stcount(text,0);
  og_set_str(text, num);
END;
```

Display Properties

Close Trigger Property
Date Format Property
Height Property
Open Trigger Property
Width Property

Close Trigger Property

Description Is the name of display's Close Display trigger.

Syntax

```
PROCEDURE OG_Set_Closetrigger
  (trigger  VARCHAR2);
FUNCTION OG_Get_Closetrigger
RETURN VARCHAR2;
```

Parameters

trigger	Is the name of display's Close Display trigger.

Close Trigger Property Examples

```
/*The following procedure reads the name
**of the close trigger of the current
**display.  If the current trigger is not
**new_trigger, it sets new_trigger to be the
**current close trigger procedure.
*/
PROCEDURE CloseTrigger IS
  val varchar2(20);
  new_trigger varchar2(20) := 'CURSORDEFAULT';
BEGIN
   val := og_get_closetrigger;
   if val != new_trigger then
     og_set_closetrigger('CursorDefault');
   end if;
END;
```

Date Format Property

Description Specifies the date format for parameters. This must be a valid SQL format string. For more information, see your*Oracle7 Server SQL Reference*.

Syntax

```
OG_Set_Dateformat
  (dateformat  VARCHAR2);
OG_Get_Dateformat
RETURN VARCHAR2;
```

Parameters

dateformat	Specifies the date format for parameters. This must be a valid SQL format string.

Date Format Property Examples

```
/*The following procedure reads the date
**format of display.  If the format is not
**the same as new_datefmt, it sets the current
**format to new_format.
*/
PROCEDURE datefmt IS
  datefmt varchar2(20);
  new_datefmt varchar2(20) := 'DD/MM/YYYY';
BEGIN
  datefmt := og_get_dateformat;
  if datefmt != new_datefmt then
    og_set_dateformat('DD/MM/YYYY');
  end if;
END;
```

Height Property

Description Is the height of the layout (in layout units).

Syntax

(See OG_Set_Display_Size.)

```
FUNCTION OG_Get_Display_Height
RETURN NUMBER;
```

Parameters

None

Height Property Examples

```
/*The following procedure reads the width
**and height of the current display and
**reduces the display size by half.
*/

PROCEDURE dimension0 IS
  width number;
  height number;
BEGIN
  width := og_get_display_width;
  height := og_get_display_height;
  og_set_display_size(width/2, height/2);
END;
```

Open Trigger Property

Description Is the name of display's Open Display trigger.

Syntax

```
PROCEDURE OG_Set_Opentrigger
  (trigger  VARCHAR2);
FUNCTION OG_Get_Opentrigger
RETURN VARCHAR2;
```

Parameters

trigger	Is the name of display's Open Display trigger.

Open Trigger Property Examples

```
/*The following procedure reads the name of
**the open trigger of the current display.
**If the current trigger is not new_trigger,
**it sets new_trigger to be the current open
**trigger procedure.
*/

PROCEDURE OpenTrigger IS
  val varchar2(20);
  new_trigger varchar2(20) := 'TOBLUE';
BEGIN
  val := og_get_opentrigger;
  if val != 'TOBLUE' then
    og_set_opentrigger('toblue');
  end if;
END;
```

Width Property

Description Is the width of the layout (in layout units).

Syntax

```
PROCEDURE OG_Set_Display_Size
  (width   NUMBER,
   height  NUMBER);
FUNCTION OG_Get_Display_Width
RETURN NUMBER;
```

Parameters

width	Is the width of the layout (in layout units).
height	Is the height of the layout (in layout units).

Width Property Examples

```
/*The following procedure reads the width
**and height of the current display and
**reduces the display size by half.
*/

PROCEDURE dimension0 IS
  width number;
  height number;
BEGIN
  width := og_get_display_width;
  height := og_get_display_height;
  og_set_display_size(width/2, height/2);
END;
```

Frame (Axis Chart) Properties

Baseline Axis Property
Baseline Value Property
Category Width Property
Custom Date Format Property
Custom Number Format Property
Reference Line Count Property
Second Y Axis Property

Baseline Axis Property

Description Specifies the axis to which the baseline value is compared to determine its position. The value of this property may be one of the following built-in constants:
OG_Template
OG_Y1_Axis
OG_Y2_Axis
Syntax

```
PROCEDURE OG_Set_Baseaxis
  (template  OG_Template,
   baseaxis  NUMBER);
FUNCTION OG_Get_Baseaxis
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
baseaxis	Specifies the axis to which the baseline value is compared to determine its position.

Baseline Axis Property Examples

```
*/The following procedure specifies the
**date format for the baseline label.
*/
PROCEDURE CusDateFmt IS
  chart og_object;
  template og_template;
  custDate date;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  custDate := og_get_cust_date(template);
  if custDate != '06-DEC-88' then
    og_set_cust_date(template, '06-DEC-96');
  end if;
END;
```

Baseline Value Property

Description Is the value used as the starting point for plotting fields along the value axis. The value of this property may be one of the following built-in constants:

OG_Custom_Baseline

OG_Min_Baseline

OG_Zero_Baseline

Syntax

```
PROCEDURE OG_Set_Basevalue
  (template   OG_Template,
   basevalue  NUMBER);
FUNCTION OG_Get_Basevalue
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
basevalue	Is the value used as the starting point for plotting fields along the value axis.

Baseline Value Property Examples

```
/*The following procedure reads
**the baseline value of the field
**template of a chart.If the current
** baseline value is ZERO,
**the procedure resets the value to
**MAX; If the current baseline value
**is any value other than ZERO, the
**procedure resets the value to ZERO.
*/
PROCEDURE BaseLine IS
  chart og_object;
  template og_template;
  value number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  value := og_get_basevalue(template);
  if value = og_zero_baseline then
    og_set_basevalue(template, og_max_baseline);
  else
    og_set_basevalue(template, og_zero_baseline);
  end if;
  og_update_chart(chart);
END;
```

Category Width Property

Description Is the width of the bars in a bar or column chart, as a percentage of the "strip width." The strip width is the widest the bars can be without overlapping each other, and it is determined by dividing the length of the category axis by the number of bars to be plotted.

Syntax

```
PROCEDURE OG_Set_Catwidth
  (template  OG_Template,
   catwidth  NUMBER);
FUNCTION OG_Get_Catwidth
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
catwidth	Is the width of the bars in a bar or column chart, as a percentage of the "strip width." The strip width is the widest the bars can be without overlapping each other, and it is determined by dividing the length of the category axis by the number of bars to be plotted.

Category Width Property Examples

```
/* The following procedure reduces the
** category width of the bars by half of
**its original width.
*/
PROCEDURE CatWidth IS
  chart og_object;
  template og_template;
  width number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  width := og_get_catwidth(template);
  og_set_catwidth(template, width/2);
END;
```

Custom Date Format Property

Description Specifies the custom date to set the custom date value to. This will also automatically set the base value to OG_CUSTOM_BASELINE**.**

Syntax

```
PROCEDURE OG_Set_Cust_Date
  (template   OG_Template,
   cust_date  DATE);
FUNCTION OG_Get_Cust_Date
  (template  OG_Template)
RETURN DATE;
```

Parameters

template	Is the chart template.
cust_date	Specifies the date value for a date axis type. This value is used as a reference for drawing the data points along the value axis.

Custom Date Format Property Examples

```
/*The following procedure specifies
**the date format for the baseline label.
*/
PROCEDURE CusDateFmt IS
  chart og_object;
  template og_template;
  custDate date;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  custDate := og_get_cust_date(template);
  if custDate != '06-DEC-88' then
    og_set_cust_date(template, '06-DEC-96');
  end if;
END;
```

Custom Number Format Property

Description Specifies the custom number to set the baseline to. This will also automatically set the base value to OG_CUSTOM_BASELINE.

Syntax

```
PROCEDURE OG_Set_Cust_Num
  (template  OG_Template,
   cust_num  NUMBER);
FUNCTION OG_Get_Cust_Num
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
cust_num	Specifies the baseline value for a number axis type. This value is used as a reference for drawing the data points along the value axis.

Custom Number Format Property Examples

```
/*The following procedure specifies
**the number format for the baseline label.
*/
PROCEDURE CusNumFmt IS
  chart og_object;
  template og_template;
  num number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  num := og_get_cust_num(template);
  og_set_cust_num(template, num/2);
END;
```

Reference Line Count Property

Description Is the number of reference lines that belong to the chart template.

Syntax

```
FUNCTION OG_Get_Reflinect
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Reference Line Count Property Examples

```
/*The following procedure reads the
**reference line count and prints the
**number to a text object.
*/
PROCEDURE RefLineCnt IS
  text og_object;
  chart og_object;
  template og_template;
  cnt number;
BEGIN
  text := og_get_object('text object');
  chart := og_get_object('chart');
  template := og_get_template(chart);
  cnt := og_get_reflinect(template);
  og_set_str(text, cnt);
END;
```

Second Y Axis Property

Description Specifies whether a second Y axis appears in the chart.

Syntax

```
PROCEDURE OG_Set_Second_Y
  (template  OG_Template,
   second_y  BOOLEAN);
FUNCTION OG_Get_Second_Y
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
second_y	Specifies whether a second Y axis appears in the chart.

Second Y Axis Property Examples

```
/* The following procedure determines if
**a second Y axis appears on the chart.
**If not, it adds a second one.
*/
PROCEDURE SecondY IS
  chart og_object;
  template og_template;
  axis boolean;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  axis := og_get_second_y(template);
    if axis = false then
      og_set_second_y(template, true);
   end if;
  og_update_chart(chart);
END;
```

Frame (Generic) Properties

Depth Size Property
Field Template Count Property
Frame Type Property
Legend Property
Legend Column Count Property
Name Property
Plot Frame Property
Root Property
Shadow Direction Property
Shadow Size Property

Depth Size Property

Description Specifies the amount of depth with which the chart frame and elements are drawn to provide them with a 3-dimensional look. The value of this property may be one of the following built-in constants:

OG_None_Depthsize
OG_Small_Depthsize
OG_Medium_Depthsize
OG_Large_None_Depthsize
OG_Xlarge_Depthsize

Syntax

```
PROCEDURE OG_Set_Depthsize
  (template   OG_Template,
   depthsize  NUMBER);
FUNCTION OG_Get_Depthsize
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
depthsize	Specifies the amount of depth with which the

chart frame and elements are drawn to provide them with a 3-dimensional look.

Depth Size Property Examples

```
/*The following reads the depth size
**of the chart, and changes the depth
**to a different value.
*/
PROCEDURE FrameDepth IS
  chart og_object;
  template og_template;
  depth number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  depth := og_get_depthsize(template);
  if depth = og_none_depthsize then
     og_set_depthsize(template, og_small_depthsize);
  elsif depth = og_small_depthsize then
     og_set_depthsize(template, og_medium_depthsize);
  elsif depth = og_medium_depthsize then
     og_set_depthsize(template, og_large_depthsize);
  elsif depth = og_large_depthsize then
     og_set_depthsize(template, og_xlarge_depthsize);
  elsif depth = og_xlarge_depthsize then
     og_set_depthsize(template, og_none_depthsize);
  end if;
END;
```

Field Template Count Property

Description Is the number of field templates that belong to the chart template.

Syntax

```
FUNCTION OG_Get_Ftempct
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Field Template Count Property Examples

```
/*The following procedure reads the number of
**the field template that belongs to the current
**template,and prints the value to a text object.
*/
PROCEDURE FTempCnt IS
  text og_object;
  chart og_object;
  template og_template;
  num number;
BEGIN
  text := og_get_object('text object');
  chart := og_get_object('chart');
  template := og_get_template(chart);
  num := og_get_ftempct(template);
  og_set_str(text, num);
 END;
```

Frame Type Property

Description Is the type of chart represented by this template The value of this property may be one of the following built-in constants:

OG_Axis_Frametype

OG_Pie_Frametype

OG_Table_Frametype

Syntax

```
FUNCTION OG_Get_Frametype
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Frame Type Property Examples

```
/*The following reads the frame type
**and prints the value to a text object.
*/
PROCEDURE FrameType IS
  text og_object;
  chart og_object;
  template og_template;
  num number;
BEGIN
  text := og_get_object('text object');
  chart := og_get_object('chart');
  template := og_get_template(chart);
  num := og_get_frametype(template);
  if num = og_axis_frametype then
    og_set_str(text, 'axis');
  elsif num = og_pie_frametype then
    og_set_str(text ,'pie');
  elsif num = og_table_frametype then
    og_set_str(text, 'table');
  end if;
END;
```

Legend Property

Description Specifies whether the chart's legend should be shown. (Not applicable to table charts.)

Syntax

```
PROCEDURE OG_Set_Legend
  (template  OG_Template,
   show      BOOLEAN);
FUNCTION OG_Get_Legend
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
show	Specifies whether the chart's legend should be shown. (Not applicable to table charts.)

Legend Property Examples

```
/*The following procedure determines
**if a legend is shown.  If a legend
**is shown, it hides it; if a legend
**is hidden, it shows it.
*/
PROCEDURE FrameLegend IS
  chart og_object;
  template og_template;
  val boolean;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  val := og_get_legend(template);
  if val = true then
     og_set_legend(template, false);
  else
     og_set_legend(template, true);
  end if;
END;
```

Legend Column Count Property

Description Is the number of columns used to display the labels that appear in the legend.

Syntax

```
PROCEDURE OG_Set_Legendcolct
  (template  OG_Template,
   colct     NUMBER);
FUNCTION OG_Get_Legendcolct
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
colct	Is the number of columns used to display the labels that appear in the legend.

Legend Column Count Property Examples

```
/*The following procedure reads the number of
**columns in the legend box.  If there is more
**than one column in the box, it changes the
**number of the columns to one; if there is
**one column, it changes the number of columns
**to two.
*/
PROCEDURE FrameLegendCol IS
  chart og_object;
  template og_template;
  num number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  num := og_get_legendcolct(template);
  if num > 1 then
     og_set_legendcolct(template, 1);
  else
     og_set_legendcolct(template, 2);
  end if;
END;
```

Name Property

Description Is the name of the chart template.

Syntax

```
PROCEDURE OG_Set_Frame_Name
  (template  OG_Template,
   name      VARCHAR2);
FUNCTION OG_Get_Frame_Name
  (template  OG_Template)
RETURN VARCHAR2;
```

Parameters

template	Is the chart template.
name	Is the name of the chart template.

Name Property Examples

```
/*The following reads the frame name.
**If the name is not 'template1', it sets
**it to 'template1'.
*/
PROCEDURE FrameName IS
  chart og_object;
  template og_template;
  name varchar2(30);
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  name := og_get_frame_name(template);
  if name != 'template1' then
    og_set_frame_name(template, 'template1');
  end if;
END;
```

Plot Frame Property

Description Specifies whether the rectangle that surrounds the chart should be shown. (Not applicable to pie charts.)

Syntax

```
PROCEDURE OG_Set_Plotframe
  (template  OG_Template,
   show      BOOLEAN);
FUNCTION OG_Get_Plotframe
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
show	Specifies whether the rectangle that surrounds the chart should be shown.

Plot Frame Property Examples

```
/*The following procedure determines
**whether a plot frame is drawn.  If
**true, it removes the plot frame;
**if false, it adds a plot frame to
**the current chart.
*/
PROCEDURE FramePlot IS
  chart og_object;
  template og_template;
  val boolean;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  val := og_get_plotframe(template);
  if val = true then
     og_set_plotframe(template, false);
  else
     og_set_plotframe(template, true);
  end if;
END;
```

Root Property

Description Is the handle to the chart template.

Syntax

```
FUNCTION OG_Get_Root
  (template  OG_Template)
RETURN OG_Object;
```

Parameters

template	Is the chart template.

Root Property Examples

```
/*The procedure gets the handle
**(root)of the chart object.
*/
PROCEDURE FrameRoot IS
  chart og_object;
  template og_template;
  root og_object;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  root := og_get_root(template);
END;
```

Shadow Direction Property

Description Specifies the direction of the shadow with which the chart frame and elements are drawn. The value of this property may be one of the following built-in constants:

OG_Upperright_Shadowdir
OG_Upperleft_Shadowdir
OG_Lowerright_Shadowdir
OG_Lowerleft_Shadowdir

Syntax

```
PROCEDURE OG_Set_Shadowdir
  (template   OG_Template,
   shadowdir  NUMBER);
FUNCTION OG_Get_Shadowdir
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
shadowdir	Specifies the direction of the shadow with which the chart frame and elements are drawn.

Shadow Direction Property Examples

```
*/The following reads the shadow direction of
**the chart, and changes it to a different
**value.
*/
PROCEDURE FrameShadowDir IS
  chart og_object;
  template og_template;
  shadow number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  shadow := og_get_shadowdir(template);
  if shadow = og_upperright_shadowdir then
     og_set_shadowdir(template, og_lowerleft_shadowdir);
  elsif shadow = og_lowerleft_shadowdir then
     og_set_shadowdir(template, og_upperleft_shadowdir);
  elsif shadow = og_upperleft_shadowdir then
     og_set_shadowdir(template, og_lowerright_shadowdir);
  elsif shadow = og_lowerright_shadowdir then
     og_set_shadowdir(template, og_upperright_shadowdir);
  end if;
END;
```

Shadow Size Property

Description Specifies the size of the shadow with which the chart frame and elements are drawn. The value of this property may be one of the following built-in constants:

OG_None_Shadowsize
OG_Small_Shadowsize
OG_Medium_Shadowsize
OG_Large_Shadowsize
OG_Xlarge_Shadowsize

Syntax

```
PROCEDURE OG_Set_Shadowsize
  (template    OG_Template,
   shadowsize  NUMBER);
FUNCTION OG_Get_Shadowsize
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
shadowsize	Specifies the size of the shadow with which the chart frame and elements are drawn.

Shadow Size Property Examples

```
/*The following procedure reads the shadow size
**of the chart, and changes the size to a
**different value.
*/
PROCEDURE FrameShadow IS
  chart og_object;
  template og_template;
  shadow number;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  shadow := og_get_shadowsize(template);
  if shadow = og_none_shadowsize then
     og_set_shadowsize(template, og_small_shadowsize);
  elsif shadow = og_small_shadowsize then
     og_set_shadowsize(template, og_medium_shadowsize);
  elsif shadow = og_medium_shadowsize then
     og_set_shadowsize(template, og_large_shadowsize);
  elsif shadow = og_large_shadowsize then
     og_set_shadowsize(template, og_xlarge_shadowsize);
  elsif shadow = og_xlarge_shadowsize then
     og_set_shadowsize(template, og_none_shadowsize);
  end if;
END;
```

Frame (Pie Chart) Properties

Categories Property
Category Date Format Property
Category Number Format Property
Data Values Property
No Overlap Property
Other Property
Percent Format Property
Percent Values Property
Plot Order Property
Ticks Property
Usage Property
Usage Value Property
Value Format Property

Categories Property

Description Specifies whether each pie slice is labeled with the name of the category it represents.

Syntax

```
PROCEDURE OG_Set_Categs
  (template  OG_Template,
   categs    BOOLEAN);
FUNCTION OG_Get_Categs
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.

categs	Specifies whether each pie slice is labeled with the name of the category it represents.

Categories Property Examples

```
/* The following procedure gets
**information about the relationship
**between individual pie slices and
**the complete chart.  If the current
**relationship is TOTALVALUE, the
**procedure resets the relationship
** to PERCENTAGE with a value of  50;
**If the current relationship is
**PERCENTAGE, the procedure resets
**the relationship to TOTALVALUE with
**a value of 400000.
*/
PROCEDURE PieUsage IS
  pie og_object;
  template og_template;
  usage number;
  usagevalue number;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  usage := og_get_usage(template);
  if usage = og_totalvalue_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_pct_usage, 50);
   elsif usage = og_pct_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_totalvalue_usage, 400000);
  end if;
  og_update_chart(pie);
END;
```

Category Date Format Property

Description Specifies the date format for the category labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Catdatefmt
  (template    OG_Template,
   catdatefmt  VARCHAR2);
FUNCTION OG_Get_Catdatefmt
  (template  OG_Template)
RETURN VARCHAR2;
```

Parameters

template	Is the chart template.
catdatefmt	Specifies the date format for the category labels. This must be a valid SQL format string.

Category Date Format Property Examples

```
/*The following procedure changes the
**pie slice label's date format if the
**format is not currently
**'DD-MM-YY'.
*/
PROCEDURE CatDateFmt IS
  pie og_object;
  template og_template;
  format varchar2(20);
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  format := og_get_catdatefmt(template);
  if format != 'MM-DD-YY' then
   og_set_catdatefmt(template, 'MM-DD-YY');
  end if;
  og_update_chart(pie);
END;
```

Category Number Format Property

Description Specifies the number format for the category labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Catnumfmt
  (template   OG_Template,
   catnumfmt  VARCHAR2);
FUNCTION OG_Get_Catnumfmt
  (template  OG_Template)
RETURN VARCHAR2;
```

Parameters

template	Is the chart template.
catnumfmt	Specifies the number format for the category labels. This must be a valid SQL format string.

Category Number Format Property Examples

```
/*The following procedure changes the
**pie slice label's number format if
**the format is not currently
**'9,9,9,9'.
*/
PROCEDURE CatNumFmt IS
  pie og_object;
  template og_template;
  format varchar2(20);
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  format := og_get_catnumfmt(template);
  if format != '9,9,9,9' then
    og_set_catnumfmt(template, '9,9,9,9');
  end if;
  og_update_chart(pie);
END;
```

Data Values Property

Description Specifies whether each pie slice is labeled with its data value.

Syntax

```
PROCEDURE OG_Set_Datavals
  (template  OG_Template,
   datavals  BOOLEAN);
FUNCTION OG_Get_Datavals
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
datavals	Specifies whether each pie slice is labeled with its data value.

Data Values Property Examples

```
/* The following procedure hides/shows
** the data value for each pie slice.
*/
PROCEDURE DataVals IS
  pie og_object;
  template og_template;
  val boolean;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  val := og_get_datavals(template);
  if val = true then
   og_set_datavals(template, false);
  elsif val = false then
   og_set_datavals(template, true);
  end if;
  og_update_chart(pie);
END;
```

No Overlap Property

Description Specifies that the labels for the pie slices should not overlap each other.

Syntax

```
PROCEDURE OG_Set_Nooverlap
  (template   OG_Template,
   nooverlap  BOOLEAN);
FUNCTION OG_Get_Nooverlap
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
nooverlap	Specifies that the labels for the pie slices should not overlap each other.

No Overlap Property Examples

```
/*The following procedure determines if
**pie slice labels are allowed to overlap.
**If overlapping is allowed, the procedure
**disallows it.
*/
PROCEDURE NoOverlap IS
  pie og_object;
  template og_template;
  val boolean;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  val := og_get_nooverlap(template);
  if val = false then
     og_set_nooverlap(template, true);
  end if;
  og_update_chart(pie);
END;
```

Other Property

Description Specifies the minimum percentage of the chart that a data value must represent in order for it to appear as an individual slice in the pie chart. Data values that represent percentages below this number are combined into a single pie slice with the label "Other".

Syntax

```
PROCEDURE OG_Set_Other
  (template  OG_Template,
   other     NUMBER);
FUNCTION OG_Get_Other
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
other	Specifies the minimum percentage of the chart that a data value must represent in order for it to appear as an individual slice in the pie chart. Data values that represent percentages below this number are combined into a single pie slice with the label "Other".

Other Property Examples

```
/*The following procedure doubles
**the percentage value for which
**any chart slice with a value
**less than or equal to the
**percentage value will be labeled
**”other.”
*/
PROCEDURE Other IS
  pie og_object;
  template og_template;
  num number;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  num := og_get_other(template);
  og_set_other(template, num*2);
  og_update_chart(pie);
END;
```

Percent Format Property

Description Specifies the number format for the percent value labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Pctfmt
  (template  OG_Template,
   pctfmt    VARCHAR2);
FUNCTION OG_Get_Pctfmt
  (template  OG_Template)
RETURN VARCHAR2;
```

Parameters

template	Is the chart template.
pctfmt	Specifies the number format for the percent value labels. This must be a valid SQL format string.

Percent Format Property Examples

```
/*The following procedure hides/shows the
**percent value for each pie slice.
*/
PROCEDURE PctVals IS
  pie og_object;
  template og_template;
  val boolean;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  val := og_get_pctvalues(template);
  if val = true then
   og_set_pctvalues(template, false);
  elsif val = false then
   og_set_pctvalues(template, true);
  end if;
  og_update_chart(pie);
END;
```

Percent Values Property

Description Specifies whether each pie slice is labeled with the percentage of the complete chart it represents.

Syntax

```
PROCEDURE OG_Set_Pctvalues
  (template   OG_Template,
   pctvalues  BOOLEAN);
FUNCTION OG_Get_Pctvalues
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
pctvalues	Specifies whether each pie slice is labeled with the percentage of the complete chart it represents.

Percent Values Property Examples

```
/* The following procedure hides/shows
**the percent value for each pie slice.
*/
PROCEDURE PctVals IS
  pie og_object;
  template og_template;
  val boolean;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  val := og_get_pctvalues(template);
  if val = true then
   og_set_pctvalues(template, false);
  elsif val = false then
   og_set_pctvalues(template, true);
  end if;
  og_update_chart(pie);
END;
```

Plot Order Property

Description Specifies the direction in which the data values are plotted. The value of this property may be one of the following built-in constants:

OG_Ccw_Plotorder Means values are plotted in a counter-clockwise direction.

OG_Cw_Plotorder Means values are plotted in a clockwise direction.

Syntax

```
PROCEDURE OG_Set_Plotorder
  (template   OG_Template,
   plotorder  NUMBER);
FUNCTION OG_Get_Plotorder
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
plotorder	Specifies the direction in which the data values are plotted.

Plot Order Property Examples

```
/*The following procedure reads the
**direction in which the data values
**are plotted, and reverses the
**plotting direction.
*/
PROCEDURE plotOrder IS
  pie og_object;
  template og_template;
  porder number;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  porder := og_get_plotorder(template);
  if porder = og_cw_plotorder then
     og_set_plotorder(template, og_ccw_plotorder);
  else
     og_set_plotorder(template, og_cw_plotorder);
  end if;
  og_update_chart(pie);
END;
```

Ticks Property

Description Specifies whether the tick marks that connect each pie slice to its label are shown.

Syntax

```
PROCEDURE OG_Set_Ticks
  (template  OG_Template,
   ticks     BOOLEAN);
FUNCTION OG_Get_Ticks
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
ticks	Specifies whether the tick marks that connect each pie slice to its label are shown.

Ticks Property Examples

```
/*The following procedure hides/
**shows the ticks for each pie
**slice.
*/
PROCEDURE ticks IS
  pie og_object;
  template og_template;
  val boolean;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  val := og_get_ticks(template);
  if val = true then
   og_set_ticks(template, false);
  else
   og_set_ticks(template, true);
  end if;
  og_update_chart(pie);
END;
```

Usage Property

Description Specifies the relationship between the individual pie slices and the complete chart. The value of this property may be one of the following built-in constants:

OG_Totalvalue_Usage

OG_Pct_Usage

Syntax

```
PROCEDURE OG_Set_Usage
  (template  OG_Template,
   usage       NUMBER,
   usagevalue  NUMBER);
FUNCTION OG_Get_Usage
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
usage	Specifies the relationship between the individual pie slices and the complete chart.
usagevalue	Each pie slice is plotted as if its data value is a percentage of the total value specified here. (Valid only is *usage* is set to OG_TOTALVALUE_USAGE.

Usage Property Examples

```
/*The following procedure gets
**information about the relationship
**between individual pie slices and
**the complete chart.  If the current
**relationship is TOTALVALUE, the procedure
**resets the relationship to PERCENTAGE
**with a value of  50.  If the current
**relationship is PERCENTAGE, the procedure
**resets the relationship to TOTALVALUE
**with a value of 400000.
*/
PROCEDURE PieUsage IS
  pie og_object;
  template og_template;
  usage number;
  usagevalue number;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  usage := og_get_usage(template);
  if usage = og_totalvalue_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_pct_usage, 50);
   elsif usage = og_pct_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_totalvalue_usage, 400000);
  end if;
  og_update_chart(pie);
END;
```

Usage Value Property

Description Each pie slice is plotted as if its data value is a percentage of the total value specified here.

Syntax

(See OG_Set_Usage, above.)

```
FUNCTION OG_Get_Usagevalue
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Usage Value Property Examples

```
/*The following procedure gets
**information about the relationship
**between individual pie slices and
**the complete chart.  If the current
**relationship is TOTALVALUE, the procedure
**resets the relationship to PERCENTAGE
**with a value of  50.  If the current
**relationship is PERCENTAGE, the procedure
**resets the relationship to TOTALVALUE
**with a value of 400000.
*/
PROCEDURE PieUsage IS
  pie og_object;
  template og_template;
  usage number;
  usagevalue number;
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  usage := og_get_usage(template);
  if usage = og_totalvalue_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_pct_usage, 50);
   elsif usage = og_pct_usage then
     usagevalue := og_get_usagevalue(template);
     og_set_usage(template, og_totalvalue_usage, 400000);
  end if;
  og_update_chart(pie);
END;
```

Value Format Property

Description Specifies the number format for the data value labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Valuefmt
  (template     OG_Template,
   valuenumfmt  VARCHAR2);
FUNCTION OG_Get_Valuefmt
  (template  OG_Template)
RETURN VARCHAR2;
```

Parameters

template	Is the chart template.
valuenumfmt	Specifies the number format for the data value labels.

Value Format Property Examples

```
/*The following procedure changes the pie
**slice label's value format if the format
**is not currently '0999'.
*/
PROCEDURE ValFmt IS
  pie og_object;
  template og_template;
  format varchar2(20);
BEGIN
  pie := og_get_object('pie');
  template := og_get_template(pie);
  format := og_get_valuefmt(template);
  if format != '0999' then
    og_set_valuefmt(template, '0999');
  end if;
  og_update_chart(pie);
END;
```

Frame (Table Chart) Properties

Auto Maximum Property
Auto Minimum Property
Column Names Property
Grid Count Property
Horizontal Grid Property
Maximum Number Of Rows Property
Minimum Number Of Rows Property
Vertical Grid Property

Auto Maximum Property

Description Specifies whether the maximum number of rows that appear on the chart is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Automax
  (template  OG_Template,
   automax   BOOLEAN,
   maxrows   NUMBER);
FUNCTION OG_Get_Automax
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
automax	Specifies whether the maximum number of rows that appear on the chart is set to *Auto*.
maxrows	Specifies the maximum number of rows that appear on the chart (if *automax* is FALSE).

Auto Maximum Property Examples

```
/*The following procedure determines if
**there is a maximum number of rows to
**be displayed in the table or if the
**number of rows is automatically
**determined.  If the number of
**rows is not automatically determined,
**the procedure reads the number of rows
**the table displays currently and resets
**it to be automatically determined.
*/
PROCEDURE AutoMax IS
  table1 og_object;
  template og_template;
  val boolean;
  maxrows number := 2;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_automax(template);
  if val = false then
     maxrows := og_get_maxrows(template);
     og_set_automax(template, true, maxrows/2);
  end if;
  og_update_chart(table1);
END;
```

Auto Minimum Property

Description Specifies whether the minimum number of rows that appear on the chart is set to *Auto*.

Syntax

```
PROCEDURE OG_Set_Automin
  (template  OG_Template,
   automin   BOOLEAN,
   minrows   NUMBER);
FUNCTION OG_Get_Automin
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
automin	Specifies whether the minimum number of rows that appear on the chart is set to *Auto*.
minrows	Specifies the minimum number of rows that appear on the chart (if *automin* is FALSE).

Auto Minimum Property Examples

```
/*"The following procedure
**determines if there is a
**minimum number of rows that
**must be displayed in the
**table or whether the number of
**rows is automatically determined.
**If the number of rows is not
**automatically determined, the procedure
**reads the number of rows the table
**currently displays and resets it to
**be automatically determined.
 */
PROCEDURE AutoMax IS
  table1 og_object;
  template og_template;
  val boolean;
  maxrows number := 2;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_automax(template);
  if val = false then
     maxrows := og_get_maxrows(template);
     og_set_automax(template, true, maxrows/2);
  end if;
  og_update_chart(table1);
END;
```

Column Names Property

Description Specifies whether the names of the columns appear as the first row in the chart.

Syntax

```
PROCEDURE OG_Set_Cname
  (template  OG_Template,
   cname     BOOLEAN);
FUNCTION OG_Get_Cname
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
cname	Specifies whether the names of the columns appear as the first row in the chart.

Column Names Property Examples

```
/*The following procedure hides/shows the
**table's column names.
*/
PROCEDURE ColNames IS
  table1 og_object;
  template og_template;
  val boolean;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_cname(template);
  if val = true then
    og_set_cname(template, false);
  elsif val = false then
    og_set_cname(template, true);
  end if;
  og_update_chart(table1);
END;
```

Grid Count Property

Description Is the number of rows of data plotted before each horizontal grid line is drawn (if *Horizontal Grid* is set to TRUE).

Syntax

```
PROCEDURE OG_Set_Gridct
  (template  OG_Template,
   gridct    NUMBER);
FUNCTION OG_Get_Gridct
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.
gridct	Is the number of rows of data plotted before each horizontal grid line is drawn (if *Horizontal Grid* is set to TRUE).

Grid Count Property Examples

```
/*The following procedure doubles
**the grid count of the table.
*/
PROCEDURE gridcnt IS
  table1 og_object;
  template og_template;
  cnt number;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  cnt := og_get_gridct(template);
  og_set_gridct(template, cnt*2);
  og_update_chart(table1);
END;
```

Horizontal Grid Property

Description Specifies whether horizontal grid lines appear between the rows.

Syntax

```
PROCEDURE OG_Set_Hgrid
  (template  OG_Template,
   hgrid     BOOLEAN);
FUNCTION OG_Get_Hgrid
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
hgrid	Specifies whether horizontal grid lines appear between the rows.

Horizontal Grid Property Examples

```
/*The following procedure hides/shows
**horizontal grid lines.
*/
PROCEDURE HoriGrid IS
  table1 og_object;
  template og_template;
  val boolean;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_hgrid(template);
  if val = true then
     og_set_hgrid(template, false);
  elsif val = false then
     og_set_hgrid(template, true);
  end if;
  og_update_chart(table1);
END;
```

Maximum Number of Rows Property

Description Specifies the maximum number of rows that appear on the chart (if *Auto Maximum* is FALSE).

Syntax

(See OG_Set_Automax, above.)

```
FUNCTION OG_Get_Maxrows
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Maximum Number of Rows Property Examples

```
/*The following procedure determines
**if there is a maximum number of rows to
**be displayed in the table or if the
**number of rows is automatically
**determined.  If the number of rows is
**automatically determined, the procedure
**reads the number of rows the table
**displays currently and resets it to be
**automatically determined.
*/
PROCEDURE AutoMax IS
  table1 og_object;
  template og_template;
  val boolean;
  maxrows number := 2;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_automax(template);
  if val = false then
     maxrows := og_get_maxrows(template);
     og_set_automax(template, true, maxrows/2);
  end if;
  og_update_chart(table1);
END;
```

Minimum Number of Rows Property

Description Specifies the maximum number of rows that appear on the chart (if *Auto Minimum* is FALSE).

Syntax

(See OG_Set_Automin, above.)

```
FUNCTION OG_Get_Minrows
  (template  OG_Template)
RETURN NUMBER;
```

Parameters

template	Is the chart template.

Minimum Number of Rows Property Examples

```
/*"The following procedure
**determines if there is a
**minimum number of rows that
**must be displayed in the
**table or whether the number of
**rows is automatically determined.
**If the number of rows is not
**automatically determined, the procedure
**reads the number of rows the table
**currently displays and resets it to
**be automatically determined.
 */
PROCEDURE AutoMax IS
  table1 og_object;
  template og_template;
  val boolean;
  maxrows number := 2;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_automax(template);
  if val = false then
     maxrows := og_get_maxrows(template);
     og_set_automax(template, true, maxrows/2);
  end if;
  og_update_chart(table1);
END;
```

Vertical Grid Property

Description Specifies whether vertical grid lines appear between the columns.

Syntax

```
PROCEDURE OG_Set_Vgrid
  (template  OG_Template,
   vgrid     BOOLEAN);
FUNCTION OG_Get_Vgrid
  (template  OG_Template)
RETURN BOOLEAN;
```

Parameters

template	Is the chart template.
vgrid	Specifies whether vertical grid lines appear between the columns.

Vertical Grid Property Examples

```
/* The following procedure hides/shows
**vertical grid lines.
*/
PROCEDURE VertGrid IS
  table1 og_object;
  template og_template;
  val boolean;
BEGIN
  table1 := og_get_object('table');
  template := og_get_template(table1);
  val := og_get_vgrid(template);
  if val = true then
     og_set_vgrid(template, false);
  elsif val = false then
     og_set_vgrid(template, true);
  end if;
  og_update_chart(table1);
END.
```

Field Template (Generic) Properties

Color Rotation Property
Date Format Property
Name Property
Number Format Property
Root Property

Color Rotation Property

Description Specifies whether Graphics Builder automatically rotates through the color or pattern palette to select a unique shading for each field that uses this field template. The value of this property may be one of the following built-in constants:

OG_None_Colorrot
OG_Auto_Colorrot
OG_Color_Colorrot
OG_Pattern_Colorrot
OG_Both_Colorrot

Syntax

```
PROCEDURE OG_Set_Colorrot
  (ftemp     OG_Ftemp,
   colorrot  NUMBER);
FUNCTION OG_Get_Colorrot
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
colorrot	Specifies whether Graphics Builder

automatically rotates through the color or pattern palette to select a unique shading for each field that uses this field template.

Color Rotation Property Examples

```
/*The following procedure reads if any
**color rotation is applied to the chart.
**If none has been applied, it applies
**AUTO color rotation.  If another method
**of color rotation is currently applied,
**it changes the rotation to NONE.
*/
PROCEDURE fieldColRot IS
  ftemp og_ftemp;
  color number;
BEGIN
  ftemp := og_get_ftemp(og_get_template(og_get_object('chart')),0);
  color := og_get_colorrot(ftemp);
  if color = og_none_colorrot then
     og_set_colorrot(ftemp, og_auto_colorrot);
  else
     og_set_colorrot(ftemp, og_none_colorrot);
 end if;
     og_update_chart(og_get_object('chart'));
END;
```

Date Format Property

Description Specifies the date format for the field labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Datefmt
  (ftemp    OG_Ftemp,
   date_fmt  VARCHAR2);
FUNCTION OG_Get_Datefmt
  (ftemp  OG_Ftemp)
RETURN VARCHAR2;
```

Parameters

ftemp	Is the field template being described.
date_fmt	Specifies the date format for the field labels. This must be a valid SQL format string.

Date Format Property Examples

```
*/The following procedure
**determines if label number
**formats are all'9,9,9,9'.
**If not, it changes them all
**to '9,9,9,9'.
*/
PROCEDURE fieldDateFmt IS
  ftemp og_ftemp;
  datefmt varchar2(20);
BEGIN
  ftemp := og_get_ftemp(og_get_template(og_get_object('chart')),0);
  datefmt := og_get_datefmt(ftemp);
  if datefmt != 'DD-MM-YYYY' then
    og_set_datefmt(ftemp, 'DD-MM-YYYY');
  end if;
END;
```

Name Property

Description Is the name of the field template.

Syntax

```
PROCEDURE OG_Set_Ftemp_Name
  (ftemp  OG_Ftemp,
   name   VARCHAR2);
FUNCTION OG_Get_Ftemp_Name
  (ftemp  OG_Ftemp)
RETURN VARCHAR2;
```

Parameters

ftemp	Is the field template being described.
name	Is the name of the field template.

Name Property Examples

```
/*The following button procedure
**appends a '1' to the current
**field template's name.
*/
PROCEDURE fieldname IS
  ftemp og_ftemp;
  chart og_object;
  name varchar2(20);
BEGIN
  ftemp := og_get_ftemp(og_get_template(og_get_object('chart')),0);
  name := og_get_ftemp_name(ftemp);
  og_set_ftemp_name(ftemp, name||'1');
END;
```

Number Format Property

Description Specifies the number format for the field labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Syntax

```
PROCEDURE OG_Set_Numfmt
  (ftemp    OG_Ftemp,
   num_fmt  VARCHAR2);
FUNCTION OG_Get_Numfmt
  (ftemp  OG_Ftemp)
RETURN VARCHAR2;
```

Parameters

ftemp	Is the field template being described.
num_fmt	Specifies the number format for the field labels. This must be a valid SQL format string.

Number Format Property Examples

```
*/The following procedure
**determines if the labels' number
**format is '9,9,9,9'.  If
**not, it changes the format
**to '9,9,9,9'.
*/
PROCEDURE fieldNumFmt IS
  ftemp og_ftemp;
  numfmt varchar2(20);
BEGIN
  ftemp := og_get_ftemp(og_get_template(og_get_object('chart')),0);
  numfmt := og_get_numfmt(ftemp);
  if numfmt != '9,9,9,9' then
    og_set_numfmt(ftemp, '9,9,9,9');
  end if;
END;
```

Root Property

Description Is a handle to the chart template to which the field template belongs.

Syntax

```
FUNCTION OG_Get_Root
  (ftemp  OG_Ftemp)
RETURN OG_Object;
```

Parameters

ftemp	Is the field template being described.

Root Property Examples

```
/*The following procedure gets
**a chart’s field template handles.
*/
PROCEDURE fieldname IS
  ftemp og_ftemp;
  root og_object;
BEGIN
  ftemp := og_get_ftemp(og_get_template(og_get_object('chart')),0);
  root := og_get_root(ftemp);
END;
```

Field Template (Axis Chart) Properties

Axis Property
Curve Fit Property
Label Rotation Property
Line Style Property
Overlap Property
Plot Position Property
Plot Type Property

Axis Property

Description Specifies the axis to which data values are compared to determine how the field is plotted. The value of this property may be one of the following built-in constants:
OG_Y1_Axis
OG_Y2_Axis
Syntax

```
PROCEDURE OG_Set_Axis
  (ftemp  OG_Ftemp,
   axis   NUMBER);

FUNCTION OG_Get_Axis
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
axis	Specifies the axis to which data values are compared to determine how the field is plotted.

Axis Property Examples

```
/*The following procedure
**rotates the main Y axis the
**chart currently refers to
**(if there is more than one
**Y axis)and switches the main
**Y axis to a different Y axis.
*/

PROCEDURE axis IS
  axis number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  axis := og_get_axis(ftemp);
  if axis = og_y1_axis then
     og_set_axis(ftemp, og_y2_axis);
  elsif axis = og_y2_axis then
     og_set_axis(ftemp, og_y1_axis);
  end if;
  og_update_chart(chart);
END;
```

Curve Fit Property

Description Specifies whether a curve fit is applied to the chart and, if so, which algorithm is used. The value of this property may be one of the following built-in constants:

OG_No_Curvefit
OG_Linear_Curvefit
OG_LOG_Curvefit
OG_Exp_Curvefit
OG_Power_Curvefit

Syntax

```
PROCEDURE OG_Set_Curvefit
  (ftemp     OG_Ftemp,
   curvefit  NUMBER);

FUNCTION OG_Get_Curvefit
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
curvefit	Specifies whether a curve fit is applied to the chart and, if so, which algorithm is used.

Curve Fit Property Examples

```
/*The following procedure determines
**if a curve fit is applied to the chart.
**If not, it applies a Linear CurveFit
**to the chart.  If a curve fit is currently
**applied to the chart, it removes it.
*/

PROCEDURE CurveFit IS
  curve number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  curve := og_get_curvefit(ftemp);
  if curve = og_no_curvefit then
    og_set_curvefit(ftemp,og_linear_curvefit);
  else
    og_set_curvefit(ftemp,og_no_curvefit);
  end if;
  og_update_chart(chart);
END;
```

Label Rotation Property

Description Specifies the rotation angle of the labels for a field with a label plot type. The value of this property may be one of the following built-in constants:

OG_Ccw_Rotation Means counter-clockwise rotation.

OG_Cw_Rotation Means clockwise rotation.

OG_No_Rotation

Syntax

```
PROCEDURE OG_Set_Labelrot
  (ftemp    OG_Ftemp,
   linesty  NUMBER);

FUNCTION OG_Get_Labelrot
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
linesty	Specifies the rotation angle of the labels for a field with a label plot type.

Label Rotation Property Examples

```
/* The following procedure rotates a
**chart's rotation labels.
*/

PROCEDURE lblrot IS
  rot number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  rot := og_get_labelrot(ftemp);
  if rot = og_no_rotation then
    og_set_labelrot(ftemp,og_cw_rotation);
  elsif rot = og_cw_rotation then
    og_set_labelrot(ftemp, og_ccw_rotation);
  elsif rot = og_ccw_rotation then
    og_set_labelrot(ftemp, og_no_rotation);
  end if;
  og_update_chart(chart);
END;
```

Line Style Property

Description Specifies the line style used to connect the data points of a field with a line plot type. The value of this property may be one of the following built-in constants:
OG_Spline_Linestyle
OG_Step_Linestyle
OG_Straight_Linestyle
Syntax

```
PROCEDURE OG_Set_Linesty
  (ftemp    OG_Ftemp,
   linesty  NUMBER);

FUNCTION OG_Get_Linesty
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
linesty	Specifies the line style used to connect the data points of a field with a line plot type.

Line Style Property Examples

```
/*The following procedure rotates
**the line style of a chart.
*/

PROCEDURE linesty IS
  style number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  style := og_get_linesty(ftemp);
  if style = og_spline_linestyle then
     og_set_linesty(ftemp, og_step_linestyle);
  elsif style = og_step_linestyle then
     og_set_linesty(ftemp, og_straight_linestyle);
  elsif style = og_straight_linestyle then
     og_set_linesty(ftemp, og_spline_linestyle);
  end if;
  og_update_chart(chart);
END;
```

Overlap Property

Description Specifies the percentage by which bars representing data values from multiple fields in a bar or column chart overlap each other.

Syntax

```
PROCEDURE OG_Set_Overlap
  (ftemp    OG_Ftemp,
   overlap  NUMBER);

FUNCTION OG_Get_Overlap
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
overlap	Specifies the percentage by which bars representing data values from multiple fields in a bar or column chart overlap each other.

Overlap Property Examples

```
/*The following procedure reads
**the overlap percentage that has
**been specified. If the specified
**percentage is between 0 to 50,
**it redraws the column using
**90% overlap, if the percentage is
**over 90%,it redraws the columns
**with 0% overlap.
*/

PROCEDURE overlap IS
  percent number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  percent := og_get_overlap(ftemp);
  if percent between 0 and 50 then
     og_set_overlap(ftemp, 90);
  else
     og_set_overlap(ftemp, 0);
  end if;
END;
```

Plot Position Property

Description Specifies for each category the relationship between the data values of two or more fields. The value of this property may be one of the following built-in constants:

OG_Normal_Plotpos

OG_Fromprev_Plotpos

OG_Stacked_Plotpos

Syntax

```
PROCEDURE OG_Set_Plotpos
  (ftemp    OG_Ftemp,
   plotpos  NUMBER);

FUNCTION OG_Get_Plotpos
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
plotpos	Specifies for each category the relationship between the data values of two or more fields.

Plot Position Property Examples

```
/*The following button procedure rotates
**the plot position of columns in a chart.
*/

PROCEDURE plotpos IS
  pos number;
  ftemp og_ftemp;
  chart og_object;
BEGIN
  chart := og_get_object('chart');
  ftemp := og_get_ftemp(og_get_template(chart),0);
  pos := og_get_plotpos(ftemp);
   if pos = og_normal_plotpos then
     og_set_plotpos(ftemp,og_fromprev_plotpos);
   elsif pos = og_fromprev_plotpos then
     og_set_plotpos(ftemp, og_stacked_plotpos);
   elsif pos = og_stacked_plotpos then
     og_set_plotpos(ftemp, og_normal_plotpos);
   end if;
  og_update_chart(chart);
END;
```

Plot Type Property

Description Specifies the elements used to plot this field on the chart. The value of this property may be one of the following built-in constants:

OG_None_Plottype
OG_Bar_Plottype
OG_Line_Plottype
OG_Symbol_Plottype
OG_Fill_Plottype
OG_Spike_Plottype
OG_Label_Plottype

Syntax

```
PROCEDURE OG_Set_Plottype
  (ftemp     OG_Ftemp,
   plottype  NUMBER);

FUNCTION OG_Get_Plottype
  (ftemp  OG_Ftemp)
RETURN NUMBER;
```

Parameters

ftemp	Is the field template being described.
plottype	Specifies the elements used to plot this field on the chart.

Plot Type Property Examples

```
/*On a mouse click, the following
**procedure rotates the plot type
**of a chart.
*/

PROCEDURE Plottype (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
chart og_object;
template og_template;
ftemp og_ftemp;
num number;
BEGIN
chart := og_get_object('chart');
template := og_get_template(chart);
ftemp := og_get_ftemp(template, 0);
num := og_get_plottype(ftemp);
if num = og_none_plottype then
   og_set_plottype(ftemp, og_bar_plottype);
elsif num = og_bar_plottype then
   og_set_plottype(ftemp, og_line_plottype);
elsif num = og_line_plottype then
   og_set_plottype(ftemp, og_symbol_plottype);
elsif num = og_symbol_plottype then
   og_set_plottype(ftemp, og_fill_plottype);
elsif num = og_fill_plottype then
   og_set_plottype(ftemp, og_spike_plottype);
elsif num = og_spike_plottype then
   og_set_plottype(ftemp, og_label_plottype);
elsif num = og_label_plottype then
   og_set_plottype(ftemp, og_none_plottype);
end if;
og_update_chart(chart);
END;
```

Generic Properties

Button Procedure Property
Column Property
Events Property
Execute Query Property
Format Trigger Property
Hide Object Property
Inner Bounding Box Property
Name Property
Object Type Property
Outer Bounding Box Property
Parent Property
Set Parameter Property

Button Procedure Property

Description Is the handle to the button procedure to be associated with this object. Note that the *Events* property must also be set properly in order to ensure that this procedure receives the desired mouse events.

Syntax

```
PROCEDURE OG_Set_Button
  (object          OG_Object,
   buttonproc      OG_Buttonproc,
   damage          BOOLEAN        :=  TRUE,
   update_bbox     BOOLEAN        :=  TRUE);
```

```
FUNCTION OG_Get_Button
  (object  OG_Object)
RETURN OG_Buttonproc;
```

Parameters

object	Is the object being described.
buttonproc	Is the handle to the button procedure to be associated with this object.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Button Procedure Property Examples

```
/*The following procedure reads the button
**procedure from a rectangle object, and adds
**the same procedure to a circle object.
*/
PROCEDURE transfer (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  rect og_object;
  circle og_object;
  proc og_buttonproc;
BEGIN
  rect := og_get_object('rect');
  circle := og_get_object('circle');
  proc := og_get_button(rect);
  og_set_button(circle, proc);
END;
```

Column Property

Description Is the column value to which the parameter is set when the object is selected. This property applies only to chart elements.

Syntax

```
PROCEDURE OG_Set_Keycol
  (object  OG_Object,
   keycol  VARCHAR2);
FUNCTION OG_Get_Keycol
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
keycol	Is the column value to which the parameter is set when the object is selected.

Column Property Examples

```
/*The following procedure reads
**the column value of a parameter
**and assigns a different value to it
*/
PROCEDURE GenColumn IS
  rect og_object;
  param varchar2(20);
BEGIN
  rect := og_get_object('rect');
  param := og_get_keycol(rect);
  og_set_keycol(rect, 'init');
END;
```

Events Property

Description Is the type of mouse events that the procedure specified by the button property should receive. The value of this property may be one of the built-in constants listed below. To enable the procedure to receive multiple event types, set this property to be the sum of the constants for the desired events. Note that OG_Mouse_Move_Up and OG_Mouse_Move_Down are only used for traversing display layers.

OG_No_Events
OG_Mouse_Down
OG_Mouse_Up
OG_Mouse_Move_Down

Syntax

```
PROCEDURE OG_Set_Events
  (object       OG_Object,
   events       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Events
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
events	Is the type of mouse events that the procedure specified by the button property should receive.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Events Property Examples

```
/*The following procedure reads
**the current mouse event in an object,
**and assigns a different event to it.
*/
PROCEDURE Events IS
  rect og_object;
  events number;
BEGIN
  rect := og_get_object('rect');
  events := og_get_events(rect);
  if events = og_no_events then
     og_set_events(rect, og_mouse_down);
  elsif events = og_mouse_down then
     og_set_events(rect, og_mouse_up);
  elsif events = og_mouse_up then
     og_set_events(rect, og_mouse_move_down);
  elsif events = og_mouse_move_down then
     og_set_events(rect, og_no_events);
  end if;
END;
```

Execute Query Property

Description Specifies the query to execute when the object is selected.

Syntax

```
PROCEDURE OG_Set_Execquery
  (object     OG_Object,
   execquery  OG_Query);
FUNCTION OG_Get_Execquery
  (object  OG_Object)
RETURN OG_Query;
```

Parameters

object	Is the object being described.
execquery	Specifies the query to execute when the object is selected.

Execute Query Property Examples

```
/*The following procedure reads the specified
**query of the object, and assigns a different
**query to it.
*/
PROCEDURE GenQuery IS
  rect og_object;
  query og_query;
  query1 og_query;
BEGIN
  rect := og_get_object('rect');
  query := og_get_execquery(rect);
  query1 := og_get_query('query1');
  og_set_execquery(rect, query1);
END;
```

Format Trigger Property

Description Is the object's format trigger. This property applies only to chart elements.

Syntax

```
PROCEDURE OG_Set_Fmttrig
  (object   OG_Object,
   fmttrig  VARCHAR2);
FUNCTION OG_Get_Fmttrig
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
fmttrig	Is the object's format trigger. This property applies only to chart elements.

Format Trigger Property Examples

```
/*The following procedure reads the specified
**format trigger from an object, and assigns a
**different format trigger to it.
*/

*/PROCEDURE GenFmtTrigger IS
  rect og_object;
  fmttrig varchar2(20);
BEGIN
  rect := og_get_object('rect');
  fmttrig := og_get_fmttrig(rect);
  og_set_fmttrig(rect, 'fmttrig1');
END;
```

Hide Object Property

Description Hides the object.

Syntax

```
PROCEDURE OG_Set_Hide
  (object  OG_Object)
   hide BOOLEAN);
FUNCTION OG_Get_Hide
  (object  OG_Object)
RETURN BOOLEAN;
```

Parameters

object	Is the object being described.
hide	Hides the object.

Hide Object Property Examples

```
/*The following button
**procedure hides or
**shows an object as it
**is selected.
*/
PROCEDURE OGBUTTONPROC0 (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
val boolean;
BEGIN
val := og_get_hide(hitobj);
if val then
   og_set_hide(hitobj, false);
 og_set_bfcolor(hitobj, 'red');
else
   og_set_hide(hitobj, true);
 og_set_bfcolor(hitobj, 'red');
end if;
END;
```

Inner Bounding Box Property

Description Is the object's inner bounding box. This is the rectangle that constitutes the object's ideal shape (i.e., connects the object's four control points), regardless of edge thickness or other property settings.

Syntax

```
FUNCTION OG_Get_Ibbox
  (object  OG_Object)
RETURN OG_Rectangle;
```

Parameters

object	Is the object being described.

Inner Bounding Box Property Examples

```
/*The following reads the dimensions
**of the inner bounding and outer
**bounding boxes and calculates
**the size of the actual bounding box.
*/
PROCEDURE GenIOBox IS
  obj og_object;
  ibox og_rectangle;
  obox og_rectangle;
  num number;
BEGIN
  obj := og_get_object('rect');
  ibox := og_get_ibbox(obj);
  obox := og_get_obbox(obj);
  num := (obox.height * obox.width)-(ibox.height*ibox.width);
END;
```

Name Property

Description Is the object's name.

Syntax

```
PROCEDURE OG_Set_Name
  (object       OG_Object,
   name         VARCHAR2
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Name
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
name	Is the object's name.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Name Property Examples

```
/*The following procedure reads
**the name of the object and assigns
**another name to it.
*/
PROCEDURE GenName IS
  obj og_object;
  name varchar2(20);
BEGIN
  obj := og_get_object('circle');
  name := og_get_name(obj);
  og_set_name(obj, 'teresa');
END;
```

Object Type Property

Description Is the object's type. The value of this property may be one of the following built-in constants:

OG_Arc_Objtype
OG_Chart_Objtype
OG_Group_Objtype
OG_Image_Objtype
OG_Line_Objtype
OG_Poly_Objtype
OG_Rect_Objtype
OG_Rrect_Objtype
OG_Symbol_Objtype
OG_Text_Objtype

Syntax

```
FUNCTION OG_Get_Objtype
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.

Object Type Property Examples

```
/*The following button procedure checks
**the type of object being selected by
**the mouse, and prints the type name to
**a text object.
*/
PROCEDURE GenObjType2 (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  text og_object;
  objtype number;

BEGIN
  text := og_get_object('text object');
  objtype := og_get_objtype(hitobj);
  if objtype = og_arc_objtype then
     og_set_str(text, 'arc');
  elsif objtype = og_chart_objtype then
     og_set_str(text, 'chart');
  elsif objtype = og_group_objtype then
     og_set_str(text, 'group');
  elsif objtype = og_image_objtype then
     og_set_str(text, 'image');
  elsif objtype = og_line_objtype then
     og_set_str(text, 'line');
  elsif objtype = og_poly_objtype then
     og_set_str(text, 'poly');
  elsif objtype = og_rect_objtype then
     og_set_str(text, 'rect');
  elsif objtype = og_rrect_objtype then
     og_set_str(text, 'rrect');
  elsif objtype = og_symbol_objtype then
     og_set_str(text, 'symbol');
  elsif objtype = og_text_objtype then
     og_set_str(text, 'text');
  end if;
END;
```

Outer Bounding Box Property

Description Is the object's outer bounding box. This is the smallest rectangle that completely surrounds the object. This may differ from the inner bounding box if the object has a thick edge. While the inner bounding box traces only the ideal shape of the object, the outer bounding box surrounds the entire object.

Syntax

```
FUNCTION OG_Get_Obbox
  (object  OG_Object)
RETURN OG_Rectangle;
```

Parameters

object	Is the object being described.

Outer Bounding Box Property Examples

```
/*The following reads the dimensions of the
**inner bounding and outer bounding boxes and
**calculates the size of the actual bounding box.
*/
PROCEDURE GenIOBox IS
  obj og_object;
  ibox og_rectangle;
  obox og_rectangle;
  num number;
BEGIN
  obj := og_get_object('rect');
  ibox := og_get_ibbox(obj);
  obox := og_get_obbox(obj);
  num := (obox.height * obox.width)-(ibox.height*ibox.width);
END;
```

Parent Property

Description Is the handle to the object's parent object.

Syntax

```
FUNCTION OG_Get_Parent
  (object  OG_Object)
RETURN OG_Object;
```

Parameters

object	Is the object being described.

Parent Property Examples

```
/*The following procedure gets the
**parent of the current object, and
**prints the name of the parent object
**to a text object.
*/
PROCEDURE GenParent IS
  text og_object;
  obj og_object;
  parent og_object;
  name varchar2(20);
BEGIN
text := og_get_object('text object');
  obj := og_get_object('circle');
  parent := og_get_parent(obj);
  name := og_get_name(parent);
  og_set_str(text, name);
END;
```

Set Parameter Property

Description Is the parameter whose value is set when the object is selected.

Syntax

```
PROCEDURE OG_Set_Setparam
  (object   OG_Object,
  setparam  VARCHAR2);
FUNCTION OG_Get_Setparam
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
setparam	Is the parameter whose value is set when the object is selected.

Set Parameter Property Examples

```
/*The following procedure reads the
**parameter of a rectangle object, and
**assigns a new parameter to it.
*/
PROCEDURE SetParam IS
  rect og_object;
  param varchar2(20);
BEGIN
  rect := og_get_object('rect');
  param := og_get_setparam(rect);
  og_set_setparam(rect, 'PARAM1');
END;
```

Graphic Properties

Background Edge Color Property
Background Fill Color Property
Bevel Style Property
Cap Style Property
Dash Style Property
Edge Pattern Property
Edge Width Property
Fill Pattern Property
Foreground Edge Color Property
Foreground Fill Color Property
Join Style Property
Rotation Angle Property
Transfer Mode Property

Background Edge Color Property

Description Is the object's background edge color.

Syntax

```
PROCEDURE OG_Set_Becolor
  (object       OG_Object,
   becolor      VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Becolor
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
becolor	Is the object's background edge color.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Background Edge Color Property Examples

```
/* /*The following procedure swaps the foreground
**and background edge colors.
*/
PROCEDURE FBEdgeColor IS
  obj og_object;
  fcolor varchar2(20);
  bcolor varchar2(20);
BEGIN
  obj := og_get_object('rect');
  fcolor := og_get_fecolor(obj);
  bcolor := og_get_becolor(obj);
  og_set_fecolor(obj, bcolor);
  og_set_becolor(obj, fcolor);
END;
```

Background Fill Color Property

Description Is the object's background fill color.

Syntax

```
PROCEDURE OG_Set_Bfcolor
  (object       OG_Object,
   bfcolor      VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Bfcolor
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
bfcolor	Is the object's background fill color.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Background Fill Color Property Examples

```
/*The following procedure swaps the foreground
**and background fill colors.
*/
PROCEDURE FBFillColor IS
  obj og_object;
  fcolor varchar2(20);
  bcolor varchar2(20);
BEGIN
  obj := og_get_object('rect');
  fcolor := og_get_ffcolor(obj);
  bcolor := og_get_bfcolor(obj);
  og_set_ffcolor(obj, bcolor);
  og_set_bfcolor(obj, fcolor);
END;
```

Bevel Style Property

Description Is the object's bevel style. The value of this property may be one of the following built-in constants:

OG_Inset_Bstyle

OG_Lowered_Bstyle

OG_Outset_Bstyle

OG_Plain_Bstyle

OG_Raised_Bstyle

Syntax

```
PROCEDURE OG_Set_Bevelstyle
  (object       OG_Object,
   bevelstyle   NUMBER,
   damage       BOOLEAN    := TRUE,
   update_bbox  BOOLEAN    := TRUE);
FUNCTION OG_Get_Bevelstyle
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
bevelstyle	Is the object's bevel style.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Bevel Style Property Examples

```
/*The following button procedure rotates
**the bevel style of a selected object.
*/
PROCEDURE bevel (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  obj og_object;
  num number;
BEGIN
  obj := og_get_object('rect');
  num := og_get_bevelstyle(obj);
  if num = og_inset_bstyle then
     og_set_bevelstyle(obj, og_lowered_bstyle);
  elsif num = og_lowered_bstyle then
     og_set_bevelstyle(obj, og_outset_bstyle);
  elsif num = og_outset_bstyle then
     og_set_bevelstyle(obj, og_plain_bstyle);
  elsif num = og_plain_bstyle then
     og_set_bevelstyle(obj, og_raised_bstyle);
  elsif num = og_raised_bstyle then
     og_set_bevelstyle(obj, og_inset_bstyle);
  end if;
END;
```

Cap Style Property

Description Is the cap style of the object's edge. The value of this property may be one of the following built-in constants:

OG_Butt_Cstyle

OG_Projecting_Cstyle

OG_Round_Cstyle

Syntax

```
PROCEDURE OG_Set_Capstyle
  (object        OG_Object,
   capstyle      NUMBER,
   damage        BOOLEAN    :=  TRUE,
   update_bbox   BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Capstyle
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
capstyle	Is the cap style of the object's edge.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Cap Style Property Examples

```
/*The following button procedure rotates
**the cap style of an object's edge.
*/
PROCEDURE CapStyle (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  num number;
BEGIN
  num := og_get_capstyle(hitobj);
  if num = og_butt_cstyle then
     og_set_capstyle(hitobj, og_projecting_cstyle);
  elsif num = og_projecting_cstyle then
     og_set_capstyle(hitobj,og_round_cstyle);
  elsif num = og_round_cstyle then
     og_set_capstyle(hitobj,og_butt_cstyle);
  end if;
END;
```

Dash Style Property

Description Is the dash style of the object's edge. The value of this property may be one of the following built-in constants:

OG_Solid_Dstyle
OG_Dot_Dstyle
OG_Long_Dstyle
OG_Dashdot_Dstyle
OG_Dotdot_Dstyle
OG_Short_Dstyle
OG_Dashdotdot_Dstyle

Syntax

```
PROCEDURE OG_Set_Dashstyle
  (object         OG_Object,
   dashstyle      NUMBER,
   damage         BOOLEAN    :=  TRUE,
   update_bbox    BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Dashstyle
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
dashstyle	Is the dash style of the object's edge.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Dash Style Property Examples

```
*/The following button procedure rotates
**the dash style on hit object.
*/
PROCEDURE DashStyle (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
    num number;
BEGIN
  num := og_get_dashstyle(hitobj);
  if num = og_solid_dstyle then
     og_set_dashstyle(hitobj, og_dot_dstyle);
  elsif num = og_dot_dstyle then
     og_set_dashstyle(hitobj,og_long_dstyle);
  elsif num = og_long_dstyle then
     og_set_dashstyle(hitobj,og_dashdot_dstyle);
  elsif num = og_dashdot_dstyle then
     og_set_dashstyle(hitobj,og_dotdot_dstyle);
  elsif num = og_dotdot_dstyle then
     og_set_dashstyle(hitobj,og_short_dstyle);
  elsif num = og_short_dstyle then
     og_set_dashstyle(hitobj,og_dashdotdot_dstyle);
  elsif num = og_dashdotdot_dstyle then
     og_set_dashstyle(hitobj,og_solid_dstyle);
  end if;
END;
```

Edge Pattern Property

Description Is the object's edge pattern.

Syntax

```
PROCEDURE OG_Set_Edgepatt
  (object        OG_Object,
   edgepatt      VARCHAR2,
   damage        BOOLEAN    :=  TRUE,
   update_bbox   BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Edgepatt
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
edgepatt	Is the object's edge pattern.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Edge Pattern Property Examples

```
/*The following procedure swaps the edge
**and fill patterns of an object.
*/
PROCEDURE EdgePattern IS
  obj og_object;
  edgepatt varchar2(20);
  fillpatt varchar2(20);
BEGIN
  obj := og_get_object('rect');
  edgepatt := og_get_edgepatt(obj);
  fillpatt := og_get_fillpatt(obj);
  og_set_edgepatt(obj, fillpatt);
  og_set_fillpatt(obj, edgepatt);
END;
```

Edge Width Property

Description Is the width of the object's edge (in layout units).

Syntax

```
PROCEDURE OG_Set_Ewidth
  (object       OG_Object,
   ewidth       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Ewidth
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
ewidth	Is the width of the object's edge (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Edge Width Property Examples

```
/*The following procedure reads the edge
**width of a selected object.  If the width
**is 0, it resets the width to value 800.
*/
PROCEDURE EdgeWidth IS
  obj og_object;
  width number;
BEGIN
  obj := og_get_object('rect');
  width := og_get_ewidth(obj);
  if width = 0 then
    og_set_ewidth(obj, 800);
  end if;
END;
```

Fill Pattern Property

Description Is the object's fill pattern.

Syntax

```
PROCEDURE OG_Set_Fillpatt
  (object       OG_Object,
   fillpatt     VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Fillpatt
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
fillpatt	Is the object's fill pattern.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Fill Pattern Property Examples

```
/*The following procedure swaps the edge
**and fill patterns of an object.
*/
PROCEDURE EdgePattern IS
  obj og_object;
  edgepatt varchar2(20);
  fillpatt varchar2(20);
BEGIN
  obj := og_get_object('rect');
  edgepatt := og_get_edgepatt(obj);
  fillpatt := og_get_fillpatt(obj);
  og_set_edgepatt(obj, fillpatt);
  og_set_fillpatt(obj, edgepatt);
END;
```

Foreground Edge Color Property

Description Is the object's foreground edge color.

Syntax

```
PROCEDURE OG_Set_Fecolor
  (object       OG_Object,
   fecolor      VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Fecolor
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
fecolor	Is the object's foreground edge color.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Foreground Edge Color Property Examples

```
/*The following procedure swaps the foreground
**and background edge colors.
*/
PROCEDURE FBEdgeColor IS
  obj og_object;
  fcolor varchar2(20);
  bcolor varchar2(20);
BEGIN
  obj := og_get_object('rect');
  fcolor := og_get_fecolor(obj);
  bcolor := og_get_becolor(obj);
  og_set_fecolor(obj, bcolor);
  og_set_becolor(obj, fcolor);
END;
```

Foreground Fill Color Property

Description Is the object's foreground fill color.

Syntax

```
PROCEDURE OG_Set_Ffcolor
  (object       OG_Object,
   ffcolor      VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Ffcolor
  (object  OG_Object)
RETURN VARCHAR2;
```

Parameters

object	Is the object being described.
ffcolor	Is the object's foreground fill color.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Foreground Fill Color Property Examples

```
/*The following procedure swaps the foreground
**and background fill colors.
*/
PROCEDURE FBFillColor IS
  obj og_object;
  fcolor varchar2(20);
  bcolor varchar2(20);
BEGIN
  obj := og_get_object('rect');
  fcolor := og_get_ffcolor(obj);
  bcolor := og_get_bfcolor(obj);
  og_set_ffcolor(obj, bcolor);
  og_set_bfcolor(obj, fcolor);
END;
```

Join Style Property

Description Is the join style of the object's edge. The value of this property may be one of the following built-in constants:

OG_Mitre_Jstyle
OG_Bevel_Jstyle
OG_Round_Jstyle

Syntax

```
PROCEDURE OG_Set_Joinstyle
  (object       OG_Object,
   joinstyle    NUMBER,
   damage         BOOLEAN   := TRUE,
   update_bbox    BOOLEAN   := TRUE);

FUNCTION OG_Get_Joinstyle
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
joinstyle	Is the join style of the object's edge.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Join Style Property Examples

```
/*The following button procedure rotates
**the join style of an object's edge.
*/
PROCEDURE JoinStyle (mitreonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  num number;
BEGIN
  num := og_get_joinstyle(hitobj);
  if num = og_mitre_jstyle then
     og_set_joinstyle(hitobj, og_bevel_jstyle);
  elsif num = og_bevel_jstyle then
     og_set_joinstyle(hitobj,og_round_jstyle);
  elsif num = og_round_jstyle then
     og_set_joinstyle(hitobj,og_mitre_jstyle);
  end if;
END;
```

Rotation Angle Property

Description Is the object's rotation angle. The angle at which the object is initially created is considered to be 0, and this property is the number of degrees clockwise the object currently differs from that initial angle. You can rotate an object to an absolute angle by setting this property, or use the OG_Rotate procedure to rotate an object by a relative amount. (Note that when you use OG_Rotate to rotate an object, the *Rotation Angle* property will automatically be updated to reflect the new absolute angle.)

Syntax

```
PROCEDURE OG_Set_Rotang
  (object       OG_Object,
   rotang       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Rotang
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
rotang	Is the object's rotation angle. The angle at which the object is initially created is considered to be 0, and this property is the number of degrees clockwise the object currently differs from that initial angle.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Rotation Angle Property Examples

```
/*The following procedure reads the rotation
**angle from a selected object, and rotates
**the object another 45 degrees to the right.
*/
PROCEDURE RotAngle IS
  obj og_object;
  rotang number;
BEGIN
  obj := og_get_object('rect');
  rotang := og_get_rotang(obj);
  og_set_rotang(obj, rotang+45);
END;
```

Transfer Mode Property

Description Is the object's transfer mode. The value of this property may be one of the following built-in constants:

OG_Copy_Transfer
OG_Revcopy_Transfer
OG_Or_Transfer
OG_Revor_Transfer
OG_Clear_Transfer
OG_Revclear_Transfer
OG_Invert_Transfer
OG_Backinvert_Transfer

Syntax

```
PROCEDURE OG_Set_Transfer
  (object        OG_Object,
   transfer      NUMBER,
   damage        BOOLEAN    := TRUE,
   update_bbox   BOOLEAN    := TRUE);
FUNCTION OG_Get_Transfer
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.
transfer	Is the object's transfer mode.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Transfer Mode Property Examples

```
/*The following button procedure rotates the
**transfer mode of a selected object.
*/
PROCEDURE transher (copyonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  num number;
BEGIN
  num := og_get_transfer(hitobj);
  if num = og_copy_transfer then
     og_set_transfer(hitobj, og_revcopy_transfer);
  elsif num = og_revcopy_transfer then
     og_set_transfer(hitobj,og_or_transfer);
  elsif num = og_or_transfer then
     og_set_transfer(hitobj,og_revor_transfer);
  elsif num = og_revor_transfer then
     og_set_transfer(hitobj,og_clear_transfer);
  elsif num = og_clear_transfer then
     og_set_transfer(hitobj,og_revclear_transfer);
  elsif num = og_revclear_transfer then
     og_set_transfer(hitobj,og_invert_transfer);
  elsif num = og_invert_transfer then
     og_set_transfer(hitobj,og_backinvert_transfer);
  elsif num = og_backinvert_transfer then
     og_set_transfer(hitobj,og_copy_transfer);
  end if;
END;
```

Group Properties

Child Count Property
Clip Flag Property

Child Count Property

Description Is the number of children that belong to the group object. If another group object is a child of the group being checked, that object will be counted only as one object.

Syntax

```
FUNCTION OG_Get_Childcount
  (object  OG_Object)
RETURN NUMBER;
```

Parameters

object	Is the object being described.

Child Count Property Examples

```
/*The following procedure gets the number
**of children in a group object.
*/
PROCEDURE GrpChildCnt IS
  grp og_object;
  cnt number;
BEGIN
  grp := og_get_object('group');
  cnt := og_get_childcount(grp);
END;
```

Clip Flag Property

Description Specifies whether the first object in the group is a rectangle object that should be used as a clipping rectangle. If TRUE, the only members of the group that appear on the layout are those objects-or portions of those objects-that appear within the bounds of the clipping rectangle. The rectangle object itself also appears.

Syntax

```
PROCEDURE OG_Set_Clipflag
  (object        OG_Object,
   clipflag      BOOLEAN,
   damage        BOOLEAN    :=  TRUE,
   update_bbox   BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Clipflag
  (object  OG_Object)
RETURN BOOLEAN;
```

Parameters

object	Is the object being described.
clipflag	Specifies whether the first object in the group is a rectangle object that should be used as a clipping rectangle.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Clip Flag Property Examples

```
/*The following procedure checks if
**clipflag is true.  If not, it sets the
**flag to true.
*/
PROCEDURE GrpClipFlg IS
  grp og_object;
  flag boolean;
BEGIN
  grp := og_get_object('group');
  flag := og_get_clipflag(grp);
  if flag = false then
     og_set_clipflag(grp, true);
  end if;
END;
```

Image Properties

Clip Rectangle Property
Dither Property
Height Property
Position Property
Quality Property
Width Property

Clip Rectangle Property

Description Is the x- and y-coordinates, height, and width of the image's clipping rectangle (in layout units). Only the portion of the image that falls within this clipping rectangle will be displayed. If this property is not specified, the clipping rectangle will equal the full dimensions of the image.

Syntax

```
PROCEDURE OG_Set_Cliprect
  (image         OG_Object,
   cliprect      OG_Rectangle,
   damage        BOOLEAN       :=  TRUE,
   update_bbox   BOOLEAN       :=  TRUE);
FUNCTION OG_Get_Cliprect
  (image  OG_Object)
RETURN OG_Rectangle;
```

Parameters

image	Is the image object being described.
cliprect	Is the x- and y-coordinates, height, and width of the image's clipping rectangle (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Clip Rectangle Property Examples

```
/*The following procedure reduces the
**size of the clipping rectangle by half.
*/
PROCEDURE ClipRect IS
  image og_object;
  rect og_rectangle;
BEGIN
  image := og_get_object('image');
  rect := og_get_cliprect(image);
  rect.height := rect.height/2;
  rect.width := rect.width/2;
  og_set_cliprect(image, rect);
  og_set_clipflag(image, true);
END;
```

Dither Property

Description Specifies whether Graphics Builder dithers the image when displaying it. The value of this property may be one of the following:

Syntax

```
PROCEDURE OG_Set_Image_Dither
  (image   OG_Object,
   dither  BOOLEAN);
FUNCTION OG_Get_Image_Dither
  (image  OG_Object)
RETURN BOOLEAN;
```

Parameters

image	Is the image object being described.
dither	Specifies whether Graphics Builder dithers the image when displaying it.

Dither Property Examples

```
/*The following button procedure
**dithers an image or removes
**dithering.
*/
PROCEDURE SetDither (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
val boolean;
image og_object;
BEGIN
  image := og_get_object('image');
  val := og_get_image_dither(image);
  if val then
    og_set_image_dither(og_get_object('image'), false);
  else
    og_set_image_dither(og_get_object('image'), true);
  end if;
END;
```

Height Property

Description Is the image's height (in layout units). If you set this property to some value other than the image's default height, the image will be scaled to fit within the new height.

Syntax

(See OG_Set_Image_Size, above.)

```
FUNCTION OG_Get_Image_Height
  (image  OG_Object)
RETURN NUMBER;
```

Parameters

image	Is the image object being described.

Height Property Examples

```
/*The following procedure reduces
**an image's size by half.
*/
PROCEDURE SizeWidthHeight IS
  image og_object;
  height number;
  width number;
BEGIN
  image := og_get_object('image');
  width := og_get_image_width(image);
  height := og_get_image_height(image);
  og_set_image_size(image, width/2, height/2);
END;
```

Position Property

Description Is the x- and y-coordinates of the image's upper-left corner (in layout units).

Syntax

```
PROCEDURE OG_Set_Upperleft
  (image          OG_Object,
   upperleft      OG_Point,
   damage         BOOLEAN    :=  TRUE,
   update_bbox    BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Upperleft
  (image  OG_Object)
RETURN OG_Point;
```

Parameters

image	Is the image object being described.
upperleft	Is the x- and y-coordinates of the image's upper-left corner (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Position Property Examples

```
/*The following procedure reads the
**(x,y) coordinates of the image's
**upper-left corner.  If the coordinate
**is not (0,0), the procedure
**moves the image’s upper-left
**corner to the (0,0) coordinate.
*/
PROCEDURE Position IS
  image og_object;
  pos og_point;
BEGIN
  image := og_get_object('image');
  pos := og_get_upperleft(image);
  if pos.x != 0 and pos.y != 0 then
    pos.x := 0;
    pos.y := 0;
    og_set_upperleft(image, pos);
  end if;
END;
```

Quality Property

Description Specifies with what quality the image is drawn. Higher quality images look better, but require more processing time to manipulate (e.g., draw, move, scale, etc.). The value of this property may be one of the following built-in constants:

OG_High_Iquality

OG_Medium_Iquality

OG_Low_Iquality

Syntax

```
PROCEDURE OG_Set_Image_Quality
  (image    OG_Object,
   quality  NUMBER);
FUNCTION OG_Get_Image_Quality
  (image  OG_Object)
RETURN NUMBER;
```

Parameters

image	Is the image object being described.
quality	Specifies with what quality the image is drawn.

Quality Property Examples

```
/*The following procedure checks image
**quality.  If the image is currently drawn
**with high quality, the procedure redraws
**it with low quality.
*/
PROCEDURE GetQuality (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  image og_object;
  qty number;
BEGIN
  image := og_get_object('image');
  qty := og_get_image_quality(image);
  if qty = og_high_iquality then
    og_set_image_quality(image, og_low_iquality);
  end if;
END;
```

Width Property

Description Is the image's width (in layout units). If you set this property to some value other than the image's default width, the image will be scaled to fit within the new width.

Syntax

```
PROCEDURE OG_Set_Image_Size
  (image        OG_Object,
   width        NUMBER,
   height       NUMBER,
   damage       BOOLEAN    := TRUE,
   update_bbox  BOOLEAN    :=   TRUE);
FUNCTION OG_Get_Image_Width
  (image  OG_Object)
RETURN NUMBER;
```

Parameters

image	Is the image object being described.
width	Is the image's width (in layout units).
height	Is the image’s height (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Width Property Examples

```
/* The following procedure reduces
**an image's size by half.
*/
PROCEDURE SizeWidthHeight IS
  image og_object;
  height number;
  width number;
BEGIN
  image := og_get_object('image');
  width := og_get_image_width(image);
  height := og_get_image_height(image);
  og_set_image_size(image, width/2, height/2);
END;
```

Line Properties

Arrow Style Property
End Point Property
Start Point Property

Arrow Style Property

Description Is the line's arrow style. The value of this property may be one of the following built-in constants:

OG_Noarrow_Astyle Means the line has no arrow.
OG_Start_Astyle Means the line has an arrow at its starting point.
OG_End_Astyle Means the line has an arrow at its end point.
OG_Both_Astyle Means the line has an arrow at both ends.
OG_Midtostart_Astyle Means the line has an arrow at its middle, pointing toward its starting point.
OG_Midtoend_Astyle Means the line has an arrow at its middle, pointing toward its end point.

Syntax

```
PROCEDURE OG_Set_Arrowstyle
  (line         OG_Object,
   arrowstyle   NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Arrowstyle
  (line  OG_Object)
RETURN NUMBER;
```

Parameters

line	Is the line object being described.
arrowstyle	Is the line's arrow style.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Arrow Style Property Examples

```
/*The following procedure determines a
*line's current arrow style.  If the line
**does not include arrows, the procedure adds
**arrows to both ends of the line.  If the
**line does include arrows, the
**procedure removes them.
*/
PROCEDURE Arrow (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  arrow og_object;
  num number;
BEGIN
  arrow := og_get_object('arrow');
  num := og_get_arrowstyle(arrow);
  if num = og_noarrow_astyle then
    og_set_arrowstyle(arrow, og_both_astyle);
  else
    og_set_arrowstyle(arrow, og_noarrow_astyle);
  end if;
END;
```

End Point Property

Description Is the x- and y-coordinates of the line's end point (in layout units).

Syntax

```
PROCEDURE OG_Set_Endpt
  (line         OG_Object,
   endpt        OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Endpt
  (line  OG_Object)
RETURN OG_Point;
```

Parameters

line	Is the line object being described.
endpt	Is the x- and y-coordinates of the line's end point (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

End Point Property Examples

```
/*The following procedure reads the
**coordinates of the line's ending point.
**If the line does not end at the upper-left
**corner of the display, the procedure resets
**the end point to (0,0).
*/
PROCEDURE OGBUTTONPROC0 (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  arrow og_object;
  pos og_point;
BEGIN
  arrow := og_get_object('a');
  pos := og_get_endpt(arrow);
  if pos.x != 0 and pos.y != 0 then
    pos.x := 0;
    pos.y := 0;
    og_set_endpt(arrow, pos);
  end if;
END;
```

Start Point Property

Description Is the x- and y-coordinates of the line's starting point (in layout units).

Syntax

```
PROCEDURE OG_Set_Startpt
  (line         OG_Object,
   startpt      OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Startpt(line OG_Object)
RETURN OG_Point;
```

Parameters

line	Is the line object being described.
startpt	Is the x- and y-coordinates of the line's starting point (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Start Point Property Examples

```
/* /*The following procedure reads
**the coordinates of a line's
**starting point.  If the line does
**not start from the upper-left corner
**of the display, the procedure resets
**the start point to (0,0).
*/
PROCEDURE  StartPt (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  arrow og_object;
  pos og_point;
BEGIN
  arrow := og_get_object('a');
  pos := og_get_startpt(arrow);
  if pos.x != 0 and pos.y != 0 then
    pos.x := 0;
    pos.y := 0;
    og_set_startpt(arrow, pos);
  end if;
END;
```

Polygon Properties

Closure Property
Point Count Property

Closure Property

Description Is the closure of the polygon. The value of this property may be one of the following:

TRUE Means the polygon is closed.

FALSE Means the polygon is open.

Syntax

```
PROCEDURE OG_Set_Poly_Closed
  (poly         OG_Object,
   closed       BOOLEAN,
   damage       BOOLEAN    := TRUE,
   update_bbox  BOOLEAN    := TRUE);
FUNCTION OG_Get_Poly_Closed
  (poly  OG_Object)
RETURN BOOLEAN;
```

Parameters

poly	Is the polygon being described.
closed	Is the closure of the polygon.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Closure Property Examples

```
/*The following procedure determines
**whether a polygon is closed.
**If the polygon is open, the procedure
**closes it.
*/
PROCEDURE closure IS
  polygon og_object;
  val boolean;
BEGIN
  polygon := og_get_object('polygon');
  val := og_get_poly_closed(polygon);
  if val = false  then
     og_set_poly_closed(polygon, true);
  end if;
END;
```

Point Count Property

Description Is the number of points that compose the polygon object.

Syntax

```
FUNCTION OG_Get_Pointct
  (poly  OG_Object)
RETURN NUMBER;
```

Parameters

poly	Is the polygon being described.

Point Count Property Examples

```
/*The following procedure reads the
**number of points that compose the
**polygon object and prints the number
**to a text object.
*/
PROCEDURE PntCnt IS
  text og_object;
  polygon og_object;
  cnt number;
BEGIN
text := og_get_object('text object');
  polygon := og_get_object('polygon');
  cnt := og_get_pointct(polygon);
  og_set_str(text, cnt);
END;
```

Printer Properties Property

Copies Property
End Page Property
Landscape Property
Name Property
Page Size Property
Print File Property
Start Page Property

Copies Property

Description Is the number of copies to print.

Syntax

```
PROCEDURE OG_Set_Copies
  (copies  NUMBER);
FUNCTION OG_Get_Copies
RETURN NUMBER;
```

Parameters

copies	Is the number of copies to print.

Copies Property Examples

```
/*The following procedure reads
**the number of copies and adds two more
**copies to print.
*/
PROCEDURE PrinterCopies IS
  copies number;
BEGIN
  copies := og_get_copies;
  og_set_copies(copis+2);
END;
```

End Page Property

Description Is the last page to print.

Syntax

```
PROCEDURE OG_Set_Endpage
  (endpage  NUMBER);
FUNCTION OG_Get_Endpage
RETURN NUMBER;
```

Parameters

endpage	Is the last page to print.

End Page Property Examples

```
/*The following procedure reads the
**end page number and resets it to the
**original number plus two.
*/.
PROCEDURE PrinterEndPage IS
  ep number;
BEGIN
  ep := og_get_endpage;
  og_set_endpage(ep+2);
END;
```

Landscape Property

Description Specifies whether the display is printed in landscape or portrait mode.

Syntax

```
PROCEDURE OG_Set_Landscape
  (landscape  BOOLEAN);
FUNCTION OG_Get_Landscape
RETURN BOOLEAN;
```

Parameters

landscape	Specifies whether the display is printed in landscape or portrait mode.

Landscape Property Examples

```
/*The following procedure determines
**if the display is printed in landscape
**or portrait mode, and prints the mode
**type to a text object.
*/
PROCEDURE PrinterLandscape IS
  landscape boolean;
BEGIN
  landscape := og_get_landscape;
  if landscape then
     og_set_str(og_get_object('text object'), 'landscape');
  else
     og_set_str(og_get_object('text object'), 'portrait');
  end if;
END;
```

Name Property

Description Is the name of the current printer.

Syntax

```
PROCEDURE OG_Set_Printer_Name
  (name  VARCHAR2);
FUNCTION OG_Get_Printer_Name
RETURN VARCHAR2;
```

Parameters

name	Is the name of the current printer.

Name Property Examples

```
/*The following procedure sets the
**printer name and prints the name to
**a text object.
*/
PROCEDURE PrinterName IS
  name varchar2(30);
BEGIN
  name := og_get_printer_name;
  og_set_str(og_get_object('text object'), name);
END;
```

Page Size Property

Description Is the page size (in inches).

Syntax

```
PROCEDURE OG_Set_Pagesize
  (width   NUMBER,
  (height  NUMBER);
```

Parameters

width	Is the width of the page (in inches).
height	Is the height of the page (in inches).

Page Size Property Examples

```
/*The following procedure sets the
**page size.
*/
PROCEDURE PrinterPageSize IS
  height number := 10*og_inch;
  width number := 10*og_inch;
  printfile varchar2(20);
BEGIN
  og_set_pagesize(height, width);
END;
```

Print File Property

Description Is the name of the PostScript file to print to. If this property is NULL, the output is sent to the printer.

Syntax

```
PROCEDURE OG_Set_Printfile
  (filename  VARCHAR2);
FUNCTION OG_Get_Printfile
RETURN VARCHAR2;
```

Parameters

filename	Is the name of the PostScript file to print to. If this property is NULL, the output is sent to the printer.

Print File Property Examples

```
/*.The following procedure sets the
**PostScript file name and prints it
**to a text object.
*/
PROCEDURE PrinterPrintFile IS
  printfile varchar2(20);
BEGIN
  og_set_printfile('myfile');
  printfile := og_get_printfile;
  og_set_str(og_get_object('text object'), printfile);
END;
```

Start Page Property

Description Is the first page to print.

Syntax

```
PROCEDURE OG_Set_Startpage
  (startpage  NUMBER);
FUNCTION OG_Get_Startpage
RETURN NUMBER;
```

Parameters

startpage	Is the first page to print.

Start Page Property Examples

```
/*The following procedure reads the start
**page number and resets the page number
**to the original number plus two.
*/
PROCEDURE PrinterStartPage IS
  sp number;
BEGIN
  sp := og_get_startpage;
  og_set_startpage(sp+2);
END;
```

Query Properties

Cache Type Property
Custom Query Procedure Property
Date Format Property
Execute On Open Property
Execute On Timer Property
Maximum Rows Property
Name Property
Post-Query Trigger Procedure Property
Query Source Property
Query Type Property

Cache Type Property

Description Determines how the newly retrieved data from a query execution is treated. The value of this property may be one of the following built-in constants:
OG_Append_Cachetype Means all of the existing rows of data are retained, and the new rows of data are added to the bottom of the existing data set.
OG_Copy_Cachetype Means all of the data from the previous execution is copied to a special buffer, and the newly retrieved data replaces it.
OG_None_Cachetype Means all of the data from the previous execution is discarded, and the newly retrieved data replaces it.
Syntax

```
PROCEDURE OG_Set_Cachetype
  (query      OG_Query,
   cachetype  NUMBER);
FUNCTION OG_Get_Cachetype
  (query  OG_Query)
RETURN NUMBER;
```

Parameters

query	Is the query being described.
cachetype	Determines how the newly retrieved data from a query execution is treated.

Cache Type Property Examples

```
/*The following procedure rotates the query
**cache type of a query.
*/
PROCEDURE QryCacheType (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  qry og_query;
  num number;
BEGIN
  qry := og_get_query('query0');
  num := og_get_cachetype(qry);
  if num = og_append_cachetype then
     og_set_cachetype(qry, og_copy_cachetype);
  elsif num = og_copy_cachetype then
     og_set_cachetype(qry, og_none_cachetype);
  elsif num = og_none_cachetype then
     og_set_cachetype(qry, og_append_cachetype);
  end if;
END;
```

Custom Query Procedure Property

Description Is the PL/SQL procedure that is invoked when a Custom query is executed.

Syntax

```
PROCEDURE OG_Set_Customproc
  (query       OG_Query,
   customproc  VARCHAR2);
FUNCTION OG_Get_Customproc
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
customproc	Is the PL/SQL procedure that is invoked when a Custom query is executed.

Custom Query Procedure Property Examples

```
/*The following button procedure swaps the two PL/SQL
**procedures which are invoked when a custom query is
**executed.
*/
PROCEDURE CustQry (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  proc varchar2(20);
  qry og_query;
BEGIN
  qry := og_get_query('query0');
  proc := og_get_customproc(qry);
  if proc = 'CUSTQRY1' then
    og_set_customproc(qry, 'CUSTQRY2');
  elsif proc = 'CUSTQRY2' then
     og_set_customproc(qry, 'CUSTQRY1');
  end if;
  og_execute_query(qry);
END;
```

Date Format Property

Description Is the date format mask for the query.

Syntax

```
PROCEDURE OG_Set_Dateformat
  (query       OG_Query,
   dateformat  VARCHAR2);
FUNCTION OG_Get_Dateformat
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
dateformat	Is the date format mask for the query.

Date Format Property Examples

```
/*The following procedure reads and sets
**the Date Format mask for the query.
*/
PROCEDURE QueryDateFmt IS
  qry og_query;
  DateFmt varchar2(20);
BEGIN
  qry := og_get_query('query0');
  DateFmt := og_get_dateformat(qry);
  og_set_dateformat(qry, 'DD-MM-YYYY');
  DateFmt := og_get_dateformat(qry);
END;
```

Execute on Open Property

Description Specifies whether the query is automatically executed when the display is opened at runtime.

Syntax

```
PROCEDURE OG_Set_Execopen
  (query     OG_Query,
   execopen  BOOLEAN);
FUNCTION OG_Get_Execopen
  (query  OG_Query)
RETURN BOOLEAN;
```

Parameters

query	Is the query being described.
execopen	Specifies whether the query is automatically executed when the display is opened at runtime.

Execute on Open Property Examples

```
/*The following procedure checks if the Execute
**on Open checkbox is checked.  If it is checked,
**it unchecks it, or vice versa.
*/
PROCEDURE ExecOpen IS
  execOpen boolean;
  qry og_query;
BEGIN
  qry := og_get_query('query0');
  execOpen := og_get_execopen(qry);
  if execOpen then
     og_set_execopen(qry, false);
  else
     og_set_execopen(qry, true);
  end if;
END;
```

Execute on Timer Property

Description Is the name of the timer on which the query executes. If NULL, the query is not executed on a timer.

Syntax

```
PROCEDURE OG_Set_Exectimer
  (query      OG_Query,
   exectimer  VARCHAR2);
FUNCTION OG_Get_Exectimer
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
exectimer	Is the name of the timer on which the query executes. If NULL, the query is not executed on a timer.

Execute on Timer Property Examples

```
/*The following procedure reads the name of
**the timer on which the query executes and
**assigns a new timer to the query.
*/
PROCEDURE ExecTimer IS
  exectimer varchar2(20);
  qry og_query;
BEGIN
  qry := og_get_query('query0');
  exectimer := og_get_exectimer(qry);
  og_set_exectimer(qry, 'timer1');
END;
```

Maximum Rows Property

Description Specifies the maximum number of rows of data that are retained in the query's data set. If NULL, all rows are retained.

Syntax

```
PROCEDURE OG_Set_Maxrows
  (query    OG_Query,
   maxrows  NUMBER);
FUNCTION OG_Get_Maxrows
  (query  OG_Query)
RETURN NUMBER;
```

Parameters

query	Is the query being described.
maxrows	Specifies the maximum number of rows of data that are retained in the query's data set. If NULL, all rows are retained.

Maximum Rows Property Examples

```
/*The following procedure reads the maximum
**number of rows of data that are retained
**in the query's data set, and adds two rows to
**the original number.
*/
PROCEDURE QueryMaxRow IS
  qry og_query;
  num number;
BEGIN
  qry := og_get_query('query0');
  num := og_get_maxrows(qry);
  og_set_maxrows(qry, num+2);
END;
```

Maximum Rows Flag Property

Description Specifies whether a limit is placed on the number of rows contained in the data set. This is only used when the cachetype is of type OG_APPEND_CACHETYPE.

Syntax

```
PROCEDURE OG_Set_Maxflag
  (query    OG_Query,
   maxflag  BOOLEAN);
FUNCTION OG_Get_Maxflag
  (query  OG_Query)
RETURN BOOLEAN;
```

Parameters

query	Is the query being described.
maxflag	Specifies the maximum number of rows of data that can be contained in the query's data set.

Maximum Rows Flag Property Examples

```
/*The following procedure reads the maximum
**number of rows of data that are retained
**in the query's data set, and adds two rows to
**the original number.  If the incremented number
**is greater than 1024, then it disables the
**maximum rows flag, thus allowing the query to get
**all the rows of data.
*/
PROCEDURE MaxFlagToggle IS
  qry og_query;
  num number;
BEGIN
  qry := og_get_query('query0');
  num := og_get_maxrows(qry);
  num := num+2;
  og_set_maxrows(qry, num);
  IF ((num > 1024) AND (og_get_maxflag(qry)=TRUE)) THEN
    og_set_maxflag(qry,FALSE);
  END IF;
END;
```

Name Property

Description Is the name of the query.

Syntax

```
PROCEDURE OG_Set_Name
  (query  OG_Query,
   name   VARCHAR2);
FUNCTION OG_Get_Name
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
name	Is the name of the query.

Name Property Examples

```
/*The following procedure swaps
** the name of two queries.
*/
PROCEDURE QueryName IS
  qry0 og_query;
  qry1 og_query;
  name0 varchar2(30);
  name1 varchar2(30);
BEGIN
  qry0 := og_get_query('query0');
  qry1 := og_get_query('query1');
  name0 := og_get_name(qry0);
  name1 := og_get_name(qry1);
  og_set_name(qry1, 'tmp');
  og_set_name(qry0, name1);
  og_set_name(qry1, name0);
END;
```

Post-Query Trigger Procedure Property

Description Is the PL/SQL procedure that is invoked after the query is executed.

Syntax

```
PROCEDURE OG_Set_Postproc
  (query     OG_Query,
   postproc  VARCHAR2);
FUNCTION OG_Get_Postproc
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
postproc	Is the PL/SQL procedure that is invoked after the query is executed.

Post-Query Trigger Procedure Property Examples

```
/*The following button procedure swaps the two PL/SQL
**procedures which are invoked after the query is
**executed.
*/
PROCEDURE PostTrigger (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  proc varchar2(20);
  qry og_query;
BEGIN
  qry := og_get_query('query0');
  proc := og_get_postproc(qry);
  if  proc = 'POST1' then
      og_set_postproc(qry, 'POST2');
  elsif proc = 'POST2' then
      og_set_postproc(qry, 'POST1');
  end if;
  og_execute_query(qry);
END;
```

Query Source Property

Description Is the source of the query's data. If the data comes from a database, this property should contain the text of the query's SQL SELECT statement. If the data is stored in the filesystem, this property should contain the path and name of the data file.

Syntax

```
PROCEDURE OG_Set_Querysource
  (query        OG_Query,
   querysource  VARCHAR2);
FUNCTION OG_Get_Querysource
  (query  OG_Query)
RETURN VARCHAR2;
```

Parameters

query	Is the query being described.
querysource	Is the source of the query's data.

Query Source Property Examples

```
/*The following procedure swaps the source
**of two queries.
*/
PROCEDURE QuerySource IS
  qry0 og_query;
  qry1 og_query;
  source0 varchar2(50);
  source1 varchar2(50);
BEGIN
  qry0 := og_get_query('query0');
  qry1 := og_get_query('query1');
  source0:= og_get_querysource(qry0);
  source1:= og_get_querysource(qry1);
  og_set_querysource(qry0, source1);
  og_set_querysource(qry1, source0);
END;
```

Query Type Property

Description Is the type of query. The value of this property may be one of the following built-in constants:

OG_Custom_Qtype Means the query is a Custom query.

OG_Exsql_Qtype Means the query retrieves its data from a text file that contains a SQL SELECT statement.

OG_Prn_Qtype Means the query is based on a PRN file.

OG_Sql_Qtype Means the query is a SQL SE.LECT statement.

OG_Sylk_Qtype Means the query is based on a SYLK file.

OG_Wks_Qtype Means the query is based on a WKS file.

Syntax

```
PROCEDURE OG_Set_Querytype
  (query      OG_Query,
   querytype  NUMBER);
FUNCTION OG_Get_Querytype
  (query  OG_Query)
RETURN NUMBER;
```

Parameters

query	Is the query being described.
querytype	Is the type of query.

Query Type Property Examples

```
/*The following procedure rotates the
**query type of a query.
*/
PROCEDURE QryType (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  qry og_query;
  num number;
BEGIN
  qry := og_get_query('query0');
  num := og_get_querytype(qry);

  if num = og_custom_qtype then
     og_set_querytype(qry, og_exsql_qtype);
  elsif num = og_exsql_qtype then
     og_set_querytype(qry, og_prn_qtype);
  elsif num = og_prn_qtype then
     og_set_querytype(qry, og_sql_qtype);
  elsif num = og_sql_qtype then
     og_set_querytype(qry, og_sylk_qtype);
   elsif num = og_sylk_qtype then
     og_set_querytype(qry, og_wks_qtype);
 elsif num = og_wks_qtype then
     og_set_querytype(qry, og_custom_qtype);
  end if;
END;
```

Rectangle Properties Property

Base Rectangle Property

Base Rectangle Property

Description Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).

Syntax

```
PROCEDURE OG_Set_Rect_Baserect
  (rect         OG_Object,
   baserect     OG_Rectangle,
   damage       BOOLEAN       :=  TRUE,
   update_bbox  BOOLEAN       :=  TRUE);
FUNCTION OG_Get_Rect_Baserect
  (rect  OG_Object)
RETURN OG_Rectangle;
```

Parameters

rect	Is the rectangle object being described.
baserect	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Base Rectangle Property Examples

```
/*The following procedure determines the size
**of the rectangle base and doubles it.
*/
PROCEDURE baseRect IS
  rect og_rectangle;
  obj og_object;
BEGIN
  obj := og_get_object('rect');
  rect := og_get_rect_baserect(obj);
  rect.x := rect.x * 2;
  rect.y := rect.y * 2;
  rect.height := rect.height * 2;
  rect.width := rect.width * 2;
  og_set_rect_baserect(obj, rect);
END;);
  source0:= og_get_querysource(qry0);
  source1:= og_get_q?_
```

Reference Line Properties

Axis Property
Date Value Property
Label Property
Number Value Property

Axis Property

Description Specifies which axis the reference value is compared to determine its position. The value of this property may be one of the following built-in constants:

OG_X_Axis
OG_Y1_Axis
OG_Y2_Axis

Syntax

```
PROCEDURE OG_Set_Axis
  (refline  OG_Refline,
   axis     NUMBER);
FUNCTION OG_Get_Axis
  (refline  OG_Refline)
RETURN NUMBER;
```

Parameters

refline	Is the reference line being described.
axis	Specifies which axis the reference value is compared to determine its position.

Axis Property Examples

```
/*The following procedure maps
**the reference line against the
**Y1-axis if the line is currently
**mapped against Y2-axis.
*/
PROCEDURE Axis IS
  chart og_object;
  axis number;
  refline og_refline;
  template og_template;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  refline := og_get_refline(template, 0);
  axis := og_get_axis(refline);
  if axis = og_y2_axis then
    og_set_axis(refline, og_y1_axis);
  end if;
  og_update_chart(chart);
END;
```

Date Value Property

Description Is the date value at which the reference line appears.

Syntax

```
PROCEDURE OG_Set_Datevalue
  (refline    OG_Refline,
   datevalue  DATE);
FUNCTION OG_Get_Datevalue
  (refline  OG_Refline)
RETURN DATE;
```

Parameters

refline	Is the reference line being described.
datevalue	Is the date value at which the reference line appears.

Date Value Property Examples

```
/*The following procedure increases
**reference line value by 30 days.
*/
PROCEDURE DateVal IS
  chart og_object;
  dateval date;
  refline og_refline;
  template og_template;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  refline := og_get_refline(template, 0);
  dateval := og_get_datevalue(refline);
  og_set_datevalue(refline, dateval+30);
  og_update_chart(chart);
END;
```

Label Property

Description Is the text label that identifies the reference line in the legend.

Syntax

```
PROCEDURE OG_Set_Label
  (refline  OG_Refline,
   label    VARCHAR2);
FUNCTION OG_Get_Label
  (refline  OG_Refline)
RETURN VARCHAR2;
```

Parameters

refline	Is the reference line being described.
label	Is the text label that identifies the reference line in the legend.

Label Property Examples

```
/*The following procedure changes
**the reference line name to 'New Label'
**if this is not the current name of the
**label.
*/
PROCEDURE label IS
  chart og_object;
  label varchar2(20);
  refline og_refline;
  template og_template;
BEGIN
chart := og_get_object('chart');
  template := og_get_template(chart);
  refline := og_get_refline(template, 0);
  label := og_get_label(refline);
  if label != 'New Label' then
    og_set_label(refline, 'New label');
  end if;
  og_update_chart(chart);
END;
```

Number Value Property

Description Is the number value at which the reference line appears.

Syntax

```
PROCEDURE OG_Set_Numvalue
  (refline   OG_Refline,
   numvalue  NUMBER);
FUNCTION OG_Get_Numvalue
  (refline  OG_Refline)
RETURN NUMBER;
```

Parameters

refline	Is the reference line being described.
numvalue	Is the number value at which the reference line appears.

Number Value Property Examples

```
/*The following procedure increases reference
**line value by 500.
*/
PROCEDURE NumVal IS
  chart og_object;
  num number;
  refline og_refline;
  template og_template;
BEGIN
  chart := og_get_object('chart');
  template := og_get_template(chart);
  refline := og_get_refline(template, 0);
  num := og_get_numvalue(refline);
  og_set_numvalue(refline, num+500);
  og_update_chart(chart);
END;
```

Rounded Rectangle Properties

Base Rectangle Property
Corner Radii Property

Base Rectangle Property

Description Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).

Syntax

```
PROCEDURE OG_Set_Rrect_Baserect
  (rrect        OG_Object,
   baserect     OG_Rectangle,
   damage       BOOLEAN       :=  TRUE,
   update_bbox  BOOLEAN       :=  TRUE);
FUNCTION OG_Get_Rrect_Baserect
  (rrect  OG_Object)
RETURN OG_Rectangle;
```

Parameters

rrect	Is the rounded rectangle being described.
baserect	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Base Rectangle Property Examples

```
/*The following button procedure reduces
**the size of the base rectangle or the
**rounded rectangle.
*/
PROCEDURE baserect (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  brect og_rectangle;
  rrect og_object;
BEGIN
  rrect := og_get_object('rrect');
  brect := og_get_rrect_baserect(rrect);
  brect.x := brect.x/2;
  brect.y := brect.y/2;
  brect.height := brect.height/2;
  brect.width := brect.width/2;
  og_set_rrect_baserect(rrect, brect);
END;
```

Corner Radii Property

Description Is the x- and y-radii (in layout units) of the ellipse that would result if the arcs that form the rounded corners were continued to follow a full 360 degree path.

Syntax

```
PROCEDURE OG_Set_Corner
  (rrect        OG_Object,
   corner       OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Corner
  (rrect  OG_Object)
RETURN OG_Point;
```

Parameters

rrect	Is the rounded rectangle being described.
corner	Is the x- and y-radii (in layout units) of the ellipse that would result if the arcs that form the rounded corners were continued to follow a full 360 degree path.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Corner Radii Property Examples

```
/*The following button procedure doubles
**the x and y radii of the ellipse used to
**create the corners of a rounded rectangle
*/.
PROCEDURE corner (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  pt og_point;
  rrect og_object;
BEGIN
  rrect := og_get_object('rrect');
  pt := og_get_corner(rrect);
  pt.x := pt.x*2;
  pt.y := pt.y*2;
  og_set_corner(rrect, pt);
END;
```

Simple Text Properties

Color Property
Font Property
Text String Property

Color Property

Description Is the color in which the character string's text should be displayed. Note that this is the color for the text itself. To set the text object's edge or fill colors, change the text object's graphic properties.

Syntax

```
FUNCTION OG_Get_Color
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN VARCHAR2;
```

Parameters

text	Is the text object being described.
cmptext_index	Is the index number of the compound text element being described.
smptext_index	Is the index number of the simple text element being described.

Color Property Examples

```
/* The following procedure reads the color
**of the current text object.  If the
**current color is not red, it changes
**it to red.
*/
PROCEDURE color (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
color varchar2(20);
BEGIN
   color := og_get_color(hitobj, 0, 0);
   if color != 'red' then
     og_set_gcolor(hitobj, 'red');
   end if;
END;
```

Font Property

Description Is the font in which the character string's text is displayed.

Syntax

```
FUNCTION OG_Get_Font_Typeface
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN VARCHAR2;
FUNCTION OG_Get_Font_Ptsize
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN NUMBER;
FUNCTION OG_Get_Font_Style
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN NUMBER;
FUNCTION OG_Get_Font_Weight
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN NUMBER;
FUNCTION OG_Get_Font_Width
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN NUMBER;
FUNCTION OG_Get_Font_Kerning
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN BOOLEAN;
FUNCTION OG_Get_Font_Charset
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
cmptext_index	Is the index number of the compound text

	element being described.
smptext_index	Is the index number of the simple text element being described.

Font Property Examples

```
 */The following procedure reads
**the current typeface from the
**selected text object.  If the
**current style is not the same
**as the typeface from the argument,
**it assigns a new typeface to the
**text object.
*/
PROCEDURE fonttypeface (text og_object, typeface varchar2)IS
style varchar2(10);
BEGIN
  style := og_get_font_typeface(text, 0,0);
  if style != typeface then
    og_set_font_typeface(text, typeface);
  end if;
END;
```

Text String Property

Description Is the character string containing the actual text for the simple text element.

Syntax

```
PROCEDURE OG_Set_Str
  (text         OG_Object,
   str          VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Str
  (text           OG_Object,
   cmptext_index  NUMBER,
   smptext_index  NUMBER)
RETURN VARCHAR2;
```

Parameters

text	Is the text object being described.
str	Is the character string containing the actual text for the simple text element.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.
cmptext_index	Is the index number of the compound text element being described.
smptext_index	Is the index number of the simple text element being described.

Text String Property Examples

```
 /*The following procedure reads a text string from
**a display and appends numbers to it.
*/
PROCEDURE TextString IS
  text og_object;

BEGIN
  text := og_get_object('text object');
  og_set_str(text,og_get_str(text,0,0)||'123');
END;
```

Sound Properties Property

Name Property

Name Property

Description Is the name of the sound.

Syntax

```
PROCEDURE OG_Set_Name
  (sound  OG_Sound,
   name   VARCHAR2);
FUNCTION OG_Get_Name
  (sound  OG_Sound)
RETURN VARCHAR2;
```

Parameters

sound	Is the sound object being described.
name	Is the name of the sound.

Name Property Examples

```
/*The following procedure gets the
**name of sound from the sound handler
**and assigns a new name to it.
*/
PROCEDURE SoundName (sound in og_sound) IS
name varchar2(10);
BEGIN
  name := og_get_name(sound);
  og_set_name(sound, name||'2');
END;
```

Symbol Properties Property

Center Property
Index Property
Symbol Size Property

Center Property

Description Is the x- and y-coordinates of the symbol's center (in layout units).

Syntax

```
PROCEDURE OG_Set_Center
  (symbol       OG_Object,
   center       OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Center
  (symbol  OG_Object)
RETURN OG_Point;
```

Parameters

symbol	Is the symbol object being described.
center	Is the x- and y-coordinates of the symbol's center (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Center Property Examples

```
/*The following procedure moves
**the symbol from its original
**coordinate (x,y) to (x/2, y/2).
*/
PROCEDURE Center IS
  center og_point;
  symbol og_object;
BEGIN
  symbol := og_get_object('symbol');
  center := og_get_center(symbol);
  center.x := center.x/2;
  center.y := center.y/2;
  og_set_center(symbol, center);
END;
```

Index Property

Description Is the index (or number) of the symbol's position as it appears in the symbol palette in the Builder.

Syntax

```
PROCEDURE OG_Set_Indx
  (symbol       OG_Object,
   indx         NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Indx
  (symbol  OG_Object)
RETURN NUMBER;
```

Parameters

symbol	Is the symbol object being described.
indx	Is the index (or number) of the symbol's position as it appears in the symbol palette in the Builder.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Index Property Examples

```
 /*The following procedure gets the
**index of an object's symbol position
**in the symbol palette, and replaces
**the current symbol with the symbol
**which has an index value equal to the
**current index value + 1.
*/
PROCEDURE get_index IS
  sym_index number;
  symbol og_object;
BEGIN
  symbol := og_get_object('symbol');
  sym_index := og_get_indx(symbol);
  og_set_indx(symbol, sym_index+1);
END;
```

Symbol Size Property

Description Is the symbol's size. The value of this property may be one of the following built-in constants:

OG_Large_Symsize

OG_Medium_Symsize

OG_Small_Symsize

Syntax

```
PROCEDURE OG_Set_Symsize
  (symbol       OG_Object,
   symsize      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Symsize
  (symbol  OG_Object)
RETURN NUMBER;
```

Parameters

symbol	Is the symbol object being described.
symsize	Is the symbol's size.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Symbol Size Property Examples

```
 /*The following procedure reads a symbol's
**size.  If the symbol’s size is not LARGE,
**the procedure changes it to LARGE;
**if a symbol’s size is LARGE, the procedure
**changes it to small.
*/
PROCEDURE get_size IS
  sym_size number;
  symbol og_object;
BEGIN
  symbol := og_get_object('symbol');
  sym_size := og_get_symsize(symbol);
  if sym_size != og_large_symsize then
     og_set_symsize(symbol, og_large_symsize);
  else
     og_set_symsize(symbol, og_small_symsize);
  end if;
END;
```

Text Properties

Bounding Box Height Property
Bounding Box Width Property
Character Set Property
Color Property
Compound Text Count Property
Custom Spacing Property
Fixed Bounding Box Property
Horizontal Alignment Property
Horizontal Origin Property
Invisible Property
Kerning Property
Nearest Property
Origin Point Property
Point Size Property
Scalable Bounding Box Property
Scalable Font Property
Spacing Property
Style Property
Synthesize Property
Typeface Property
Vertical Alignment Property
Vertical Origin Property
Weight Property
Width Property
Wraparound Property

Bounding Box Height Property

Description Is the height of the bounding box (in layout units). Whenever the bounding box changes, this property will automatically be updated to reflect the new height. This property is used to set the height only if the *Fixed Bounding Box* property is TRUE.

Syntax

(See OG_Set_Text_Size.)

```
FUNCTION OG_Get_Text_Height
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.

Bounding Box Height Property Examples

```
/*The following procedure doubles the size of the
**text object's bounding box.
*/
PROCEDURE BBoxSize IS
  width number;
  height number;
  text og_object;
BEGIN
   text := og_get_object('text object');
   width := og_get_text_width(text);
   height := og_get_text_height(text);
   og_set_text_size(text, width*2, height*2);
END;
```

Bounding Box Width Property

Description Is the width of the bounding box (in layout units). Whenever the bounding box changes, this property will automatically be updated to reflect the new width. This property is used to set the width only if the *Fixed Bounding Box* property is TRUE.

Syntax

```
PROCEDURE OG_Set_Text_Size
  (text         OG_Object,
   width        NUMBER,
   height       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Text_Width
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
width	Is the width of the bounding box (in layout units).
height	Is the height of the bounding box (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Bounding Box Width Property Examples

```
 /* The following procedure doubles the size of the
**text object's bounding box.
*/
PROCEDURE BBoxSize IS
  width number;
  height number;
  text og_object;
BEGIN
   text := og_get_object('text object');
   width := og_get_text_width(text);
   height := og_get_text_height(text);
   og_set_text_size(text, width*2, height*2);
END;
```

Character Set Property

Description Is the font's character set. Values for this field specify character sets such as U.S. ASCII, Kanji, and Arabic. Not all character sets are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants:

OG_Us7ascii_Charset
OG_We8dec_Charset
OG_We8hp_Charset
OG_Us8pc437_Charset
OG_We8ebcdic37_Charset
OG_We8ebcdic500_Charset
OG_We8pc850_Charset
OG_D7dec_Charset
OG_F7dec_Charset
OG_S7dec_Charset
OG_E7dec_Charset
OG_Sf7ascii_Charset
OG_Ndk7dec_Charset
OG_I7dec_Charset
OG_Nl7dec_Charset
OG_Ch7dec_Charset
OG_Sf7dec_Charset
OG_We8iso8859p1_Charset
OG_Ee8iso8859p2_Charset
OG_Se8iso8859p3_Charset
OG_Nee8iso8859p4_Charset
OG_Cl8iso8859p5_Charset
OG_Ar8iso8859p6_Charset
OG_El8iso8859p7_Charset
OG_Iw8iso8859p8_Charset
OG_We8iso8859p9_Charset
OG_Ar8asmo708plus_Charset
OG_Ar7asmo449plus_Charset
OG_We8macroman8_Charset
OG_Jvms_Charset

OG_Jeuc_Charset
OG_Jdec_Charset
OG_Sjis_Charset
OG_Jdbcs_Charset
OG_Jhp_Charset
OG_Ksc5601_Charset
OG_Kibm5540_Charset
OG_Kdbcs_Charset
OG_Cgb231380_Charset
OG_Cdbcs_Charset
OG_Big5_Charset
OG_Cns1164386_Charset

Syntax

```
PROCEDURE OG_Set_Font_Charset
  (text         OG_Object,
   charset      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
charset	Is the font's character set.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Character Set Property Examples

```
 /*The following button procedure checks
**if the selected text object's font set
**is in US ASCII Character Set.  If not,
**it assigns ASCII Character Set to the object.
*/
PROCEDURE CharSet (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
setNo number;
BEGIN

     SetNo := OG_get_Font_charset(hitobj,0,0);
     if SetNo != og_US7ASCII_Charset then
        og_set_font_charset(hitobj, og_US7ASCII_charset);
     end if;
END;
```

Color Property

Description Is the text object's color.

Syntax

```
PROCEDURE OG_Set_Gcolor
  (text         OG_Object,
   gcolor       VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
gcolor	Is the text object's color.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Color Property Examples

```
 /*The following procedure reads
** the color of the current text object.
**If the current color is not red,
**it changes it to red.
*/
PROCEDURE color (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
color varchar2(20);
BEGIN
   color := og_get_color(hitobj, 0, 0);
   if color != 'red' then
     og_set_gcolor(hitobj, 'red');
   end if;
END;
```

Compound Text Count Property

Description Is the number of compound text elements that compose the text object.

Syntax

```
FUNCTION OG_Get_Ctcount
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.

Compound Text Count Property Examples

```
 /*The following procedure counts the number of
**compound text elements in a text object.
/*
PROCEDURE CompoundTextCnt IS
  num number;
  text og_object;
BEGIN
  text := og_get_object;
  num := og_get_ctcount(text);
END;
```

Custom Spacing Property

Description Is the custom spacing for the text object (in layout units). This property is used to specify spacing only if the *Spacing* property is set to custom spacing.

Syntax

```
PROCEDURE OG_Set_Custom
  (text         OG_Object,
   custom       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Custom
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
custom	Is the custom spacing for the text object (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Custom Spacing Property Examples

```
 /*The following procedure resets the
**custom spacing to twice its original setting.
*/
PROCEDURE CustomSpacing (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
num number;
BEGIN
   num := og_get_custom(hitobj);
   og_set_str(hitobj, 'abc'||num);
   og_set_custom(hitobj, num*2);
END;
```

Fixed Bounding Box Property

Description Specifies whether the text object's bounding box should remain a fixed size. If this property is TRUE, the values of the *Width* and *Height* properties should specify the size of the bounding box.

Syntax

```
PROCEDURE OG_Set_Fixed
  (text         OG_Object,
   fixed        BOOLEAN,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Fixed
  (text  OG_Object)
RETURN BOOLEAN;
```

Parameters

text	Is the text object being described.
fixed	Specifies whether the text object's bounding box should remain a fixed size.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Fixed Bounding Box Property Examples

```
 /*The following procedure checks if the text object's bounding box
**remains a fixed size.
*/
PROCEDURE FixBBox IS
  val boolean;
  text og_object;
BEGIN
   text := og_get_object('text object');
   val := og_get_fixed(text);
   if val  then
      og_set_fixed(text, false);
   else
      og_set_fixed(text, true);
   end if;
END;
```

Horizontal Alignment Property

Description Is the horizontal alignment of the text object. The value of this property may be one of the following built-in constants:

OG_Left_Halign
OG_Center_Halign
OG_Right_Halign

Syntax

```
PROCEDURE OG_Set_Halign
  (text         OG_Object,
   halign       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Halign
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
halign	Is the horizontal alignment of the text object.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Horizontal Alignment Property Examples

```
Horizontal Alignment
 /*The following procedure reads the horizontal
**alignment and readjusts it.
*/
PROCEDURE Halign IS
  num number:=og_right_halign;
  text og_object;
BEGIN
   text := og_get_object('text object');
   num := og_get_halign(text);
   if num = og_left_halign then
     og_set_halign(text, og_center_halign);
   elsif num = og_center_halign then
     og_set_halign(text, og_right_halign);
   elsif num = og_right_halign then
     og_set_halign(text, og_left_halign);
   end if;
END;
```

Horizontal Origin Property

Description Is the horizontal position of the text object relative to its origin point. The value of this property may be one of the following built-in constants:
OG_Left_Horigin Means the origin point lies along the left edge of the bounding box.
OG_Center_Horigin Means the origin point lies equally between the left and right edges of the boundingbox.
OG_Right_Horigin Means the origin point lies along the right edge of the bounding box.
Syntax

```
PROCEDURE OG_Set_Horigin
  (text        OG_Object,
   horigin     NUMBER,
   damage      BOOLEAN    :=  TRUE,
   update_bbox BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Horigin
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
horigin	Is the horizontal position of the text object relative to its origin point.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Horizontal Origin Property Examples

```
/*The following procedure reads the horizontal
**origin and readjusts it.
*/
PROCEDURE Horigin IS
  num number;
  text og_object;
BEGIN
   text := og_get_object('text object');
   num := og_get_horigin(text);
   if num = og_left_horigin then
     og_set_horigin(text, og_center_horigin);
   elsif num = og_center_horigin then
     og_set_horigin(text, og_rightr_horigin);
   elsif num = og_right_horigin then
     og_set_horigin(text, og_left_horigin);
   end if;
END;
```

Invisible Property

Description Specifies whether the text in the text object should be invisible. This is useful for text fields in which a user enters a password, if you don't want the password to be seen.

Syntax

```
PROCEDURE OG_Set_Invisible
  (text       OG_Object,
   invisible  BOOLEAN,
   damage         BOOLEAN    :=  TRUE,
   update_bbox    BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Invisible
  (text  OG_Object)
RETURN BOOLEAN;
```

Parameters

text	Is the text object being described.
invisible	Specifies whether the text in the text object should be invisible.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Invisible Property Examples

```
 /*The following procedure determines if
** text in a text object is invisible. If it
** is invisible it makes it visible; if it is
** visible it makes it invisible.
*/
PROCEDURE Invisible IS
  val boolean;
  text og_object;
BEGIN
   text := og_get_object('text object');
   val := og_get_invisible(text);
   if val  then
      og_set_invisible(text, false);
   else
      og_set_invisible(text, true);
   end if;
END;
```

Kerning Property

Description Specifies whether the font should be kerned. Kerning is the adjustment of the space between adjacent letters to improve the readability of the text.

Syntax

```
PROCEDURE OG_Set_Font_Kerning
  (text         OG_Object,
   kerning      BOOLEAN,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
kerning	Specifies whether the font should be kerned.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Kerning Property Examples

```
 /*The following button procedure turns font
**kerning on and off for a selected text object.
PROCEDURE kerning (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  val boolean;
BEGIN
  val := og_get_font_kerning(hitobj,0,0);
  if val then
     og_set_font_kerning(hitobj, false);
  else
     og_set_font_kerning(hitobj, true);
  end if;
END;
```

Nearest Property

Description Specifies whether Graphics Builder should substitute the nearest matching font if the exact font specified cannot be found. The precedence for finding the nearest font is typeface, point size, style, weight, and width (meaning that Graphics Builder first tries to find the specified typeface, then size, etc.).

Syntax

```
PROCEDURE OG_Set_Font_Nearest
  (text         OG_Object,
   nearest      BOOLEAN,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
nearest	Specifies whether Graphics Builder should substitute the nearest matching font if the exact font specified cannot be found.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Nearest Property Examples

```
 /*The following button procedure sets nearest
**properties to true.
*/
PROCEDURE nearest (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS

BEGIN
     OG_Set_Font_Nearest(hitobj, true);
END;
```

Origin Point Property

Description Is the x- and y-coordinates of the text object's upper-left corner (in layout units).

Syntax

```
PROCEDURE OG_Set_Origin
  (text         OG_Object,
   origin       OG_Point,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Origin
  (text  OG_Object)
RETURN OG_Point;
```

Parameters

text	Is the text object being described.
origin	Is the x- and y-coordinates of the text object's upper-left corner (in layout units).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Point Size Property

Description Is the font's point size. Values for this field are system-specific. For more information, consult your system administrator or your system documentation.

Syntax

```
PROCEDURE OG_Set_Font_Ptsize
  (text         OG_Object,
   ptsize       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
ptsize	Is the font's point size.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Point Size Property Examples

```
 /*The following procedure reads the point size of the
**current object and enlarges the text object to double
** its original size.
*/
PROCEDURE ptsize (text og_object)IS
num number;
BEGIN
  num := og_get_font_ptsize(text,0,0);
  og_set_font_ptsize(text, num*2);
END;
```

Scalable Bounding Box Property

Description Specifies whether the text object's bounding box should be scaled when the text object is scaled.

Syntax

```
PROCEDURE OG_Set_Bbscale
  (text         OG_Object,
   bbscale      BOOLEAN,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Bbscale
  (text  OG_Object)
RETURN BOOLEAN;
```

Parameters

text	Is the text object being described.
bbscale	Specifies whether the text object's bounding box should be scaled when the text object is scaled.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Scalable Bounding Box Property Examples

```
 /*The following procedure checks if the text
**object's bounding box is scaled when the
**text object is scaled.
*/
PROCEDURE Scalebox IS
  val boolean;
  text og_object;
BEGIN
   text := og_get_object('text object');
   val := og_get_bbscale(text);
   if val  then
      og_set_bbscale(text, false);
   else
      og_set_bbscale(text, true);
   end if;
END;
```

Scalable Font Property

Description Specifies whether the point size of the font should be scaled when the text object is scaled. The value of this property may be one of the following:

Syntax

```
PROCEDURE OG_Set_Fontscale
  (text        OG_Object,
   fontscale   BOOLEAN,
   damage      BOOLEAN    :=  TRUE,
   update_bbox BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Fontscale
  (text  OG_Object)
RETURN BOOLEAN;
```

Parameters

text	Is the text object being described.
fontscale	Specifies whether the point size of the font should be scaled when the text object is scaled.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Scalable Font Property Examples

```
 /* The following procedure checks if the point size is
** scaled when the text object is scaled.
*/
PROCEDURE Scalable IS
  val boolean;
  text og_object;
BEGIN
   text := og_get_object('text object');
   val := og_get_fontscale(text);
   if val  then
      og_set_fontscale(text, false);
   else
      og_set_fontscale(text, true);
   end if;
END;
```

Spacing Property

Description Is the line spacing for the text object. The value of this property may be one of the built-in constants listed below. If custom spacing is set, the value of the *Custom Spacing* property should specify the exact spacing amount.
OG_Single_Space
OG_Onehalf_Space
OG_Double_Space
OG_Custom_Space   Means the text uses custom line spacing.   The actual spacing used is defined in the *Custom Spacing* property.
Syntax

```
PROCEDURE OG_Set_Spacing
  (text         OG_Object,
   spacing      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Spacing
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
spacing	Is the line spacing for the text object.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Spacing Property Examples

```
 /* The following button procedure
**rotates the spacing setting of a
**text object.
*/
PROCEDURE Spacing (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
num number;
BEGIN
   num := og_get_spacing(hitobj);
   if num = og_single_space then
      og_set_spacing(hitobj, og_onehalf_space);
   elsif num = og_onehalf_space then
   og_set_spacing(hitobj, og_double_space);
   elsif num = og_double_space then
   og_set_spacing(hitobj, og_custom_space);
   elsif num = og_custom_space then
   og_set_spacing(hitobj, og_single_space);
   end if;
END;
```

Style Property

Description Is the font's style. Not all styles are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants:

OG_Blink_Fontstyle
OG_Inverted_Fontstyle
OG_Italic_Fontstyle
OG_Oblique_Fontstyle
OG_Outline_Fontstyle
OG_Overstrike_Fontstyle
OG_Plain_Fontstyle
OG_Shadow_Fontstyle
OG_Underline_Fontstyle
OG_Unknown_Fontstyle Means the style is unknown. You cannot *set* a style to this value; however, if you *get* a font and Graphics Builder cannot determine its style, this value is returned.

Syntax

```
PROCEDURE OG_Set_Font_Style
  (text         OG_Object,
   style        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
style	Is the font's style.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Style Property Examples

```
 /*The following button procedure
**rotates the text style
*/
PROCEDURE style (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
text og_object;
num number;
BEGIN
  text := og_get_object('text object');
  num := og_get_font_style(text,0,0);
  if num = og_blink_fontstyle then
    og_set_font_style(text, og_inverted_fontstyle);
  elsif num = og_inverted_fontstyle then
    og_set_font_style(text, og_italic_fontstyle);
  elsif num = og_italic_fontstyle then
    og_set_font_style(text, og_oblique_fontstyle);
  elsif num = og_oblique_fontstyle then
    og_set_font_style(text, og_outline_fontstyle);
  elsif num = og_outline_fontstyle then
    og_set_font_style(text, og_overstrike_fontstyle);
  elsif num = og_overstrike_fontstyle then
    og_set_font_style(text, og_plain_fontstyle);
  elsif num = og_plain_fontstyle then
    og_set_font_style(text, og_shadow_fontstyle);
  elsif num = og_shadow_fontstyle then
    og_set_font_style(text, og_underline_fontstyle);
  elsif num = og_underline_fontstyle then
    og_set_font_style(text, og_unknown_fontstyle);
  elsif num = og_unknown_fontstyle then
    og_set_font_style(text, og_blink_fontstyle);
  end if;
END;
```

Synthesize Property

Description Specifies whether Graphics Builder should try to synthesize the desired font (if the specified font cannot be found) by transforming the nearest-matching font.

Syntax

```
PROCEDURE OG_Set_Font_Synthesize
  (text         OG_Object,
   synthesize   BOOLEAN,
   damage       BOOLEAN  :=  TRUE,
   update_bbox  BOOLEAN  :=  TRUE);
```

Parameters

text	Is the text object being described.
synthesize	Specifies whether Graphics Builder should try to synthesize the desired font (if the specified font cannot be found) by transforming the nearest-matching font.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Synthesize Property Examples

```
 /*The following button procedure sets synthesize
** properties to true.
*/
PROCEDURE synthesize (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS

BEGIN
     OG_Set_Font_Synthesize(hitobj, true);
END;
```

Typeface Property

Description Is the font's typeface (font name). Values for this field are system-specific, and may include typefaces such as times, courier, and helvetica. For more information, consult your system administrator or your system documentation.

Syntax

```
PROCEDURE OG_Set_Font_Typeface
  (text         OG_Object,
   typeface     VARCHAR2,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
typeface	Is the font's typeface (font name).
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Typeface Property Examples

```
*/The following procedure reads
**the current typeface from the
**selected text object.  If the
**current style is not the same
**as the typeface from the argument,
**it assigns a new typeface to the
**text object.
PROCEDURE fonttypeface (text og_object, typeface varchar2)IS
style varchar2(10);
BEGIN
  style := og_get_font_typeface(text, 0,0);
  if style != typeface then
    og_set_font_typeface(text, typeface);
  end if;
END;
```

Vertical Alignment Property

Description Is the vertical alignment of the text object. The value of this property may be one of the following built-in constants:

OG_Top_Valign

OG_Middle_Valign

OG_Bottom_Valign

Syntax

```
PROCEDURE OG_Set_Valign
  (text         OG_Object,
   valign       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Valign
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
valign	Is the vertical alignment of the text object.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Vertical Alignment Property Examples

```
 /* The following procedure reads the vertical
** alignment and readjusts it.
*/
PROCEDURE VAligin IS
  num number;
  text og_object;
BEGIN
   text := og_get_object('text object');
   num := og_get_valign(text);
   if num = og_top_valign then
     og_set_valign(text, og_middle_valign);
   elsif num = og_middle_valign then
     og_set_valign(text, og_bottom_valign);
   elsif num = og_bottom_valign then
     og_set_valign(text, og_top_valign);
   end if;
END;
```

Vertical Origin Property

Description Is the vertical position of the text object relative to its origin point. The value of this property may be one of the following built-in constants:

OG_Top_Vorigin Means the origin point lies along the top edge of the bounding box.

OG_Middle_Vorigin Means the origin point lies equally between the top and bottom edges of the bounding box.

OG_Bottom_Vorigin Means the origin point lies along the bottom edge of the bounding box.

Syntax

```
PROCEDURE OG_Set_Vorigin
  (text         OG_Object,
   vorigin      NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Vorigin
  (text  OG_Object)
RETURN NUMBER;
```

Parameters

text	Is the text object being described.
vorigin	Is the vertical position of the text object relative to its origin point.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Vertical Origin Property Examples

```
 /*The following procedure reads the
**vertical origin and readjusts it.
*/
PROCEDURE Vorigin IS
  num number;
  text og_object;
BEGIN
   text := og_get_object('text object');
   num := og_get_vorigin(text);
   if num = og_top_vorigin then
     og_set_vorigin(text, og_middle_vorigin);
   elsif num = og_middle_vorigin then
     og_set_vorigin(text, og_bottom_vorigin);
   elsif num = og_bottom_vorigin then
     og_set_vorigin(text, og_top_vorigin);
   end if;
END;
```

Weight Property

Description Is the font's weight. Not all weights are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants:
OG_Bold_Fontweight
OG_Demibold_Fontweight
OG_Demilight_Fontweight
OG_Extrabold_Fontweight
OG_Extralight_Fontweight
OG_Light_Fontweight
OG_Medium_Fontweight
OG_Ultrabold_Fontweight
OG_Ultralight_Fontweight
OG_Unknown_Fontweight Means the weight is unknown. You cannot *set* a weight to this value; however, if you *get* a font and Graphics Builder cannot determine its weight, this value is returned.
Syntax

```
PROCEDURE OG_Set_Font_Weight
  (text         OG_Object,
   weight       NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
weight	Is the font's weight.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Weight Property Examples

```
 /*The following button procedure
**rotates the font weight for a
**selected text object.
*/
PROCEDURE weight (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  num number;
  text og_object;
BEGIN
  text := og_get_object('text object');
  num := og_get_font_weight(text,0,0);
  if num = og_bold_fontweight then
    og_set_font_weight(text, og_demibold_fontweight);
  elsif num = og_demibold_fontweight then
    og_set_font_weight(text, og_demilight_fontweight);
  elsif num = og_demilight_fontweight then
    og_set_font_weight(text, og_extrabold_fontweight);
  elsif num = og_extrabold_fontweight then
    og_set_font_weight(text, og_extralight_fontweight);
  elsif num = og_extralight_fontweight then
    og_set_font_weight(text, og_light_fontweight);
  elsif num = og_light_fontweight then
    og_set_font_weight(text, og_medium_fontweight);
  elsif num = og_medium_fontweight then
    og_set_font_weight(text, og_ultrabold_fontweight);
  elsif num = og_ultrabold_fontweight then
    og_set_font_weight(text, og_ultralight_fontweight);
  elsif num = og_ultralight_fontweight then
    og_set_font_weight(text, og_unknown_fontweight);
  elsif num = og_unknown_fontweight then
    og_set_font_weight(text, og_bold_fontweight);
  end if;
END;
```

Width Property

Description Is the font's width. Not all widths are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants:
OG_Dense_Fontwidth
OG_Expand_Fontwidth
OG_Extradense_Fontwidth
OG_Extraexpand_Fontwidth
OG_Normal_Fontwidth
OG_Semidense_Fontwidth
OG_Semiexpand_Fontwidth
OG_Ultradense_Fontwidth
OG_Ultraexpand_Fontwidth
OG_Unknown_Fontwidth Means the width is unknown. You cannot *set* a weight to this value; however, if you *get* a font and Graphics Builder cannot determine its width, this value is returned.
Syntax

```
PROCEDURE OG_Set_Font_Width
  (text         OG_Object,
   width        NUMBER,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
```

Parameters

text	Is the text object being described.
width	Is the font's width.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Width Property Examples

```
 /*The following button procedure
**rotates the font width for a
**selected test object.
*/
PROCEDURE width (buttonobj IN og_object,
                         hitobj IN og_object,
                         win IN og_window,
                         eventinfo IN og_event) IS
  num number;
  text og_object;
BEGIN
  text := og_get_object('text object');
  num := og_get_font_width(text,0,0);
  if num = og_dense_fontwidth then
    og_set_font_width(text, og_expand_fontwidth);
  elsif num = og_expand_fontwidth then
    og_set_font_width(text, og_extradense_fontwidth);
  elsif num = og_extradense_fontwidth then
    og_set_font_width(text, og_extraexpand_fontwidth);
  elsif num = og_extraexpand_fontwidth then
    og_set_font_width(text, og_normal_fontwidth);
  elsif num = og_normal_fontwidth then
    og_set_font_width(text, og_semidense_fontwidth);
  elsif num = og_semidense_fontwidth then
    og_set_font_width(text, og_semiexpand_fontwidth);
  elsif num = og_semiexpand_fontwidth then
    og_set_font_width(text, og_ultradense_fontwidth);
  elsif num = og_ultradense_fontwidth then
    og_set_font_width(text, og_ultraexpand_fontwidth);
  elsif num = og_ultraexpand_fontwidth then
    og_set_font_width(text, og_unknown_fontwidth);
  elsif num = og_unknown_fontwidth then
    og_set_font_width(text, og_dense_fontwidth);
  end if;
END;
```

Wraparound Property

Description Specifies whether the text should "wrap" to fit into the text object's bounding box. As described below, a compound text element represents a line of text, and is made up of simple text elements.

Syntax

```
PROCEDURE OG_Set_Wrap
  (text         OG_Object,
   wrap         BOOLEAN,
   damage       BOOLEAN    :=  TRUE,
   update_bbox  BOOLEAN    :=  TRUE);
FUNCTION OG_Get_Wrap
  (text  OG_Object)
RETURN BOOLEAN;
```

Parameters

text	Is the text object being described.
wrap	Specifies whether the text should "wrap" to fit into the text object's bounding box.
damage	Is the damage flag.
update_bbox	Is the bounding box update flag.

Wraparound Property Examples

```
  /*The following procedure checks if the text is 'wrapped'
** into the text's bounding box.
*/
PROCEDURE wrap IS
  val boolean;
  text og_object;
BEGIN
   text := og_get_object('text object');
   val := og_get_wrap(text);
   if val  then
      og_set_wrap(text, false);
   else
      og_set_wrap(text, true);
   end if;
END;
```

Timer Properties

Active Property
Interval Property
Name Property
Procedure Property

Active Property

Description Specifies whether the timer is active.

Syntax

```
PROCEDURE OG_Set_Active
  (timer   OG_Timer,
   active  BOOLEAN);
FUNCTION OG_Get_Active
  (timer  OG_Timer)
RETURN BOOLEAN;
```

Parameters

timer	Is the timer object being described.
active	Specifies whether the timer is active.

Active Property Examples

```
/* The following sets the timer to inactive if it
**is currently in active mode.
*/
PROCEDURE TimerActive IS
  val boolean;
  timer og_timer;
BEGIN
  timer := og_get_timer('timer2');
  val := og_get_active(timer);
  if val then
    og_set_active(timer, false);
  end if;
END;
```

Interval Property

Description Is the number of seconds that will pass between each execution of the timer procedure.

Syntax

```
PROCEDURE OG_Set_Interval
  (timer     OG_Timer,
   interval  NUMBER);
FUNCTION OG_Get_Interval
  (timer  OG_Timer)
RETURN NUMBER;
```

Parameters

timer	Is the timer object being described.
interval	Is the number of seconds that will pass between each execution of the timer procedure.

Interval Property Examples

```
 /* The following procedure adds two
**seconds to the original timer
**interval.
*/
PROCEDURE TimerInterval IS
  interval number;
  timer og_timer;
BEGIN
  timer := og_get_timer('timer2');
  interval := og_get_interval(timer);
  og_set_interval(timer, interval+2);
END;
```

Name Property

Description Is the name of the timer.

Syntax

```
PROCEDURE OG_Set_Name
  (timer  OG_Timer,
   name   VARCHAR2);
FUNCTION OG_Get_Name
  (timer  OG_Timer)
RETURN VARCHAR2;
```

Parameters

timer	Is the timer object being described.
name	Is the name of the timer.

Name Property Examples

```
 /*The following procedure appends
**a '1' to the name of a timer.
*/
PROCEDURE TimerName IS
  name varchar2(10);
  timer og_timer;
BEGIN
  timer := og_get_timer('timer1');
  name := og_get_name(timer);
  og_set_name(timer, name||'1');
END;
```

Procedure Property

Description Is the name of the procedure that will be executed when the timer is fired.

Syntax

```
PROCEDURE OG_Set_Timerproc
  (timer      OG_Timer,
   timerproc  VARCHAR2);
FUNCTION OG_Get_Timerproc
  (timer  OG_Timer)
RETURN VARCHAR2;
```

Parameters

timer	Is the timer object being described.
timerproc	Is the name of the procedure that will be executed when the timer is fired.

Procedure Property Examples

```
 /* The following procedure changes the
**timer procedure to "NewProc" if it is
** current timer procedure.
*/
PROCEDURE TimerProc IS
  name varchar2(20);
  timer og_timer;
BEGIN
  timer := og_get_timer('timer2');
  name := og_get_timerproc(timer);
  if name != 'NewProc' then
    og_set_timerproc(timer, 'NewProc');
  end if;
END;
```

Window Properties

Description
The position and dimensions of windows are expressed in "screen resolution units," more commonly known as pixels. You can obtain the horizontal and vertical values of the screen resolution using the built-ins OG_Get_Ap_Hscreen_Res and OG_Get_Ap_Vscreen_Res.
You should use these built-ins instead of an actual numeric value so that your application will maintain a consistent look on systems with different screen resolutions.
Height Property
Help Target Property
Name Property
Position Property
Width Property

Height Property

Description Is the height of the window (in screen resolution units).
Syntax
(See OG_Set_Window_Size.)
```
FUNCTION OG_Get_Window_Height
  (window  OG_Window)
RETURN NUMBER;
```

Parameters

window	Is the window being described.

Height Property Examples

```
/* /*The following procedure resizes
**the window to half its original size.
*/
PROCEDURE WinSize IS
  window og_window;
  width number;
  height number;
BEGIN
  window := og_get_window('Main Layout');
  width := og_get_window_width(window);
  height := og_get_window_height(window);
  og_set_window_size(window, width/2,height/2);
END;
```

Help Target Property

Description Is the hypertext target in the runtime Help document that is displayed when the Help system is invoked while the window is active.

Syntax

```
PROCEDURE OG_Set_Helptarget
  (window      OG_Window,
   helptarget  VARCHAR2);
FUNCTION OG_Get_Helptarget
  (window  OG_Window)
RETURN VARCHAR2;
```

Parameters

window	Is the window being described.
helptarget	Is the hypertext target in the runtime Help document that is displayed when the Help system is invoked while the window is active.

Help Target Property Examples

```
 /*The following procedure sets the help
**target to 'NewTarget' if "New Target" is not
** the current help target.
*/
PROCEDURE Help IS
  window og_window;
  help varchar2(20);
BEGIN
  window := og_get_window('Main Layout');
  help :=  og_get_helptarget(window);
  if help != 'NewTarget' then
    og_set_helptarget(window, 'NewTarget');
  end if;
END;
```

Name Property

Description Is the window's name. At runtime, the default name of the layout window is "Main Layout".

Syntax

```
PROCEDURE OG_Set_Name
  (window  OG_Window,
   name    VARCHAR2);
FUNCTION OG_Get_Name
  (window  OG_Window)
RETURN VARCHAR2;
```

Parameters

window	Is the window being described.
name	Is the window's name.

Name Property Examples

```
 /*The following procedure resets
**the name of the window if its name
**is not 'Main Layout'.
*/
PROCEDURE Name IS
  window og_window;
  name varchar2(20);
BEGIN
  window := og_get_window('Main Layout');
  name :=  og_get_name(window);
  if name != 'Main Layout' then
    og_set_name(window, 'Main Layout');
  end if;
END;
```

Position Property

Description Is the x- and y-coordinates of the window's upper left corner (in screen resolution units).

Syntax

```
PROCEDURE OG_Set_Position
  (window    OG_Window,
   position  OG_Point);
FUNCTION OG_Get_Position
  (window  OG_Window)
RETURN OG_Point;
```

Parameters

window	Is the window being described.
position	Is the x- and y-coordinates of the window's upper left corner (in screen resolution units).

Position Property Examples

```
 /*The following procedure repositions
**the window.
*/
PROCEDURE Position IS
  window og_window;
  pos og_point;
BEGIN
  window := og_get_window('Main Layout');
  pos := og_get_position(window);
  pos.x := pos.x*2;
  pos.y := pos.y*2;
  og_set_position(window, pos);
END;
```

Width Property

Description Is the width of the window (in screen resolution units).

Syntax

```
PROCEDURE OG_Set_Window_Size
  (window  OG_Window,
   width   NUMBER,
   height  NUMBER);
FUNCTION OG_Get_Window_Width
  (window  OG_Window)
RETURN NUMBER;
```

Parameters

window	Is the window being described.
width	Is the width of the window (in screen resolution units).
height	Is the height of the window (in screen resolution units).

Width Property Examples

```
 /*The following procedure resizes the window
** to half its original size.
*/
PROCEDURE WinSize IS
  window og_window;
  width number;
  height number;
BEGIN
  window := og_get_window('Main Layout');
  width := og_get_window_width(window);
  height := og_get_window_height(window);
  og_set_window_size(window, width/2,height/2);
END;
```

Attributes

Using Attribute Records

- overview
- shortcut built-ins

attribute record descriptions

Overview

Many of the built-in subprograms accept an argument that is described as an "attribute record." An "attribute" is simply a property or characteristic of some Graphics Builder object. For example, one attribute of a rectangle is the foreground fill color; two attributes of an arc are the start angle and end angle. Graphics Builder has identified enough attributes to completely describe the structure and appearance of any object.

Graphics Builder provides several new built-in variable datatypes to control these attributes, most of which are defined to be RECORDs. (For more information on the RECORD datatype, see the *PL/SQL User's Guide and Reference*.) Each field in one of these records represents a particular attribute. Thus, an "attribute record" refers to some variable whose type you have declared to be one of these new record datatypes.

For example, below is the type definition of OG_LINE_ATTR, the attribute record for a line:

```
TYPE og_line_attr IS RECORD
(mask        NUMBER(1,1),
 startpt     og_point,
 endpt       og_point,
 arrowstyle  NUMBER(1,0)
);
```

This record specifies attributes for a line's starting point, end point, and arrow style (the *mask* attribute will be described later).

All of an object's attributes are represented in one of several attribute records. To programmatically modify one of these attributes (for example, to change its fill pattern), you must change the values of the appropriate fields in the appropriate attribute record, and then pass the attribute record to a procedure or function. (Note that procedures and functions actually carry out your desired actions; however, you must use an attribute record to indicate to the procedure or function exactly what it is you want it to do.)

Attribute Classes

Some attribute records contain attributes that are common to many object types, while others contain attributes that are specific only to one object type. For example, every object can have a name, but only text objects have a font size.

All attributes have been organized into the following classes:

- generic

- graphic
- object-specific

Generic Attributes

Generic attributes apply to most object classes. For example, most objects may have a name, an associated button procedure, or a parent object.

Graphic Attributes

Graphic attributes apply to many object classes, but not all. They may be applied only to graphical objects (those objects that can be created with one of the graphical tools in the Layout editor). For example, a graphic attribute such as `fill color' may be used to describe a rectangle, arc, symbol, etc. However, it is meaningless to describe an image-which has no fill color-with this attribute. Similarly, a group object cannot be described by graphic attributes. (Note that while a group is not a graphical object, the individual components of the group may be. Graphic attributes, then, may be applied to these components.)

Object-specific Attributes

Object-specific attributes apply only to a specific object class. For example, `start angle' is an attribute that describes only an arc, and not a rectangle, line, image, or any other object. Similarly, you may want to know the `number of children' that compose a group object, but it would be meaningless to use this attribute with any other object class. Graphics Builder has identified attributes that are specific to application, arc, chart, group, image, line, polygon, rectangle, rounded rectangle, symbol, text, and window objects.

A built-in attribute record has been defined for generic attributes, and another for graphic attributes. In addition, a separate attribute record has been defined for each collection of object-specific attributes.

The following is a list of Graphics Builder objects and the attribute records that are meaningful to each:

Object Class	***Attribute Records***
application	application
arc	generic
	graphic
	arc
chart	generic
	group
	chart
chart element	graphic
	chart element
display	display
graphic	generic
	graphic
group	generic
	group
image	genericimage
line	generic
	graphic
	line
polygon	generic
	graphic
	polygon
printer	printer
query	query
rectangle	generic
	graphic
	rectangle
rounded rectangle	generic
	graphic
	rounded rectangle

sound	sound
symbol	generic graphic symbol
text	generic graphic text
timer	timer
window	window

Combined Attribute Records

In addition to the attribute records described above, Graphics Builder also defines "combined attribute records." A combined attribute record *combines* into a single variable all of the attribute records needed to completely describe an object. As the name implies, it is another record, but each of its fields is either a generic, graphic, or object-specific attribute record. Thus, in most cases you can use a single combined attribute record to control all of an object's attributes, instead of using several separate attribute records to represent each of the object's attribute classes.

For example, the rectangle combined attribute record contains three fields, representing the generic, graphic, and rectangle attribute records. The image combined attribute record contains only two fields, representing the generic and image attribute records.

Below is the type definition of OG_LINE_CA, the combined attribute record for a line object:

```
TYPE og_line_ca IS RECORD
(line_caob  og_generic_attr,  /* generic attribute record*/
 line_caoh  og_graphic_attr,  /* graphic attribute record */
 line_caol  og_line_attr      /* line attribute record */
);
```

This combined attribute record contains three fields, representing generic, graphic, and line attribute records.

Mask Attributes

Each attribute record (but not *combined* attribute record) has a numeric field called a "mask." The value of this field indicates which attributes in the attribute record you want to change (i.e., set) or examine (i.e., get). When you use an attribute record as an argument for a procedure or function, that procedure or function will use the mask to determine which attributes it should pay attention to.

For example, suppose you want to change only an object's foreground fill color by setting the *ffcolor* attribute in a graphic attribute record, and then passing both that attribute record and the object's handle as arguments to the OG_SET_ATTR procedure. The procedure does not know which attributes you want it to set; should it change *all* of the object's graphic attributes, or just *some* of them? To learn this, it will look at the attribute record's *mask* attribute.

The value of a *mask* attribute indicates which attributes in its attribute record a procedure or function should use. This value is called a "mask value."

Mask Constants

To help you determine an appropriate mask value for an attribute record, Graphics Builder has associated each attribute with a different built-in numeric constant, called a "mask constant."

Below is another listing of the line attribute record, this time with its mask constants:

```
TYPE og_line_attr IS RECORD    Mask Constants:
(mask       NUMBER(1,0),
 startpt    og_point,           OG_STARTPT_LINEA
 endpt      og_point,           OG_ENDPT_LINEA
 arrowstyle NUMBER(1,0)         OG_ARROWSTYLE_LINEA
);
                                OG_ALL_LINEA
                                OG_NONE_LINEA
```

After determining which attributes in an attribute record you want to use, calculate the sum of the mask constants that are associated with those attributes. The result will be a mask value that represents only those attributes. If you set the *mask* attribute in the attribute record to this mask value, then any procedure or function to which you pass this attribute record will pay attention only to those attributes.
For example, to change the *startpt* attribute in the above line attribute record, first declare a variable of this type:

```
my_variable   og_line_attr;
```

Then set the new value of the *startpt* attribute:

```
my_variable.startpt := new_point;
```

Finally, set the mask to indicate that you want to set a new starting point:

```
my_variable.mask := OG_STARTPT_LINEA;
```

(Note that this series of actions only *prepares* an attribute record for use by a procedure or function. To understand how this relates to actually modifying an object, see the description of the specific procedure or function.)
If you wanted to set new values for both the starting point and end point of the line, you need to set the mask to indicate that. In this case, the appropriate mask value would be the sum of the mask constants for those two attributes:

```
my_variable.mask := OG_STARTPT_LINEA +  OG_ENDPT_LINEA;
```

In addition to the mask constants for each attribute, every attribute record contains two additional attributes to indicate that *all* of the attributes should be used by a procedure or function, or that *none* should be used. For the line attribute record, these mask constants are OG_ALL_LINEA and OG_NONE_LINEA.
Remember that these mask constants are numbers, and may be treated as such. Besides adding them to indicate multiple attributes, you can also subtract them. For example, to indicate that all attributes *except* the end point should be affected by a procedure or function, you can set the mask value to:

```
my_variable.mask := OG_ALL_LINEA - OG_ENDPT_LINEA;
```

In some cases, the same mask constant is used to represent multiple attributes within an attribute record. If that mask constant is used to calculate the mask value, then all of the attributes represented by that constant will be used by the procedure or function to which the attribute record is passed.

Masks in Combined Attribute Records

It was stated above that all attribute records contain a mask attribute, but *combined* attribute records do not. When you pass a combined attribute record as an argument to a procedure, that procedure will use the masks from each of the attribute records that are contained within it.
For example, suppose you declare a variable to be a line combined attribute record (recall that a line combined attribute record contains attribute records for generic, graphic, and line-specific attributes):

```
comb_variable   og_line_ca;
```

Next, you want to change several of the record's attributes. In the generic attribute record, you want to change no values; in the graphic attribute record, you want to change the values of both the *dashstyle* and *capstyle* attributes; in the line attribute record, you want to change the value of only the *arrowstyle* attribute. Below are the statements you might use:

```
comb_variable.line_caoh.dashstyle := new_dashstyle;
comb_variable.line_caoh.capstyle := new_capstyle;
comb_variable.line_caol.arrowstyle := new_arrowstyle;
```

Before you can pass this combined attribute record to a procedure that will implement your changes, you must set the mask in *each* attribute record to indicate which attributes in that record the procedure should use:

```
comb_variable.line_caob.mask := OG_NONE_GENERICA;
comb_variable.line_caoh.mask := OG_DASHSTYLE_GRAPHICA +
                                OG_CAPSTYLE_GRAPHICA;
comb_variable.line_caol.mask := OG_ARROWSTYLE_LINEA;
```

Note that you must set the mask for every attribute record within a combined attribute record, even if you do not want to use any attributes within that attribute record. In this situation, you would set the mask to the mask constant that indicates no attributes will be used.
Once you have set the masks for each of the individual attribute records, you can pass the combined attribute record to a procedure or function. Remember that an attribute record's mask value is the only way the procedure or function will know which attributes you want it to use.

Createable, Setable, Getable Attributes

Next to the listing of each attribute described below, you will find a one-, two-, or three-letter designation.

Letter	***Meaning***
C	Indicates the attribute is createable. This means that Graphics Builder will recognize the value you assign to the attribute when the object containing the attribute is first created. If the attribute is not createable, Graphics Builder will provide a default value when the object is created.
S	Indicates the attribute is setable. This means that you are able to set the value of the attribute by invoking the appropriate Graphics Builder built-in subprogram.
G	Indicates the attribute is getable. This means that you are able to get the value of the attribute by invoking the appropriate Graphics Builder built-in subprogram.

Shortcut Built-ins

In addition to the attribute record approach described above, Graphics Builder also provides a series of built-in subprograms to simplify the process of creating objects and getting or setting their attributes. Each of these "shortcut" subprograms can be used to set or get a single attribute of an object. For more information, see Graphics Builder Built-in overview.

For example, to set an object's fill and edge patterns using the attribute record approach, you need to set the new fill patterns, set the appropriate masks, and call OG_SET_ATTR:

```
PROCEDURE attr_rec_approach (my_obj OG_OBJECT) IS
   my_rec   og_graphic_ca;
BEGIN
   my_rec.graphic_caoh.fillpatt:='gray50';
   my_rec.graphic_caoh.edgepatt:='kangaroo';
   my_rec.graphic_caob.mask:=OG_NONE_GENERICA;
   my_rec.graphic_caoh.mask:=OG_FILLPATT_GRAPHICA+
                             OG_EDGEPATT_GRAPHICA;
   og_set_attr(my_obj, my_rec);
END;
```

By using the shortcuts, you can accomplish the same thing with only two procedure calls:

```
PROCEDURE shortcut_approach (my_obj OG_OBJECT) IS
BEGIN
   og_set_fillpatt(my_obj, 'gray50');
   og_set_edgepatt(my_obj, 'kangaroo');
END;
```

Advantages

Using the shortcuts instead of attribute records has the following advantages:

- It requires less PL/SQL code, thus reducing development time.
- It makes your program units easier to read and understand.

Disadvantages

Using the shortcuts instead of attribute records has the following disadvantages:

- It is less efficient. Because Graphics Builder uses attribute records internally, each time you call a shortcut, Graphics Builder must define and populate a new internal attribute record. In addition, it takes longer to execute multiple `set' routines than it does to execute just one. In the above example, the first procedure (with one `set' call) will be roughly twice as fast as the second procedure (with two `set' calls).
- It requires your application to rely on default settings, since calling multiple shortcuts to set all of the necessary attributes may seriously affect your application's performance.

Application Attribute Record

The application attribute record contains attributes that may be used with the current application.

```
TYPE og_app_attr IS RECORD          Mask Constants:
(mask          NUMBER(3,0),
 cursor        VARCHAR2(255),       OG_CURSOR_APPA
 hscreen_res   NUMBER(5,0),         OG_SCREEN_RES_APPA
 vscreen_res   NUMBER(5,0),         OG_SCREEN_RES_APPA
 hlayout_res   NUMBER(10,0),        OG_LAYOUT_RES_APPA
 vlayout_res   NUMBER(10,0),        OG_LAYOUT_RES_APPA
 platform      NUMBER(1,0),         OG_PLATFORM_APPA
 username      VARCHAR2(255),       OG_USERNAME_APPA
 password      VARCHAR2(255),       OG_PASSWORD_APPA
 connection    VARCHAR2(255)        OG_CONNECTION_APPA
);
                                    OG_ALL_APPA
```

OG_NONE_APPA

	Attribute	*Description*
SG	cursor	Is the name of the mouse cursor to use. The value of this attribute may be one of the following strings: default insertion crosshair help busy The appearance of each cursor is system-specific. For more information, refer to your system documentation. If you set this attribute to an invalid value, it assumes the value `default.'
G	hscreen_res	Is the horizontal resolution of the screen. This value is the number of screen resolution units (i.e., pixels) in one horizontal inch of the screen.
G	vscreen_res	Is the vertical resolution of the screen. This value is the number of screen resolution units (i.e., pixels) in one vertical inch of the screen.
G	hlayout_res	Is the horizontal resolution of the layout. This value is the number of layout units in one horizontal inch of the layout.
G	vlayout_res	Is the vertical resolution of the layout. This value is the number of layout units in one vertical inch of the layout.
G	platform	Is the platform on which Graphics Builder is running. The value of this attribute may be one of the following built-in constants: **OG_MACINTOSH_PLATFORM** Means the platform is the Apple Macintosh. **OG_MOTIF_PLATFORM** Means the platform is OSF/MOTIF. **OG_MSWINDOWS_PLATFORM** Means the platform is Microsoft Windows. **OG_PM_PLATFORM** Means the platform is Presentation Manager. **OG_X_PLATFORM** Means the platform is the X Window System.
G	username	Is the username for the current database connection. If the user is not connected, this attribute is NULL.
G	password	Is the password for the current database connection. If the user is not connected, or the *Keep_Password* preference setting is set to No, this

		attribute is NULL.
G	connection	Is the database connection string for the current database connection. If the user is not connected, this attribute is NULL.

Arc Combined Attribute Record

The arc combined attribute record consists of a generic attribute record, graphic attribute record, and arc attribute record:

```
TYPE og_arc_ca IS RECORD
(arc_caob  og_generic_attr,   /* generic */
 arc_caoh  og_graphic_attr,   /* graphic */
 arc_caoa  og_arc_attr        /* arc */
);
```

Arc Attribute Record

The arc attribute record contains attributes that may be used only with arc objects:

```
TYPE og_arc_attr IS RECORD              Mask Constants:
(mask     NUMBER(1,0),
 basearc  og_arc,                        OG_BASEARC_ARCA
 arcfill  NUMBER(1,0),                   OG_ARCFILL_ARCA
 closed   BOOLEAN                        OG_CLOSED_ARCA
);
                                         OG_ALL_ARCA
                                         OG_NONE_ARCA
```

	Attribute	*Description*
CSG	basearc	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the ellipse from which the arc is cut.
CSG	arcfill	Is the fill shape of the arc. The value of this attribute may be one of the following built-in constants: **OG_CHORD_ARCFILL** Means the fill shape of the arc is that of a chord. **OG_PIE_ARCFILL** Means the fill shape of the arc is that of a full pie slice.
CSG	closed	Is the closure of the arc. The value of this attribute may be one of the following: **TRUE** Means the arc is closed. **FALSE** Means the arc is open.

Continuous Axis Combined Attribute Record

```
TYPE og_contaxis_ca IS RECORD
(ca_axis  og_axis_attr,       /* generic axis */
 ca_cont  og_contaxis_attr   /* continuous axis */
);
```

Continuous Axis Attribute Record

```
TYPE og_contaxis_attr IS RECORD    Mask Constants:
(mask      NUMBER(4,0),
 automin   BOOLEAN,                 OG_MINIMUM_CONTAXISA
 minimum   NUMBER(6),               OG_MINIMUM_CONTAXISA
 autostep  BOOLEAN,                 OG_STEP_CONTAXISA
 step      NUMBER(6),               OG_STEP_CONTAXISA
 automax   BOOLEAN,                 OG_MAXIMUM_CONTAXISA
 maximum   NUMBER(6),               OG_MAXIMUM_CONTAXISA
 scale     NUMBER(1,0),             OG_SCALE_CONTAXISA
 pct_of    NUMBER(1,0),             OG_PCTOF_CONTAXISA
 pct_by    NUMBER(1,0),             OG_PCTBY_CONTAXISA
 numfmt    VARCHAR2(255)            OG_NUMFMT_CONTAXISA
);
                                    OG_ALL_CONTAXISA
                                    OG_NONE_CONTAXISA
```

	Attribute	*Description*
SG	automin	Specifies whether the axis minimum is set to *Auto*.
SG	minimum	Specifies the minimum axis value (if *automin* is FALSE).
SG	autostep	Specifies whether the axis step value is set to *Auto*.
SG	step	Specifies the axis step value (if *autostep* is FALSE).
SG	automax	Specifies whether the axis maximum is set to *Auto*.
SG	maximum	Specifies the maximum axis value (if *automax* is FALSE).
SG	scale	Specifies the algorithm used for scaling the axis. The value of this attribute may be one of the following built-in constants: **OG_LINEAR_SCALE** Means the axis is scaled using a fixed interval between the minimum and maximum axis values. **OG_LOG_SCALE** Means the axis is scaled using a logarithmic algorithm (based on powers of 10) to determine the intervals between the minimum and maximum axis values.

		OG_PCT_SCALE Means the axis is scaled so that data values will be plotted relative to the amount specified by *pct_of*.
SG	pct_of	Specifies the relative scaling factor (if *scale* is set to OG_PCT_SCALE). The value of this attribute may be one of the following built-in constants: **OG_MAXIMUM_PCTOF** Meanseach data value is plotted as a percentage of the largest data value. **OG_MINIMUM_PCTOF** Means each data value is plotted as a percentage of the smallest data value. **OG_SUM_PCTOF** Means each data value is plotted as a percentage of the sum of all data values.
SG	pct_by	Specifies how the *pct_of* scaling values are calculated. The value of this attribute may be one of the following built-in constants: **OG_CATEGORY_PCTBY** Means the percentage for each data value is calculated relative to data values for the same field in other categories. **OG_FIELD_PCTBY** Means the percentage for each data value is calculated relative to data values in the same category for other fields.
SG	numfmt	Specifies the number format for the axis tick labels. This must be a valid SQL format string. For more information, see your*Oracle7 Server SQL Reference*.

Date Axis Combined Attribute Record

```
TYPE og_dateaxis_ca IS RECORD
(ca_axis  og_axis_attr,       /* generic axis */
 ca_date  og_dateaxis_attr   /* date axis */
);
```

Date Axis Attribute Record

```
TYPE og_dateaxis_attr IS RECORD   Mask Constants:
(mask       NUMBER(5,0),
 automin    BOOLEAN,               OG_MINIMUM_DATEAXISA
 minimum    DATE,                  OG_MINIMUM_DATEAXISA
 autostep   BOOLEAN,               OG_STEP_DATEAXISA
 step       NUMBER(2,0),           OG_STEP_DATEAXISA
 automax    BOOLEAN,               OG_MAXIMUM_DATEAXISA
 maximum    DATE,                  OG_MAXIMUM_DATEAXISA
 firstmon   NUMBER(2,0),           OG_FIRSTMON_DATEAXISA
 skipwknds  BOOLEAN,               OG_SKIPWKNDS_DATEAXISA
 labels     NUMBER(4,0),           OG_LABELS_DATEAXISA
 dayfmt     NUMBER(1,0),           OG_DAYFMT_DATEAXISA
 monthfmt   NUMBER(1,0),           OG_MONTHFMT_DATEAXISA
 qtrfmt     NUMBER(1,0),           OG_QTRFMT_DATEAXISA
 yearfmt    NUMBER(1,0),           OG_YEARFMT_DATEAXISA
 custfmt    VARCHAR2(255)          OG_CUSTMT_DATEAXISA
);
                                   OG_ALL_DATEAXISA
                                   OG_NONE_DATEAXISA
```

	Attribute	*Description*
SG	automin	Specifies whether the axis minimum is set to *Auto*.
SG	minimum	Specifies the minimum axis value (if *automin* is FALSE).
SG	autostep	Specifies whether the axis step value is set to *Auto*.
SG	step	Specifies the axis step value (if *autostep* is FALSE). The value of this attribute may be one of the following built-in constants: **OG_SECOND_STEP** **OG_MINUTE_STEP** **OG_HOUR_STEP** **OG_DAY_STEP** **OG_WEEK_STEP** **OG_MONTH_STEP** **OG_QUARTER_STEP** **OG_YEAR_STEP**
SG	automax	Specifies whether the axis maximum is set to *Auto*.
SG	maximum	Specifies the maximum axis value (if *automax* is FALSE).

SG	firstmonth	Is the month that is considered to begin a new year. The value of this attribute may be one of the following built-in constants: **OG_JAN_MONTH** **OG_FEB_MONTH** **OG_MAR_MONTH** **OG_APR_MONTH** **OG_MAY_MONTH** **OG_JUN_MONTH** **OG_JUL_MONTH** **OG_AUG_MONTH** **OG_SEP_MONTH** **OG_OCT_MONTH** **OG_NOV_MONTH** **OG_DEC_MONTH**
SG	skipweekends	Specifies whether weekends are ignored when calculating axis values.
SG	labels	Specifies the major interval along the axis at which major tick marks and tick labels appear, as well as the appearance of the tick labels. The value of this attribute may be one of the following built-in constants: **OG_NO_LABELS** **OG_SECOND_LABELS** **OG_MINUTE_LABELS** **OG_HOUR_LABELS** **OG_AMPM_LABELS** **OG_DAY_LABELS** **OG_DAYOFWEEK_LABELS** **OG_WEEK_LABELS** **OG_MONTH_LABELS** **OG_QUARTER_LABELS** **OG_YEAR_LABELS** **OG_CUSTOM_LABELS** (If *labels* is set to this value, you must specify the custom date format in the *customfmt* attribute.)
SG	dayfmt	Determines the appearance of day-of-the-week labels along the axis. The value of this attribute may be one of the following built-in constants: **OG_FIRSTLETTER_FMT** **OG_THREELETTER_FMT**
SG	monthfmt	Determines the appearance of month labels along the axis. The value of this attribute may be one of the following built-in constants: **OG_FIRSTLETTER_FMT** **OG_THREELETTER_FMT**
SG	quarterfmt	Determines the appearance of quarter labels along the axis. The value of this

		attribute may be one of the following built-in constants: **OG_ARABIC_FMT** **OG_ROMAN_FMT**
SG	yearfmt	Determines the appearance of year labels along the axis. The value of this attribute may be one of the following built-in constants: **OG_FOURDIGIT_FMT** **OG_TWODIGIT_FMT**
SG	custfmt	Is the custom date format for the axis tick labels. This must be a valid SQL format string. For more information, see your*Oracle7 Server SQL Reference*.

Discrete Axis Combined Attribute Record

```
TYPE og_discaxis_ca IS RECORD
(ca_axis  og_axis_attr,       /* generic axis */
 ca_disc  og_discaxis_attr   /* discrete axis */
);
```

Discrete Axis Attribute Record

```
TYPE og_discaxis_attr IS RECORD      Mask Constants:
(mask     NUMBER(3,0),
 automin  BOOLEAN,                    OG_MINCAT_DISCAXISA
 mincat   NUMBER(10,0),               OG_MINCAT_DISCAXISA
 automax  BOOLEAN,                    OG_MAXCAT_DISCAXISA
 maxcat   NUMBER(10,0),               OG_MAXCAT_DISCAXISA
 numfmt   VARCHAR2(255),              OG_NUMFMT_DISCAXISA
 datefmt  VARCHAR2(255)               OG_DATEFMT_DISCAXISA
);
                                      OG_ALL_DISCAXISA
                                      OG_NONE_DISCAXISA
```

	Attribute	*Description*
SG	automin	Specifies whether the minimum number of categories that appear on the axis is set to *Auto*.
SG	mincat	Specifies the minimum number of categories that appear on the axis (if *automin* is FALSE).
SG	automax	Specifies whether the maximum number of categories that appear on the axis is set to *Auto*.
SG	maxcat	Specifies the maximum number of categories that appear on the axis (if *automax* is FALSE).
SG	numfmt	Specifies the number format for the axis tick labels. This must be a valid SQL format string. For more information, see your*Oracle7 Server SQL Reference*.
SG	datefmt	Specifies the date format for the axis tick labels. This must be a valid SQL format string. For more information, see your*Oracle7 Server SQL Reference*.

Axis Attribute Record

```
TYPE og_axis_attr IS RECORD         Mask Constants:
(mask           NUMBER(5,0),
 axistype       NUMBER(1,0),         OG_AXISTYPE_AXISA
 custlabel      VARCHAR2(255),       OG_CUSTLABEL_AXISA
 position       NUMBER(1,0),         OG_POSITION_AXISA
 direction      NUMBER(1,0),         OG_DIRECTION_AXISA
 tickpos        NUMBER(1,0),         OG_TICKPOS_AXISA
 ticklabelrot   NUMBER(1,0),         OG_TICKLABELROT_AXISA
 minorct        NUMBER(1,0),         OG_MINORCT_AXISA
 majorticks     BOOLEAN,             OG_MAJORTICKS_AXISA
 minorticks     BOOLEAN,             OG_MINORTICKS_AXISA
 majorgrid      BOOLEAN,             OG_MAJORGRID_AXISA
 minorgrid      BOOLEAN,             OG_MINORGRID_AXISA
 axislabel      BOOLEAN,             OG_AXISLABEL_AXISA
 ticklabels     BOOLEAN              OG_TICKLABELS_AXISA
);
                                     OG_ALL_AXISA
                                     OG_NONE_AXISA
```

	Attribute	*Description*
SG	axistype	The value of this attribute may be one of the following built-in constants: **OG_CONTINUOUS_AXISTYPE** **OG_DATE_AXISTYPE** **OG_DISCRETE_AXISTYPE**
SG	custlabel	Specifies the text of the label that appears along the axis.
SG	position	Specifies along which edge of the chart the axis appears. The value of this attribute may be one of the following built-in constants: **OG_BOTTOM_POSITION** **OG_LEFT_POSITION** **OG_RIGHT_POSITION** **OG_TOP_POSITION**
SG	direction	Specifies in which direction increasing values, or successive categories, are placed along the axis. The value of this attribute may be one of the following built-in constants: **OG_DOWN_DIRECTION** **OG_LEFT_DIRECTION** **OG_RIGHT_DIRECTION** **OG_UP_DIRECTION**
SG	tickpos	Specifies how the major and minor tick marks appear. The value of this attribute may be one of the following built-in constants: **OG_CROSS_TICKPOS** **OG_INSIDE_TICKPOS** **OG_OUTSIDE_TICKPOS**

SG	ticklabelrot	The value of this attribute may be one of the following built-in constants: **OG_CCW_ROTATION** **OG_CW_ROTATION** **OG_NO_ROTATION**
SG	minorct	Is the number of minor ticks defined within each major tick interval.
SG	majorticks	Specifies whether major tick marks appear at each major interval.
SG	minorticks	Specifies whether minor tick marks appear, as specified by the value set for Minor Ticks per Interval.
SG	majorgrid	Specifies whether a grid line appears at each major tick mark.
SG	minorgrid	Specifies whether a grid line appears at each minor tick mark.
SG	axislabel	Specifies whether labels that identify values along the axis appear.
SG	ticklabels	Specifies whether labels that identify values along the axis appear.

Chart Combined Attribute Record

A chart is treated like a group object, consisting of lines, rectangles, text, etc. Therefore, the chart combined attribute record allows you access to group attributes, as well as attributes specific to a chart. Additionally, since a chart itself is not a graphical object (although the objects that compose it are), this record does not provide access to graphic attributes. To set the graphical attributes of individual elements of a chart, use the chart element attribute record (described below).
This record may be used to access the attributes of a chart drawn manually on the layout only if the chart was specified as dynamic. If the chart is artwork, it is considered to be a group object, and not a chart object. A chart that is created programmatically is a dynamic chart.
The chart combined attribute record consists of a generic attribute record, group attribute record, and chart attribute record:

```
TYPE og_chart_ca IS RECORD
(chart_caob  og_generic_attr,   /* generic */
 chart_caog  og_group_attr,     /* graphic */
 chart_caoc  og_chart_attr      /* chart */
);
```

Chart Attribute Record

The chart attribute record contains attributes that may be used only with chart objects:

```
TYPE og_chart_attr IS RECORD       Mask Constants:
(mask        NUMBER(4,0),
 frame       og_rectangle,          OG_FRAME_CHARTA
 template    og_template,           OG_TEMPLATE_CHARTA
 query       og_query,              OG_QUERY_CHARTA
 title       VARCHAR2(255),         OG_TITLE_CHARTA
 autoupdate  BOOLEAN,               OG_AUTOUPDATE_CHARTA
 rangeflag   BOOLEAN,               OG_ROWS_CHARTA
 startrow    NUMBER(10,0),          OG_ROWS_CHARTA
 endrow      NUMBER(10,0),          OG_ROWS_CHARTA
```

```
  filter      VARCHAR2(255)          OG_FILTER_CHARTA
 );
                                      OG_ALL_CHARTA
                                      OG_NONE_CHARTA
```

	Attribute	*Description*
CSG	frame	Is the x- and y-coordinates, height, and width of the chart's frame (in layout units).
CSG	template	Is the handle to the template to be used for the chart.
CSG	query	Is the handle to the query to be used for the chart.
CSG	title	Is the title of the chart.
CSG	autoupdate	Specifies that the chart is automatically be updated when the query is executed.
CSG	rangeflag	Specifies whether the number of query rows that appear on the chart is restricted to the range specified by *startrow* and *endrow*.
CSG	startrow	Is the first row from the query that appears on the chart. The first query row is 0, the second row is 1, and so on.
CSG	endrow	Is the last row from the query that appears on the chart.
CSG	filter	Is the name of the query's filter trigger procedure.

Chart Element Combined Attribute Record

A chart element is a graphical object that represents a single value for a field. For example, bars and pie slices are chart elements.This combined attribute record is used in conjunction with the OG_SET_ATTR procedure to change the attributes of a chart element.
The chart element combined attribute record consists of a graphic attribute record and chart element attribute record:

```
 TYPE og_chelement_ca IS RECORD
 (chelement_cagr  og_graphic_attr,    /* graphic */
  chelement_cace  og_chelement_attr   /* chart element */
 );
```

Chart Element Attribute Record

The chart element attribute record contains attributes that may be used only with chart elements:

```
 TYPE og_chelement_attr IS RECORD   Mask Constants:
 (mask        NUMBER(1,0),
  button      og_buttonproc,        OG_BUTTON_CHELEMENTA
  events      NUMBER(2,0),          OG_BUTTON_CHELEMENTA
  explosion   NUMBER(10,0),         OG_EXPLOSION_CHELEMENTA
  name        VARCHAR2(255)         OG_NAME_CHELEMENTA
 );
                                    OG_ALL_CHELEMENTA
                                    OG_NONE_CHELEMENTA
```

	Attribute	*Description*
S	button	Is the handle to the button procedure that should be associated with this chart element. Note that the events attribute must be set properly in order to ensure that this procedure receives the desired mouse events.
S	events	Is the type of mouse events that the button procedure should receive. The value of this attribute may be one of the built-in constants listed below. To enable the procedure to receive multiple event types, set this attribute to be the sum of the constants for the desired events. **OG_NO_EVENTS** Means the procedure receives no mouse events. **OG_MOUSE_DOWN** Means the procedure receives only mouse down events. **OG_MOUSE_MOVE_DOWN** Means the procedure receives only mouse move down events. **OG_MOUSE_UP** Means the procedure receives only mouse up events. **OG_MOUSE_MOVE_UP** Means the procedure receives only mouse move up events.
S	explosion	Is the distance that the chart element (i.e., pie slice) should be exploded, in terms of the percentage of the chart's x- and y-radii. This attribute is meaningful only when used with a pie chart. In addition, all of the pie slices for a given category will be exploded the same amount. Therefore, the attribute record that specifies an explosion value should be associated with an independent field.
S	name	Is the name of the chart element. To get the name of a chart element, use the generic attribute record.

Display Attribute Record

The display attribute record contains attributes that may be used only with the current display:

```
TYPE og_display_attr IS RECORD          Mask Constants:
(mask           NUMBER(2,0),
 opentrigger    VARCHAR2(255),        OG_OPENTRIGGER_DISPLAYA
 closetrigger   VARCHAR2(255),        OG_CLOSETRIGGER_DISPLAYA
 width          NUMBER(10,0),         OG_SIZE_DISPLAYA
 height         NUMBER(10,0),         OG_SIZE_DISPLAYA
 dateformat     VARCHAR2(255)         OG_DATEFORMAT_DISPLAYA
);
                                      OG_ALL_DISPLAYA
                                      OG_NONE_DISPLAYA
```

	Attribute	*Description*
SG	opentrigger	Is the name of diplay's Open Display trigger.
SG	closetrigger	Is tforhe name of diplay's Close Display trigger.
SG	width	Is the width of the layout (in layout units).
SG	height	Is the height of the layout (in layout units).
SG	dateformat	Specifies the date format for parameters. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Axis Field Template Combined Attribute Record

```
TYPE og_axisftemp_ca IS RECORD
(ca_ftemp   og_ftemp_attr,     /* generic field template */
 ca_aftemp  og_axisftemp_attr  /* axis field template */
);
```

Axis Field Template Attribute Record

```
TYPE og_axisftemp_attr IS RECORD   Mask Constants:
(mask       NUMBER(3,0),
 plottype   NUMBER(3,0),           OG_PLOTTYPE_AXISFTEMPA
 linestyle  NUMBER(1,0),           OG_LINESTY_AXISFTEMPA
 labelrot   NUMBER(1,0),           OG_LABELROT_AXISFTEMPA
 plotpos    NUMBER(1,0),           OG_PLOTPOS_AXISFTEMPA
 overlap    NUMBER(3),             OG_OVERLAP_AXISFTEMPA
 axis       NUMBER(1,0),           OG_AXIS_AXISFTEMPA
 curvefit   NUMBER(1,0)            OG_CURVEFIT_AXISFTEMPA
);
                                   OG_ALL_AXISFTEMPA
                                   OG_NONE_AXISFTEMPA
```

	Attribute	***Description***
SG	plottype	Specifies the elements used to plot this field on the chart. The value of this attribute may be one of the following built-in constants: **OG_NONE_PLOTTYPE** **OG_BAR_PLOTTYPE** **OG_LINE_PLOTTYPE** **OG_SYMBOL_PLOTTYPE** **OG_FILL_PLOTTYPE** **OG_SPIKE_PLOTTYPE** **OG_LABEL_PLOTTYPE**
SG	linestyle	Specifies the line style used to connect the data points of a field with a line plot type. The value of this attribute may be one of the following built-in constants: **OG_SPLINE_LINESTYLE** **OG_STEP_LINESTYLE** **OG_STRAIGHT_LINESTYLE**
SG	labelrot	Specifies the rotation angle of the labels for a field with a label plot type. The value of this attribute may be one of the following built-in constants:
SG	plotpos	Specifies—for each category—the relationship between the data values of two or more fields. The value of this attribute may be one of the following built-in constants:

		OG_NORMAL_PLOTPOS **OG_FROMPREV_PLOTPOS** **OG_STACKED_PLOTPOS**
SG	overlap	Specifies the percentage by which bars representing data values from multiple fields in a bar or column chart overlap each other.
SG	axis	Specifies the axis to which data values are compared to determine how the field is plotted. The value of this attribute may be one of the following built-in constants: **OG_X_AXIS** **OG_Y1_AXIS** **OG_Y2_AXIS**
SG	curvefit	Specifies whether a curve fit is applied to the chart and, if so, which algorithm is used. The value of this attribute may be one of the following built-in constants: **OG_NO_CURVEFIT** **OG_LINEAR_CURVEFIT** **OG_LOG_CURVEFIT** **OG_EXP_CURVEFIT** **OG_POWER_CURVEFIT**

Field Template Attribute Record

```
TYPE og_ftemp_attr IS RECORD
(mask      NUMBER(3,0),
 name      VARCHAR2(255),
 root      OG_OBJECT,
 colorrot  NUMBER(1,0),
 numfmt    VARCHAR2(255),
 datefmt   VARCHAR2(255)
);
```

```
Mask Constants:

OG_NAME_FTEMPA
OG_ROOT_FTEMPA
OG_COLORROT_FTEMPA
OG_NUMFMT_FTEMPA
OG_DATEFMT_FTEMPA

OG_ALL_FTEMPA
OG_NONE_FTEMPA
```

	Attribute	***Description***
SG	name	Is the name of the field template.
G	root	Is a handle to the chart template to which the field template belongs.
SG	colorrot	Specifies whether Graphics Builder automatically rotates through the color or pattern palette to select a unique shading for each field that uses this field template. The value of this attribute may be one of the following built-in constants: **OG_NO_COLORROT** **OG_AUTO_COLORROT** **OG_COLOR_COLORROT** **OG_PATTERN_COLORROT** **OG_BOTH_COLORROT**
SG	numfmt	Specifies the number format for the field labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.
SG	datefmt	Specifies the date format for the field labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Axis Frame Combined Attribute Record

```
TYPE og_axisframe_ca IS RECORD
(ca_frame  og_frame_attr,       /* generic frame */
 ca_axis   og_axisframe_attr    /* axis frame */
);
```

Axis Frame Attribute Record

```
TYPE og_axisframe_attr IS RECORD   Mask Constants:
(mask       NUMBER(3,0),
 reflinect  NUMBER(3,0),           OG_REFLINECT_AXISFRAMEA
 basevalue  NUMBER(1,0),           OG_BASEVALUE_AXISFRAMEA
 cust_num   NUMBER(6),             OG_BASEVALUE_AXISFRAMEA
 cust_date  DATE,                  OG_BASEVALUE_AXISFRAMEA
 base_axis  NUMBER(1,0),           OG_BASEAXIS_AXISFRAMEA
 catwidth   NUMBER(3,0),           OG_CATWIDTH_AXISFRAMEA
 second_y   BOOLEAN                OG_SECONDY_AXISFRAMEA
);
                                   OG_ALL_AXISFRAMEA
                                   OG_NONE_AXISFRAMEA
```

	Attribute	*Description*
G	reflinect	Is the number of reference lines that belong to the chart template.
SG	baseline_value	Is the value used as the starting point for plotting fields along the value axis. The value of this attribute may be one of the following built-in constants: **OG_MIN_BASELINE** **OG_MAX_BASELINE** **OG_ZERO_BASELINE** **OG_CUSTOM_BASELINE**
SG	custom_num	Specifies the custom number to set the baseline to. This will automatically set the base value to OG_CUSTOM_BASELINE.
SG	custom_date	Specifies the custom date to set the custom date value to. This will automatically set the base value to OG_CUSTOM_BASELINE.
SG	baseline_axis	Specifies the axis to which the baseline value is compared to determine its position.
SG	catwidth	Is the width of the bars in a bar or column chart, as a percentage of the "strip width." The strip width is the widest the bars can be without overlapping each other, and it is determined by dividing the length of the category axis by the number of bars

		to be plotted.
SG	second_y	Specifies whether a second Y axis appears in the chart.

Frame Attribute Record

```
TYPE og_frame_attr IS RECORD       Mask Constants:
(mask          NUMBER(4,0),
 name          VARCHAR2(255),      OG_NAME_FRAMEA
 frametype     NUMBER(1,0),        OG_FRAMETYPE_FRAMEA
 ftempct       NUMBER(5,0),        OG_FTEMPCT_FRAMEA
 root          OG_OBJECT,          OG_ROOT_FRAMEA
 depthsize     NUMBER(1,0),        OG_DEPTHSIZE_FRAMEA
 shadowsize    NUMBER(1,0),        OG_SHADOWSIZE_FRAMEA
 shadowdir     NUMBER(1,0),        OG_SHADOWDIR_FRAMEA
 plotframe     BOOLEAN,            OG_PLOTFRAME_FRAMEA
 legend        BOOLEAN,            OG_LEGEND_FRAMEA
 legendcolct   NUMBER(3,0)         OG_LEGENDCOLCT_FRAMEA
);
                                   OG_ALL_FRAMEA
                                   OG_NONE_FRAMEA
```

	Attribute	*Description*
SG	name	Is the name of the chart template.
G	frametype	Is the type of chart represented by this template The value of this attribute may be one of the following built-in constants: **OG_AXIS_FRAMETYPE** **OG_PIE_FRAMETYPE** **OG_TABLE_FRAMETYPE**
G	ftempct	Is the number of field templates that belong to the chart template.
G	root	Is the handle to the chart template.
SG	depthsize	Specifies the amount of depth with which the chart frame and elements are drawn to provide them with a 3-dimensional look. The value of this attribute may be one of the following built-in constants: **OG_NONE_DEPTHSIZE** **OG_SMALL_DEPTHSIZE** **OG_MEDIUM_DEPTHSIZE** **OG_LARGE_NONE_DEPTHSIZE** **OG_XLARGE_DEPTHSIZE**
SG	shadowsize	Specifies the size of the shadow with which the chart frame and elements are drawn. The value of this attribute may be one of the following built-in constants: **OG_NONE_SHADOWSIZE** **OG_SMALL_SHADOWSIZE** **OG_MEDIUM_SHADOWSIZE** **OG_LARGE_SHADOWSIZE** **OG_XLARGE_SHADOWSIZE**
SG	shadowdir	Specifies the direction of the shadow with which the chart frame and elements are drawn. The value of this attribute may be one of the following built-in constants: **OG_UPPERRIGHT_SHADOWDIR** **OG_UPPERLEFT_SHADOWDIR** **OG_LOWERRIGHT_SHADOWDIR** **OG_LOWERLEFT_SHADOWDIR**
SG	plotframe	Specifies whether the rectangle that surrounds the chart should be shown. (Not applicable to pie charts.)

SG	legend	Specifies whether the chart's legend should be shown. (Not applicable to table charts.)
SG	legendcolct	Is the number of columns used to display the labels that appear in the legend.

Pie Frame Combined Attribute Record

```
TYPE og_pieframe_ca IS RECORD
(ca_frame  og_frame_attr,     /* generic frame */
 ca_pie   og_pieframe_attr   /* pie frame */
);
```

Pie Frame Attribute Record

```
TYPE og_pieframe_attr IS RECORD   Mask Constants:
(mask        NUMBER(3,0),
 usage       NUMBER(1,0),         OG_USAGE_PIEFRAMEA
 usagevalue  NUMBER(6),           OG_USAGE_PIEFRAMEA
 plotorder   NUMBER(1,0),         OG_PLOTORDER_PIEFRAMEA
 categs      BOOLEAN,             OG_CATEGS_PIEFRAMEA
 datavals    BOOLEAN,             OG_DATAVALS_PIEFRAMEA
 pctvalues   BOOLEAN,             OG_PCTVALUES_PIEFRAMEA
 ticks       BOOLEAN,             OG_TICKS_PIEFRAMEA
 other       NUMBER(3),           OG_OTHER_PIEFRAMEA
 nooverlap   BOOLEAN,             OG_NOOVERLAP_PIEFRAMEA
 catnumfmt   VARCHAR2(255),       OG_CATNUMFMT_PIEFRAMEA
 catdatefmt  VARCHAR2(255),       OG_CATDATEFMT_AXSFRAMEA
 valuefmt    VARCHAR2(255),       OG_VALUEFMT_PIEFRAMEA
 pctfmt      VARCHAR2(255)        OG_PCTFMT_PIEFRAMEA
);
                                  OG_ALL_PIEFRAMEA
                                  OG_NONE_PIEFRAMEA
```

	Attribute	***Description***
SG	usage	Specifies the relationship between the individual pie slices and the complete chart. The value of this attribute may be one of the following built-in constants: **OG_TOTALVALUE_USAGE** **OG_PCT_USAGE**
SG	usagevalue	Each pie slice is plotted as if its data value is a percentage of the total value specified here. (Valid only is *usage* is set to OG_TOTALVALUE_USAGE.
SG	plotorder	Specifies the direction in which the data values are plotted. The value of this attribute may be one of the following built-in constants: **OG_CCW_PLOTORDER** **OG_CW_PLOTORDER**
SG	categs	Specifies whether each pie slice is labeled with the name of the category it represents.
SG	datavals	Specifies whether each pie slice is labeled with its data value.
SG	pctvalues	Specifies whether each pie slice is labeled with the percentage of the

		complete chart it represents.
SG	ticks	Specifies whether the tick marks that connect each pie slice to its label are shown.
SG	other	Specifies that pie slices that individually represent percentages less than the number entered here are combined into a single pie slice with the label "Other".
SG	nooverlap	Specifies that the labels for the pie slices should not overlap each other.
SG	catnumfmt	Specifies the number format for the category labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.
SG	catdatefmt	Specifies the date format for the category labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.
SG	valuefmt	Specifies the number format for the data value labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.
SG	pctfmt	Specifies the number format for the percent value labels. This must be a valid SQL format string. For more information, see your *Oracle7 Server SQL Reference*.

Table Frame Combined Attribute Record

```
TYPE og_tableframe_ca IS RECORD
(ca_frame  og_frame_attr,       /* generic frame */
 ca_table  og_tableframe_attr   /* table frame */
);
```

Table Frame Attribute Record

```
TYPE og_tableframe_attr IS RECORD   Mask Constants:
(mask     NUMBER(3,0),
 automin  BOOLEAN,                  OG_MIN_TABLEFRAMEA
 minrows  NUMBER(10,0),             OG_MIN_TABLEFRAMEA
 automax  BOOLEAN,                  OG_MAX_TABLEFRAMEA
 maxrows  NUMBER(10,0),             OG_MAX_TABLEFRAMEA
 cname    BOOLEAN,                  OG_CNAME_TABLEFRAMEA
 vgrid    BOOLEAN,                  OG_VGRID_TABLEFRAMEA
 hgrid    BOOLEAN,                  OG_HGRID_TABLEFRAMEA
 gridct   NUMBER(10,0)              OG_GRIDCT_TABLEFRAMEA
);
                                    OG_ALL_TABLEFRAMEA
                                    OG_NONE_TABLEFRAMEA
```

	Attribute	*Description*
SG	automin	Specifies whether the minimum number of rows that appear on the chart is set to *Auto*.
SG	minrows	Specifies the maximum number of rows that appear on the chart (if *automin* is FALSE).
SG	automax	Specifies whether the maximum number of rows that appear on the chart is set to *Auto*.
SG	maxrows	Specifies the maximum number of rows that appear on the chart (if *automax* is FALSE).
SG	colnames	Specifies whether the names of the columns appear as the first row in the chart.
SG	vgrid	Specifies whether vertical grid lines appear between the columns.
SG	hgrid	Specifies whether horizontal grid lines appear between the rows.
SG	gridct	Is the number of rows of data plotted before each horizontal grid line is drawn (if *hgrid* is set to TRUE).

Generic Attribute Record

The generic attribute record contains attributes that may be used with every object.

```
TYPE og_generic_attr IS RECORD    Mask Constants:
(mask       NUMBER(6,0),
 name       VARCHAR2(255),          OG_NAME_GENERICA
 parent     og_object,              OG_PARENT_GENERICA
 ibbox      og_rectangle,           OG_IBBOX_GENERICA
 obbox      og_rectangle,           OG_OBBOX_GENERICA
 objtype    NUMBER(2,0),            OG_OBJTYPE_GENERICA
 button     og_buttonproc,          OG_BUTTON_GENERICA
 events     NUMBER(2,0),            OG_EVENTS_GENERICA
 keycol     VARCHAR2(255,           OG_KEYCOL_GENERICA
 execquery  og_query,               OG_EXECQUERY_GENERICA
 setparam   VARCHAR2(255),          OG_SETPARAM_GENERICA
 fmttrig    VARCHAR2(255),          OG_FMTTRIG_GENERICA
 hide       BOOLEAN                 OG_HIDE_GENERICA
);
                                    OG_ALL_GENERICA
                                    OG_NONE_GENERICA
```

	Attribute	***Description***
CSG	name	Is the object's name.
CG	parent	Is the handle to the object's parent object.
G	ibbox	Is the object's inner bounding box. This is the rectangle that constitutes the object's ideal shape (i.e., connects the object's four control points), regardless of edge thickness or other attribute settings.
G	obbox	Is the object's outer bounding box. This is the smallest rectangle that completely surrounds the object. This may differ from the inner bounding box if the object has a thick edge. While the inner bounding box traces only the ideal shape of the object, the outer bounding box surrounds the entire object.
G	objtype	Is the object's type. The value of this attribute may be one of the following built-in constants: **OG_ARC_OBJTYPE** Means the object is an arc. **OG_CHART_OBJTYPE** Means the object is a chart. **OG_GROUP_OBJTYPE** Means the object is a group. **OG_IMAGE_OBJTYPE** Means the object is an image. **OG_LINE_OBJTYPE** Means the object is a line. **OG_POLY_OBJTYPE** Means the

		object is a polygon or polyline.
		OG_RECT_OBJTYPE Means the object is a rectangle.
		OG_RRECT_OBJTYPE Means the object is a rounded rectangle.
		OG_SYMBOL_OBJTYPE Means the object is a symbol.
		OG_TEXT_OBJTYPE Means the object is a text object.
CSG	button	Is the handle to the button procedure to be associated with this object. Note that the events attribute must be set properly in order to ensure that this procedure receives the desired mouse events.
CSG	events	Is the type of mouse events that the procedure specified by the button attribute should receive. The value of this attribute may be one of the built-in constants listed below. To enable the procedure to receive multiple event types, set this attribute to be the sum of the constants for the desired events.
		OG_NO_EVENTS Means the procedure receives no mouse events.
		OG_MOUSE_DOWN Means the procedure receives only mouse down events.
		OG_MOUSE_UP Means the procedure receives only mouse up events.
		OG_MOUSE_MOVE_UP Means the procedure receives only mouse move up events.
CSG	keycol	Is the column to set in a drill-down chart. This attribute applies only to chart elements.
CSG	execquery	Specifies the query to execute when the object is selected.
CSG	setparam	Is the parameter whose value is set when the object is selected.
CSG	fmttrig	Is the format trigger. This attribute applies only to chart elements.
SG	hide	Hides the specified Graphics Builder object.

Graphic Combined Attribute Record

The graphic combined attribute record consists of a generic attribute record and graphic attribute record:

```
TYPE og_graphic_ca IS RECORD
(graphic_caob  og_generic_attr,   /* generic */
 graphic_caoh  og_graphic_attr    /* graphic */
);
```

Graphic Attribute Record

The graphic attribute record contains attributes that may be used only with graphical objects:

```
TYPE og_graphic_attr IS RECORD     Mask Constants:
(mask        NUMBER(4,0),
 ewidth      NUMBER(10,0),         OG_EWIDTH_GRAPHICA
 rotang      NUMBER(5,2),          OG_ROTANG_GRAPHICA
 fecolor     VARCHAR2(255),        OG_FECOLOR_GRAPHICA
 becolor     VARCHAR2(255),        OG_BECOLOR_GRAPHICA
 edgepatt    VARCHAR2(255),        OG_EDGEPATT_GRAPHICA
 ffcolor     VARCHAR2(255),        OG_FFCOLOR_GRAPHICA
 bfcolor     VARCHAR2(255),        OG_BFCOLOR_GRAPHICA
 fillpatt    VARCHAR2(255),        OG_FILLPATT_GRAPHICA
 dashstyle   NUMBER(1,0),          OG_DASHSTYLE_GRAPHICA
 capstyle    NUMBER(2,0),          OG_CAPSTYLE_GRAPHICA
 joinstyle   NUMBER(2,0),          OG_JOINSTYLE_GRAPHICA
 transfer    NUMBER(1,0)           OG_TRANSFER_GRAPHICA
 bevelstyle  NUMBER(2,0)           OG_BEVELSTYLE_GRAPHICA
);
                                   OG_ALL_GRAPHICA
                                   OG_NONE_GRAPHICA
```

	Attribute	*Description*
CSG	ewidth	Is the width of the object's edge (in layout units).
CSG	rotang	Is the object's rotation angle. The angle at which the object is initially created is considered to be 0, and this attribute is the number of degrees clockwise the object currently differs from that initial angle. You can rotate an object to an absolute angle by setting this attribute, or use the OG_ROTATE procedure to rotate an object by a relative amount. (Note that when you use OG_ROTATE to rotate an object, the rotang attribute will automatically be updated to reflect the new absolute angle.)
CSG	fecolor	Is the object's foreground edge color. For more information about valid color palettes, see Default color palettes .
CSG	becolor	Is the object's background edge color. For more information about valid color names, see Default color palettes .

CSG	edgepatt	Is the object's edge pattern. For more information about valid pattern names, see Pattern palette .
CSG	ffcolor	Is the object's foreground fill color. For more information about valid color names, see Default color palettes .
CSG	bfcolor	Is the object's background fill color. For more information about valid color names, see Default color palettes .
CSG	fillpatt	Is the object's fill pattern. For more information about valid pattern names, see Pattern palette.
CSG	dashstyle	Is the dash style of the object's edge. The value of this attribute may be one of the following built-in constants: **OG_SOLID_DSTYLE** Means the line is solid. **OG_DOT_DSTYLE** Means the line is dotted. **OG_LONG_DSTYLE** Means the line is a series of long dashes. **OG_DASHDOT_DSTYLE** Means the line is a series of dashes followed by dots. **OG_DOTDOT_DSTYLE** Means the line is a series of two consecutive dots. **OG_SHORT_DSTYLE** Means the line is a series of short dashes. **OG_DASHDOTDOT_DSTYLE** Means the line is a series of a dash followed by two dots.
CSG	capstyle	Is the cap style of the object's edge. The value of this attribute may be one of the following built-in constants: **OG_BUTT_CSTYLE** Means the cap style is butt. **OG_PROJECTING_CSTYLE** Means the cap style is projecting. **OG_ROUND_CSTYLE** Means the cap style is round.
CSG	joinstyle	Is the join style of the object's edge. The value of this attribute may be one of the following built-in constants: **OG_MITRE_JSTYLE** Means the join style is metre. **OG_BEVEL_JSTYLE** Means the join style is bevel. **OG_ROUND_JSTYLE** Means the join style is round.
CSG	transfer	Is the object's transfer mode. The value of this attribute may be one of the following built-in constants: **OG_COPY_TRANSFER** Means the

		transfer mode is *copy*. **OG_REVCOPY_TRANSFER** Means the transfer mode is *reverse copy*. **OG_OR_TRANSFER** Means the transfer mode is *or*. **OG_REVOR_TRANSFER** Means the transfer mode is *reverse or*. **OG_CLEAR_TRANSFER** Means the transfer mode is *clear*. **OG_REVCLEAR_TRANSFER** Means the transfer mode is *reverse clear*. **OG_INVERT_TRANSFER** Means the transfer mode is *invert*. **OG_BACKINVERT_TRANSFER** Means the transfer mode is *background invert*.
CSG	bevelstyle	Is the object's bevel style. The value of this attribute may be one of the following built-in constants: **OG_INSET_BSTYLE** Means the bevel is inset. **OG_LOWERED_BSTYLE** Means the bevel is lowered. **OG_OUTSET_BSTYLE** Means the bevel is outset. **OG_PLAIN_BSTYLE** Means the object has no bevel. **OG_RAISED_BSTYLE** Means the bevel is raised.

Group Combined Attribute Record

The group combined attribute record consists of a generic attribute record and group attribute record:

```
TYPE og_group_ca IS RECORD
(group_caob  og_generic_attr,   /* generic */
 group_caog  og_group_attr      /* group */
);
```

Group Attribute Record

The group attribute record contains attributes that may be used only with group objects:

```
TYPE og_group_attr IS RECORD          Mask Constants:
(mask        NUMBER(1,0),
 childcount  NUMBER(10,0),            OG_CHILDCOUNT_GROUPA
 clipflag    BOOLEAN                  OG_CLIPFLAG_GROUPA
);
```

		OG_ALL_GROUPA OG_NONE_GROUPA
	Attribute	***Description***
G	childcount	Is the number of children that belong to the group object. If another group object is a child of the group being checked, that object will be counted only as one object.
CSG	clipflag	Specifies whether the first object in the group is a rectangle object that should be used as a clipping rectangle. If TRUE, the only members of the group that appear on the layout are those objects-or portions of those objects-that appear within the bounds of the clipping rectangle. The rectangle object itself also appears. The value of this attribute may be one of the following: **TRUE** Means the first object in the group is treated as a clipping rectangle. **FALSE** Means the first object in the group is not treated as a clipping rectangle.

Image Combined Attribute Record

The image combined attribute record consists of a generic attribute record and image attribute record:

```
TYPE og_image_ca IS RECORD
(image_caob  og_generic_attr,   /* generic */
 image_caoi  og_image_attr      /* image */
);
```

Image Attribute Record

The image attribute record contains attributes that may be used only with image objects:

```
TYPE og_image_attr IS RECORD            Mask Constants:
(mask         NUMBER(3,0),
 cliprect     og_rectangle,             OG_CLIPRECT_IMAGEA
 upperleft    og_point,                 OG_UPPERLEFT_IMAGEA
 width        NUMBER(10,0),             OG_SIZE_IMAGEA
 height       NUMBER(10,0),             OG_SIZE_IMAGEA
 query        og_query,                 OG_DATA_IMAGEA
 which_data   NUMBER(1,0),              OG_DATA_IMAGEA
 colname      VARCHAR2(255),            OG_DATA_IMAGEA
 quality      NUMBER(5,0),              OG_QUALITY_IMAGEA
 dither       BOOLEAN                   OG_DITHER_IMAGEA
);
                                        OG_ALL_IMAGEA
                                        OG_NONE_IMAGEA
```

	Attribute	*Description*
SG	cliprect	Is the x- and y-coordinates, height, and width of the image's clipping rectangle (in layout units). Only the portion of the image that falls within this clipping rectangle will be displayed. If this attribute is not specified, the clipping rectangle will equal the full dimensions of the image.
SG	upperleft	Is the x- and y-coordinates of the image's upper-left corner (in layout units).
SG	width	Is the image's width (in layout units). If you set this attribute to some value other than the image's default width, the image will be scaled to fit within the new width.
SG	height	Is the image's height (in layout units). If you set this attribute to some value other than the image's default height, the image will be scaled to fit within the new height.

C	query	Is the handle to the query that retrieves the image from a table in a database. Note that this table must be a user table, and not one the private tables used by Graphics Builder when you save or export a display, drawing, chart template, color palette, image, or sound to the database. Only Oracle Format images can be stored in the database.
C	which_data	Specifies whether the image to be created is contained in a query's new or old data set. Graphics Builder provides two built-in numeric constants that may be used as values for this attribute: **OG_NEWDATA** Means the image is contained in the query's new data set. **OG_OLDDATA** Means the image is contained in the query's old data set.
C	colname	Is the name of the query column that contains the image data. The image that is created is the one contained in the query cell at the intersection of the column specified by this attribute and the row pointed to by the query's cursor.
CSG	quality	Specifies with what quality the image is drawn. Higher quality images look better, but require more processing time to manipulate (e.g., draw, move, scale, etc.). The value of this attribute may be one of the following built-in constants: **OG_HIGH_IQUALITY** Means the quality is high. **OG_MED_IQUALITY** Means the quality is medium. **OG_LOW_IQUALITY** Means the quality is low.
CSG	dither	Specifies whether Graphics Builder dithers the image when displaying it. The value of this attribute may be one of the following: **TRUE** Means dither the image. **FALSE** Means do not dither the image.

Line Combined Attribute Record

The line combined attribute record consists of a generic attribute record, graphic attribute record, and line attribute record:

```
TYPE og_line_ca IS RECORD
(line_caob  og_generic_attr,   /* generic */
 line_caoh  og_graphic_attr,   /* graphic */
 line_caol  og_line_attr       /* line */
);
```

Line Attribute Record

The line attribute record contains attributes that may be used only with line objects:

```
TYPE og_line_attr IS RECORD
(mask         NUMBER(1,0),
 startpt      og_point,
 endpt        og_point,
 arrowstyle   NUMBER(1,0)
);
```

Mask Constants:

OG_STARTPT_LINEA
OG_ENDPT_LINEA
OG_ARROWSTYLE_LINEA

OG_ALL_LINEA
OG_NONE_LINEA

	Attribute	***Description***
CSG	startpt	Is the x- and y-coordinates of the line's starting point (in layout units).
CSG	endpt	Is the x- and y-coordinates of the line's end point (in layout units).
CSG	arrowstyle	Is the line's arrow style. The value of this attribute may be one of the following built-in constants: **OG_NOARROW_ASTYLE** Means the line has no arrow. **OG_START_ASTYLE** Means the line has an arrow at its starting point. **OG_END_ASTYLE** Means the line has an arrow at its end point. **OG_BOTH_ASTYLE** Means the line has an arrow at both ends. **OG_MIDTOSTART_ASTYLE** Means the line has an arrow at its middle, pointing toward its starting point. **OG_MIDTOEND_ASTYLE** Means the line has an arrow at its middle, pointing toward its end point.

Polygon Combined Attribute Record

The polygon combined attribute record consists of a generic attribute record, graphic attribute record, and polygon attribute record:

```
TYPE og_poly_ca IS RECORD
(poly_caob  og_generic_attr,   /* generic */
 poly_caoh  og_graphic_attr,   /* graphic */
 poly_caop  og_poly_attr       /* polygon */
);
```

Polygon Attribute Record

The polygon attribute record contains attributes that may be used only with polygon objects:

```
TYPE og_poly_attr IS RECORD
(mask     NUMBER(1,0),
 pointct  NUMBER(10,0),
 closed   BOOLEAN
);
```

Mask Constants:

OG_POINTCT_POLYA
OG_CLOSED_POLYA

OG_ALL_POLYA
OG_NONE_POLYA

	Attribute	***Description***
G	pointct	Is the number of points that compose the polygon object.
CSG	closed	Is the closure of the polygon. The value of this attribute may be one of the following: **TRUE** Means the polygon is closed. **FALSE** Means the polygon is open.

Printer Attribute Record

```
TYPE og_printer_attr IS RECORD
(mask        NUMBER(3,0),
 name        VARCHAR2(255),
 landscape   BOOLEAN,
 startpage   NUMBER(5,0),
 endpage     NUMBER(5,0),
 width       NUMBER(10,0),
 height      NUMBER(10,0),
 copies      NUMBER(5,0),
 printfile   VARCHAR2(255)
);
```

Mask Constants:

OG_NAME_PRINTERA
OG_LANDSCAPE_PRINTERA
OG_STARTPAGE_PRINTERA
OG_ENDPAGE_PRINTERA
OG_WIDTH_PRINTERA
OG_HEIGHT_PRINTERA
OG_COPIES_PRINTERA
OG_PRINTFILE_PRINTERA

OG_ALL_PRINTERA
OG_NONE_PRINTERA

	Attribute	***Description***
SG	name	Is the name of the current printer.
SG	landscape	Specifies whether the display is printed in landscape or portrait mode.
SG	startpage	Is the first page to print.
SG	endpage	Is the last page to print.
S	width	Is the page width.
S	height	Is the page height.
SG	copies	Is the number of copies to print.
SG	printfile	Is the name of the PostScript file to print to. If this property is NULL, the output is sent to the printer.

Query Attribute Record

The query attribute record contains attributes that may be used only with queries:

```
TYPE og_query_attr IS RECORD        Mask Constants:
(mask          NUMBER(4,0),
 name          VARCHAR2(255),       OG_NAME_QUERYA
 dateformat    VARCHAR2(255),       OG_DATEFORMAT_QUERYA
 querysource   VARCHAR2(2000),      OG_QUERYSOURCE_QUERYA
 querytype     NUMBER(1,0),         OG_QUERYTYPE_QUERYA
 cachetype     NUMBER(1,0),         OG_CACHETYPE_QUERYA
 maxflag       BOOLEAN,             OG_MAXFLAG_QUERYA
 maxrows       NUMBER(10,0),        OG_MAXROWS_QUERYA
 execopen      BOOLEAN,             OG_EXECOPEN_QUERYA
 exectimer     VARCHAR2(255),       OG_EXECTIMER_QUERYA
 execalert     VARCHAR2(255),       OG_EXECALERT_QUERYA
 customproc    VARCHAR2(255),       OG_CUSTOMPROC_QUERYA
 postproc      VARCHAR2(255)        OG_POSTPROC_QUERYA
);
                                    OG_ALL_QUERYA
                                    OG_NONE_QUERYA
```

	Attribute	*Description*
CSG	name	Is the name of the query.
CSG	dateformat	Is the date format mask for the query.
CSG	querysource	Is the source of the query's data. If the data comes from a database, this attribute should contain the text of the the query's SQL SELECT statement. If the data is stored in the filesystem, this attribute should contain the path and name of the data file.
CSG	querytype	Is the type of query. The value of this attribute may be one of the following built-in constants: **OG_CUSTOM_QTYPE** Means the query is a Custom query. **OG_EXSQL_QTYPE** Means the query retrieves its data from a text file that contains a SQL SELECT statement. **OG_PRN_QTYPE** Means the query is based on a PRN file. **OG_SQL_QTYPE** Means the query is a SQL SE.LECT statement. **OG_SYLK_QTYPE** Means the query is based on a SYLK file. **OG_WKS_QTYPE** Means the query is based on a WKS file.
CSG	cachetype	Determines how the newly retrieved data from a query execution is treated.

		The value of this attribute may be one of the following built-in constants: **OG_APPEND_CACHETYPE** Means all of the existing rows of data are retained, and the new rows of data are added to the bottom of the existing data set. **OG_COPY_CACHETYPE** Means all of the data from the previous execution is copied to a special buffer, and the newly retrieved data replaces it. **OG_NONE_CACHETYPE** Means all of the data from the previous execution is discarded, and the newly retrieved data replaces it.
CSG	maxflag	Specifies whether a limit is placed on the number of rows contained in the data set.
CSG	maxrows	Specifies the maximum number of rows of data that are retained in the query's data set.
CSG	execopen	Specifies whether the query is automatically executed when the display is opened at runtime.
CSG	exectimer	Is the name of the timer on which the query executes.
CSG	execalert	*Reserved for future use.*
CSG	customproc	Is the PL/SQL procedure that is invoked when a Custom query is executed.
CSG	postproc	Is the PL/SQL procedure that is invoked after the query is executed.

Rectangle Combined Attribute Record

The rectangle combined attribute record consists of a generic attribute record, graphic attribute record, and rectangle attribute record:

```
TYPE og_rect_ca IS RECORD
(rect_caob  og_generic_attr,   /* generic */
 rect_caoh  og_graphic_attr,   /* graphic */
 rect_caor  og_rect_attr       /* rectangle */
);
```

Rectangle Attribute Record

The rectangle attribute record contains attributes that may be used only with rectangle objects:

```
TYPE og_rect_attr IS RECORD
(mask      NUMBER(1,0),
 baserect  og_rectangle
);
```

Mask Constants:

OG_BASERECT_RECTA

OG_ALL_RECTA
OG_NONE_RECTA

	Attribute	*Description*
CSG	baserect	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).

Reference Line Attribute Record

```
TYPE og_refline_attr IS RECORD
(mask         NUMBER(2,0),
 numvalue     NUMBER(6),
 datevalue    DATE,
 label        VARCHAR2(255),
 axis         NUMBER(1,0)
);
```

Mask Constants:

OG_VALUE_REFLINEA
OG_VALUE_REFLINEA
OG_LABEL_REFLINEA
OG_AXIS_REFLINEA

OG_ALL_REFLINEA
OG_NONE_REFLINEA

	Attribute	***Description***
SG	numvalue	Is the number value at which the reference line appears.
SG	datevalue	Is the date value at which the reference line appears.
SG	label	Is the text label that identifies the reference line in the legend.
SG	axis	Specifies which axis the reference value is compared to determine its position.

Rounded Rectangle Combined Attribute Record

The rounded rectangle combined attribute record consists of a generic attribute record, graphic attribute record, and rounded rectangle attribute record:

```
TYPE og_rrect_ca IS RECORD
(rrect_caob  og_generic_attr,   /* generic */
 rrect_caoh  og_graphic_attr,   /* graphic */
 rrect_caor  og_rrect_attr      /* rounded rectangle */
);
```

Rounded Rectangle Attribute Record

The rounded rectangle attribute record contains attributes that may be used only with rounded rectangle objects:

```
TYPE og_rrect_attr IS RECORD
(mask      NUMBER(1,0),
 baserect  og_rectangle,
 corner    og_point
);
```

Mask Constants:

OG_BASERECT_RRECTA
OG_CORNER_RRECTA

OG_ALL_RRECTA
OG_NONE_RRECTA

	Attribute	***Description***
CSG	baserect	Is the x- and y-coordinates of the upper-left corner, and the height and width of the rectangle used as the basis for the rectangle object (in layout units).
CSG	corner	Is the x- and y-radii (in layout units) of the ellipse that would result if the arcs that form the rounded corners were continued to follow a full 360 degree path.

Sound Attribute Record

The sound attibute record contains attributes that may be used only with sounds.

```
TYPE og_sound_attr IS RECORD
(mask        NUMBER(1,0),
 query       og_query,
 which_data  NUMBER(1,0),
 colname     VARCHAR2(255),
 name        VARCHAR2(255),
);
```

Mask Constants:

OG_DATA_SOUNDA
OG_DATA_SOUNDA
OG_DATA_SOUNDA
OG_NAME_SOUNDA

OG_ALL_SOUNDA
OG_NONE_SOUNDA

	Attribute	*Description*
C	query	Is the handle to the query that retrieves the sound from a table in a database. Note that this table must be a user table, and *not* one the private tables used by Graphics Builder when you save or export a display, drawing, chart template, color palette, image, or sound to the database.
C	which_data	Specifies whether the sound to be created is contained in a query's new or old data set. Graphics Builder provides two built-in numeric constants that may be used as values for this attribute: **OG_NEWDATA** Means the sound is contained in the query's new data set. **OG_OLDDATA** Means the sound is contained in the query's old data set.
C	colname	Is the name of the query column that contains the sound data. The sound that is created is the one contained in the query cell at the intersection of the column specified by this attribute and the row pointed to by the query's cursor.
CSG	name	Is the name of the sound.

Symbol Combined Attribute Record

The symbol combined attribute record consists of a generic attribute record, graphic attribute record, and symbol attribute record:

```
TYPE og_symbol_ca IS RECORD
(symbol_caob  og_generic_attr,   /* generic */
 symbol_caoh  og_graphic_attr,   /* graphic */
 symbol_caos  og_symbol_attr     /* symbol */
);
```

Symbol Attribute Record

The arc attribute record contains attributes that may be used only with arc objects:

```
TYPE og_symbol_attr IS RECORD
(mask     NUMBER(1,0),
 center   og_point,
 indx     NUMBER(3,0),
 symsize  NUMBER(1,0)
);
```

Mask Constants:

OG_CENTER_SYMBOLA
OG_INDX_SYMBOLA
OG_SYMSIZE_SYMBOLA

OG_ALL_SYMBOLA
OG_NONE_SYMBOLA

	Attribute	***Description***
CSG	center	Is the x- and y-coordinates of the symbol's center (in layout units).
CSG	indx	Is the index (or number) of the symbol's position as it appears in the symbol palette in the Designer.
CSG	symsize	Is the symbol's size. The value of this attribute may be one of the following built-in constants: **OG_LARGE_SYMSIZE** Means the symbol is large. **OG_MEDIUM_SYMSIZE** Means the symbol is medium. **OG_SMALL_SYMSIZE** Means the symbol is small.

Text Attributes Overview

The text attribute record does not contain the text that will appear in the text object. Instead, you must first create a text object, and then use the OG_INSERT_CMPTEXT procedure to insert a "compound text element" into the text object. You may insert multiple compound text elements into a text object, and each one will represent one line of text in the object. In addition, each compound text element may contain one or more "simple text elements." A simple text element contains an actual text string, and must be inserted into a compound text element with the OG_INSERT_SMPTEXT procedure.
The attribute records for compound and simple text are listed below.

Text Combined Attribute Record

The text combined attribute record consists of a generic attribute record, graphic attribute record, and text attribute record:

```
TYPE og_text_ca IS RECORD
(text_caob  og_generic_attr,   /* generic */
 text_caoh  og_graphic_attr,   /* graphic */
 text_caot  og_text_attr       /* text */
);
```

Text Attribute Record

The text attribute record contains attributes that may be used only with text objects:

```
TYPE og_text_attr IS RECORD                  Mask Constants:
(mask       NUMBER(6,0),
 origin     og_point,                        OG_ORIGIN_TEXTA
 ctcount    NUMBER(10,0),                    OG_CTCOUNT_TEXTA
 gfont      og_font_attr,                    OG_GFONT_TEXTA
 gcolor     VARCHAR2(255),                   OG_GCOLOR_TEXTA
 spacing    NUMBER(1,0),                     OG_SPACING_TEXTA
 custom     NUMBER(10,0),                    OG_SPACING_TEXTA
 horigin    NUMBER(1,0),                     OG_HORIGIN_TEXTA
 vorigin    NUMBER(1,0),                     OG_VORIGIN_TEXTA
 halign     NUMBER(2,0),                     OG_HALIGN_TEXTA
 valign     NUMBER(3,0),                     OG_VALIGN_TEXTA
 fixed      BOOLEAN,                         OG_FIXED_TEXTA
 wrap       BOOLEAN,                         OG_WRAP_TEXTA
 bbscale    BOOLEAN,                         OG_BBSCALE_TEXTA
 fontscale  BOOLEAN,                         OG_FONTSCALE_TEXTA
 invisible  BOOLEAN,                         OG_INVISIBLE_TEXTA
 width      NUMBER(10,0),                    OG_FIXEDWH_TEXTA
 height     NUMBER(10,0)                     OG_FIXEDWH_TEXTA
);
                                             OG_ALL_TEXTA
                                             OG_NONE_TEXTA
```

	Attribute	*Description*
CSG	origin	Is the x- and y-coordinates of the text object's upper-left corner (in layout units).
G	ctcount	Is the number of compound text elements that compose the text object.
S	gfont	Is the text object's global font. When this attribute is set, the *font* attribute for every simple text element in the text object will be set to this font. Note that setting this attribute will affect existing simple text elements only; any simple text elements added later will appear in the font specified in their simple text attribute records.
S	gcolor	Is the text object's global color. When this attribute is set, the *color* attribute for every simple text element in the text object will be set to this color. Note that setting this attribute will affect existing simple text elements only; any simple text elements added later will appear in the color specified in their simple text attribute records.
CSG	spacing	Is the line spacing for the text object. The value of this attribute may be one of the built-in constants listed below. If custom spacing is set, the value of the *custom* attribute should specify the exact spacing amount. **OG_SINGLE_SPACE** Means the text use single line spacing. **OG_ONEHALF_SPACE** Means the text used 1-1/2 line spacing. **OG_DOUBLE_SPACE** Means the text uses double line spacing. **OG_CUSTOM_SPACE** Means the text uses custom line spacing. The actual spacing used is defined in the *custom* attribute.
CSG	custom	Is the custom spacing for the text object (in layout units). This attribute is used to specify spacing only if the *gspacing* attribute is set to custom spacing.
CSG	horigin	Is the horizontal position of the text object relative to its origin point. The value of this attribute may be one of the following built-in constants: **OG_LEFT_HORIGIN** Means the origin point lies along the left edge of the bounding box. **OG_CENTER_HORIGIN** Means

		the origin point lies equally between the left and right edges of the bounding box. **OG_RIGHT_HORIGIN** Means the origin point lies along the right edge of the bounding box.
CSG	vorigin	Is the vertical position of the text object relative to its origin point. The value of this attribute may be one of the following built-in constants: **OG_TOP_VORIGIN** Means the origin point lies along the top edge of the bounding box. **OG_MIDDLE_VORIGIN** Means the origin point lies equally between the top and bottom edges of the bounding box. **OG_BOTTOM_VORIGIN** Means the origin point lies along the bottom edge of the bounding box.
CSG	halign	Is the horizontal alignment of the text object. The value of this attribute may be one of the following built-in constants: **OG_LEFT_HALIGN** Means the text is left-aligned. **OG_CENTER_HALIGN** Means the text is center-aligned. **OG_RIGHT_HALIGN** Means the text is right-aligned.
CSG	valign	Is the vertical alignment of the text object. The value of this attribute may be one of the following built-in constants: **OG_TOP_VALIGN** Means the text is top-aligned. **OG_MIDDLE_VALIGN** Means the text is middle-aligned. **OG_BOTTOM_VALIGN** Means the text is bottom-aligned.
CSG	wrap	Specifies whether the text should "wrap" to fit into the text object's bounding box. As described below, a compound text element represents a line of text, and is made up of simple text elements. The value of this attribute may be one of the following: **TRUE** Means wrap the text. **FALSE** Means do not wrap the text.
CSG	bbscale	Specifies whether the text object's bounding box should be scaled when the text object is scaled. The value of this attribute may be one of the

		following: **TRUE** Means scale the bounding box. **FALSE** Means do not scale the bounding box.
CSG	fontscale	Specifies whether the point size of the font should be scaled when the text object is scaled. The value of this attribute may be one of the following: **TRUE** Means scale the point size. **FALSE** Means do not scale the point size.
CSG	fixed	Specifies whether the text object's bounding box should remain a fixed size. If this attribute is TRUE, the values of the *width* and *height* attributes should specify the size of the bounding box. The value of this attribute may be one of the following: **TRUE** Means the bounding box is fixed. The dimensions of the bounding box are defined in the *width* and *height* attributes. **FALSE** Means the bounding box is not fixed.
CSG	width	Is the width of the bounding box (in layout units). Whenever the bounding box changes, this attribute will automatically be updated to reflect the new width. This attribute is used to set the width only if the *fixed* attribute is TRUE.
CSG	height	Is the height of the bounding box (in layout units). Whenever the bounding box changes, this attribute will automatically be updated to reflect the new height. This attribute is used to set the height only if the *fixed* attribute is TRUE.
CSG	invisible	Specifies whether the text in the text object should be invisible. This is useful for text fields in which a user enters a password, if you don't want the password to be seen. The value of this attribute may be one of the following: **TRUE** Means the text is invisible. **FALSE** Means the text is visible.

Font Attribute Record

The font attribute record is used to specify the properties of a font, such as typeface and point size.

```
TYPE og_font_attr IS RECORD
(mask          NUMBER(3,0),
 typeface      VARCHAR2(255),
 ptsize        NUMBER(10,2),
 style         NUMBER(5,0),
 weight        NUMBER(5,0),
 width         NUMBER(5,0),
 kerning       BOOLEAN,
 nearest       BOOLEAN,
 synthesize    BOOLEAN,
 charset       NUMBER(5,0)
);
```

Mask Constants:

OG_TYPEFACE_FONTA
OG_PTSIZE_FONTA
OG_STYLE_FONTA
OG_WEIGHT_FONTA
OG_WIDTH_FONTA
OG_KERNING_FONTA
OG_NEAREST_FONTA
OG_SYNTHESIZE_FONTA
OG_CHARSET_FONTA

OG_ALL_FONTA
OG_NONE_FONTA

	Attribute	***Description***
CG	typeface	Is the font's style. Values for this field specify styles such as italic, shadow, and underline, and are system-specific. For more information, consult your system administrator or your system documentation.
CG	ptsize	Is the font's point size. Values for this field are system-specific. For more information, consult your system administrator or your system documentation.
CG	style	Is the font's style. Not all styles are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants: **OG_BLINK_FONTSTYLE** Means the style is blinking. **OG_INVERTED_FONTSTYLE** Means the style is inverted. **OG_ITALIC_FONTSTYLE** Means the style is italic. **OG_OBLIQUE_FONTSTYLE** Means the style is oblique. **OG_OUTLINE_FONTSTYLE** Means the style is outline. **OG_OVERSTRIKE_FONTSTYLE** Means the style is overstrike.

		OG_PLAIN_FONTSTYLE Means the style is plain. **OG_SHADOW_FONTSTYLE** Means the style is shadow. **OG_UNDERLINE_FONTSTYLE** Means the style is underline. **OG_UNKNOWN_FONTSTYLE** Means the style is unknown. You cannot *set* a style to this value; however, if you *get* a font and Graphics Builder cannot determine its style, this value is returned.
CG	weight	Is the font's weight. Not all weights are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants: **OG_BOLD_FONTWEIGHT** Means the weight is bold. **OG_DEMIBOLD_FONTWEIGHT** Means the weight is demibold. **OG_DEMILIGHT_FONTWEIGHT** Means the weight is demilight. **OG_EXTRABOLD_FONTWEIGHT** Means the weight is extra bold. **OG_EXTRALIGHT_FONTWEIGHT** Means the weight is extra light. **OG_LIGHT_FONTWEIGHT** Means the weight is light. **OG_MEDIUM_FONTWEIGHT** Means the weight is medium. **OG_ULTRABOLD_FONTWEIGHT** Means the weight is ultrabold. **OG_ULTRALIGHT_FONTWEIGHT** Means the weight is ultralight. **OG_UNKNOWN_FONTWEIGHT** Means the weight is unknown. You cannot *set* a weight to this value; however, if you *get* a font and Graphics Builder cannot determine its weight, this value is returned.
CG	width	Is the font's width. Not all widths are available on all systems. For more information, consult your system administrator or your system documentation. The value of this field may be one of the following built-in constants: **OG_DENSE_FONTWIDTH** Means the width is dense. **OG_EXPAND_FONTWIDTH**

		Means the width is expanded.
		OG_EXTRADENSE_FONTWIDTH Means the width is extra dense.
		OG_EXTRAEXPAND_FONTWIDTH Means the width is extra expanded.
		OG_NORMAL_FONTWIDTH Means the width is normal.
		OG_SEMIDENSE_FONTWIDTH Means the width is semidense.
		OG_SEMIEXPAND_FONTWIDTH Means the width is semiexpanded.
		OG_ULTRADENSE_FONTWIDTH Means the width is ultradense.
		OG_ULTRAEXPAND_FONTWIDTH Means the width is ultraexpanded.
		OG_UNKNOWN_FONTWIDTH Means the width is unknown. You cannot *set* a weight to this value; however, if you *get* a font and Graphics Builder cannot determine its width, this value is returned.
CG	kerning	Specifies whether the font should be kerned. Kerning is the adjustment of the space between adjacent letters to improve the readability of the text. The value of this field may be one of the following:
		TRUE Means kern the font.
		FALSE Means do not kern the font.
C	nearest	Specifies whether Graphics Builder should substitute the nearest matching font if the exact font specified cannot be found. The precedence for finding the nearest font is typeface, point size, style, weight, and width (meaning that Graphics Builder first tries to find the specified typeface, then size, etc.). The value of this attribute may be one of the following:
		TRUE Means substitute the nearest font.
		FALSE Means do not substitute the nearest font.
C	synthesize	Specifies whether Graphics Builder should try to synthesize the desired font (if the specified font cannot be found) by transforming the nearest-matching font. The value of this field may be one of the following:
		TRUE Means synthesize the font.
		FALSE Means do not synthesize the font.
CG	charset	Is the font's character set. Values for

this field specify character sets such as U.S. ASCII, Kanji, and Arabic. For a list of valid values for this field, see the Graphics Builder documentation for your operating system.
OG_US7ASCII_CHARSET
OG_WE8DEC_CHARSET
OG_WE8HP_CHARSET
OG_US8PC437_CHARSET
OG_WE8EBCDIC37_CHARSET
OG_WE8EBCDIC500_CHARSET
OG_WE8PC850_CHARSET
OG_D7DEC_CHARSET
OG_F7DEC_CHARSET
OG_S7DEC_CHARSET
OG_E7DEC_CHARSET
OG_SF7ASCII_CHARSET
OG_NDK7DEC_CHARSET
OG_I7DEC_CHARSET
OG_NL7DEC_CHARSET
OG_CH7DEC_CHARSET
OG_SF7DEC_CHARSET
OG_WE8ISO8859P1_CHARSET
OG_EE8ISO8859P2_CHARSET
OG_SE8ISO8859P3_CHARSET
OG_NEE8ISO8859P4_CHARSET
OG_CL8ISO8859P5_CHARSET
OG_AR8ISO8859P6_CHARSET
OG_EL8ISO8859P7_CHARSET
OG_IW8ISO8859P8_CHARSET
OG_WE8ISO8859P9_CHARSET
OG_AR8ASMO708PLUS_CHARSET
OG_AR7ASMO449PLUS_CHARSET
OG_WE8MACROMAN8_CHARSET
OG_JVMS_CHARSET
OG_JEUC_CHARSET
OG_JDEC_CHARSET
OG_SJIS_CHARSET
OG_JDBCS_CHARSET
OG_JHP_CHARSET
OG_KSC5601_CHARSET
OG_KIBM5540_CHARSET
OG_KDBCS_CHARSET
OG_CGB231380_CHARSET
OG_CDBCS_CHARSET
OG_BIG5_CHARSET
OG_CNS1164386_CHARSET

Compound Text Element Attribute Record

The compound text element attribute record contains attributes that may be used only with compound text elements:

```
TYPE og_cmptext_attr IS RECORD
(mask     NUMBER(1,0),
 stcount  NUMBER(10,0)
);
```

Mask Constants:

OG_STCOUNT_CMPTEXTA

OG_ALL_CMPTEXTA
OG_NONE_CMPTEXTA

	Attribute	***Description***
G	stcount	Is the number of simple text elements that compose the compound text element.

Simple Text Element Attribute Record

The simple text element attribute record contains attributes that may be used only with simple text elements:

```
TYPE og_smptext_attr IS RECORD
(mask   NUMBER(1,0),
 str    VARCHAR2(2000)
 font   og_font_attr,
 color  VARCHAR2(255)
);
```

Mask Constants:

OG_STR_SMPTEXTA
OG_FONT_SMPTEXTA
OG_COLOR_SMPTEXTA

OG_ALL_SMPTEXTA
OG_NONE_SMPTEXTA

	Attribute	***Description***
CSG	str	Is the character string containing the actual text for the simple text element.
CSG	font	Is the font in which the character string's text should be displayed. The only font attributes that will be used are those specified by the value of the *mask* attribute(s) in the font attribute record. Fields in the attribute record for which the mask is not set will be unaffected.
CSG	color	Is the color in which the character string's text should be displayed. Note that this is the color for the text itself. To set the text object's edge or fill colors, change the text object's graphic attributes.

Example

This procedure creates a text object named "Message" at origin point (1", 1"), and contains the following two lines of text in a 12-point Times font.

This is line 1.

And now line 2.

Remember that each compound text element represents exactly one line of text in the text object.

```
PROCEDURE make_text IS
   text_obj   og_object;
   text_rec   og_text_ca;
   smp_rec    og_smptext_attr;
   font_rec   og_font_attr;
BEGIN
/* Set text object's name and origin attributes */
   text_rec.text_caob.name:='Message';
   text_rec.text_caot.origin.x:=OG_INCH;
   text_rec.text_caot.origin.y:=OG_INCH;
   text_rec.text_caob.mask:=OG_NAME_GENERICA;
   text_rec.text_caoh.mask:=OG_NONE_GRAPHICA;
   text_rec.text_caot.mask:=OG_ORIGIN_TEXTA;
/* Make the text object */
   text_obj:=og_make(text_rec);
```

```
/* Insert new compound text element into the text object at
      index 0 */
   og_insert_cmptext(text_obj, 0);
/* Set font record's typeface and point size attributes */
   font_rec.typeface:='times';
   font_rec.ptsize:=12;
   font_rec.mask:=OG_TYPEFACE_FONTA+
                  OG_PTSIZE_FONTA;
/* Set simple text record for text string and font */
   smp_rec.str:='This is line 1.';
   smp_rec.font:=font_rec;
   smp_rec.mask:=OG_STR_SMPTEXTA+
                 OG_FONT_SMPTEXTA;
/* Insert a new simple text element at index 0 in text
      object's compound text element at index 0, using
      defined simple text record */
   og_insert_smptext(text_obj, smp_rec, 0, 0);
/* Insert new compound text element into the text object at
      index 1 */
   og_insert_cmptext(text_obj, 1);
/* Change the simple text record's text string */
   smp_rec.str:='And now';
/* Insert a new simple text element at index 0 in text
      object's compound text element at index 1, using
      defined simple text record */
   og_insert_smptext(text_obj, smp_rec, 1, 0);
/* Change the simple text record's text string */
   smp_rec.str:=' line 2.';
/* Insert a new simple text element at index 1 in text
      object's compound text element at index 1, using
      defined simple text record */
   og_insert_smptext(text_obj, smp_rec, 1, 1);
END;
```

Example

This function takes a handle to a text field object as an argument, and returns the text contained in that field. Note that since only the compound text element is accessed, only the text field's first line of text is retrieved.

```
FUNCTION get_text(text_obj IN og_object) RETURN VARCHAR2 IS
   smp_rec   og_smptext_attr;
BEGIN
   /* Set the simple text record's mask, indicating that the text string
is the only attribute to get */
   smp_rec.mask:=OG_STR_SMPTEXTA;
   /* Get the 0th simple text element in the text object's
      0th compound text element, and store the results in
      the simple text record */
   og_get_smptext(text_obj, 0, 0, smp_rec);
   /* Return the text string attribute of the simple text
      record */
   RETURN(smp_rec.str);
END;
```

Timer Attributes

The timer attribute record contains attributes that may be used only with timers:

```
TYPE og_timer_attr IS RECORD
(mask       NUMBER(2,0),
 name       VARCHAR2(255),
 interval   NUMBER(10,3),
 timerproc  VARCHAR2(255),
 active     BOOLEAN
);
```

Mask Constants:

OG_NAME_TIMERA
OG_INTERVAL_TIMERA
OG_TIMERPROC_TIMERA
OG_ACTIVE_TIMERA

OG_ALL_TIMERA
OG_NONE_TIMERA

	Attribute	***Description***
CSG	name	Is the name of the timer.
CSG	interval	Is the number of seconds that will pass between each execution of the timer procedure.
CSG	timerproc	Is the name of the procedure that will be executed when the timer is fired.

Window Attribute Record

The position and dimensions of windows are expressed in "screen resolution units," more commonly known as pixels. Both the horizontal and vertical values of the screen resolution are provided in a built-in global record called OG_APP. This record is of type OG_APP_ATTR, which is fully defined in the section "Application Attribute Record" in this chapter.
You should use this global variable instead of an actual numeric value so that your application will maintain a consistent look on systems with different screen resolutions.
The window attibute record contains attributes that may be used only with windows.

```
TYPE og_window_attr is RECORD        Mask Constants:
(mask         NUMBER(2,0),
 position     og_point,              OG_POSITION_WINDOWA
 width        NUMBER(5,0),           OG_SIZE_WINDOWA
 height       NUMBER(5,0),           OG_SIZE_WINDOWA
 name         VARCHAR2(255),         OG_NAME_WINDOWA
 scrollbars   BOOLEAN,               OG_SCROLLBARS_WINDOWA
 helptarget   VARCHAR2(255)          OG_HELPTARGET_WINDOWA
);
                                     OG_ALL_WINDOWA
                                     OG_NONE_WINDOWA
```

	Attribute	*Description*
CSG	position	Is the x- and y-coordinates of the window's upper left corner (in screen resolution units).
CSG	width	Is the width of the window (in screen resolution units).
CSG	height	Is the height of the window (in screen resolution units)
CSG	name	Is the window's name. At runtime, the default name of the layout window is "Main Layout".
C	scrollbars	Specifies whether scroll bars appear in the window. The value of this attribute may be one of the following: **TRUE** Means the window has scroll bars. **FALSE** Means the window does not have scroll bars.
CSG	helptarget	Is the hypertext target in the runtime Help document that is displayed when the Help system is invoked while the window is active.

Global Variables

Built-in Global Variables

OG_App
OG_Inch
OG_Null_Axis
OG_Null_Buttonproc
OG_Null_Display
OG_Null_Ftemp
OG_Null_Layer
OG_Null_Object
OG_Null_Query
OG_Null_Refline
OG_Null_Sound
OG_Null_Template
OG_Null_Timer
OG_Null_Window

OG_App

Description Contains a snapshot of the application attribute values at the time the first Graphics built-in PL/SQL construct is executed.
Syntax

```
OG_App  OG_App_Attr;
```

Note: Since this global variable is a snapshot of values at one point in time, changes you make to the application's attributes will *not* be reflected in this variable. For example, the *username*, *password*, and *connection* attributes are *not* automatically updated when the database connection changes.

OG_Inch

Description Contains the number of layout units in one inch.
Syntax

```
OG_Inch  NUMBER;
```

OG_Null_Axis

Description Is a null handle to a chart axis.
Syntax

```
OG_Null_Axis  OG_Axis;
```

OG_Null_Buttonproc

Description Is a null handle to a button procedure.

Syntax

```
OG_Null_Buttonproc  OG_Buttonproc;
```

OG_Null_Display

Description Is a null handle to a display.

Syntax

```
OG_Null_Display  OG_Display;
```

OG_Null_Ftemp

Description Is a null handle to a field template.

Syntax

```
OG_Null_Ftemp  OG_Ftemp;
```

OG_Null_Layer

Description Is a null handle to a layer.

Syntax

```
OG_Null_Layer  OG_Layer;
```

OG_Null_Object

Description Is a null handle to a graphic object.

Syntax

```
OG_Null_Object  OG_Object;
```

OG_Null_Query

Description Is a null handle to a query.

Syntax

```
OG_Null_Query  OG_Query;
```

OG_Null_Refline

Description Is a null handle to a reference line.

Syntax

```
OG_Null_Refline  OG_Refline;
```

OG_Null_Sound

Description Is a null handle to a sound.

Syntax

```
OG_Null_Sound  OG_Sound;
```

OG_Null_Template

Description Is a null handle to a chart template.

Syntax

```
OG_Null_Template  OG_Template;
```

OG_Null_Timer

Description Is a null handle to a timer.

Syntax

```
OG_Null_Timer  OG_Timer;
```

OG_Null_Window

Description Is a null handle to a window.

Syntax

```
OG_Null_Window  OG_Window;
```

Index