The Editor Object

Programmer's Reference Manual for the Editor Object Source Code Internals

Software Farm, Inc.

Revision 2.0, October 10, 1993

The manual pages of this book were designed and automatically extracted from the source code by Michael L. Davis.

The information contained in this document is subject to change without notice.

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ABCDEFGHIJ-DO-93

Table Of Contents

Preface

This is the reference manual for the Editor Object source code. It contains a description of the major classes, methods and message handlers.

0.1 Audience

This document is a reference manual written for programmers who want to write applications using the Software Farm Editor Object, who have access to the source code, and who may have need to modify such source code.

0.2 Typographic Conventions

This document uses the following typographic conventions:

- ***Large*** boldface strings are used to represent class names.
- ***Small*** boldface strings are used to represent function/method names.
- ***Tiny*** boldface strings are used to represent methods, their arguments and return types.

0.3 Manual Page Format

The manual pages in this document use the following format:

- **Synopsis**

This section describes the syntax for using the interface.

- **Description**

This section gives a short description of the interface.

- **Parameters (required)**

This section lists and gives a short description of each of the arguments that the interface requires.

- **Parameters (optional)**

This section lists and gives a short description of each of the optional arguments to the interface.

- **Component Name**

This section lists the names of the components in which the interface may exist.

- **Messages Generated**

This section enumerates the possible messages that may be generated by using the interface.

- **Variables Set**

This section enumerates the message and component variables that may be modified by using the interface.

- **Exceptions Raised**

This section enumerates the possible error exceptions that may be caused by using the interface.

- **Caveats**

This section describes possible side-effects, unimplemented features, and other hazards of using the interface.

- **See Also**

This section lists the other, similar interfaces that may be of interest to the programmer.

- **Methods**

Lists and describes each method available to the programmer who uses the interface.

CHAPTER 1 Introduction

This document is a reference manual for C++ programmers who have access to Software Farm's Editor Object source code. It describes all of the functionality that comprises and supports the application programmer's interface (API).

1.1 The Organization of This Document

This document is split into chapters that each cover a major area of the Editor Object functionality. The following is a brief overview of each chapter.

- **Magnifiers, Views and other Kinds of Graphics Editors**

This chapter describes the large scale editing objects available to the programmer. Instantiating one of these editor objects in a window immediately provides an editing area that knows how to display graphics data structures (sometimes called display lists in the literature). Often, these editor objects work in tandem to provide unique and useful editing features for the user.

- **Message Handlers: Configuring Editor Functionality**

This chapter describes how to add and customize editor functionality by assigning message handlers to the editor. Message handlers take some specified action when some

event occurs (usually when some mouse or keyboard event is generated by the user in the editor's editing area).

- **Graphics Data Structures: Graphs, Trees and Lists**

This chapter describes the data structures provided in which to insert graphics display primitives (such as lines and circles). The graphics editor objects display these graphics data structures (often called display lists in the literature) and the message handlers operate on the graphics data structures.

- **Graphics Display Primitives: Lines, Circles and the Rest**

This chapter describes the numerous kinds of graphics display primitives available (such as lines, circles and text). These objects are inserted into a graphics data structure for display by an editor object.

- **Advanced Topics: Interface to the Low-Level Windowing/Graphics Standards**

This chapter describes the interface to and opaque wrappers around the whatever graphics and windowing system present on the machine. At this time only the X Window System and OSF/Motif are supported.

- **Advanced Topics: The Messaging System**

This chapter describes the messaging system which is the means by which low and high-level events are propagated throughout the system.

1.2 The Process of Using the Editor Object

This section provides a brief overview of how a programmer uses the Editor Object in order to provide the end users with a means to graphically view and manipulate application specific data.

1.2.1 Instantiating an editor object in a window.

1.2.2 Assigning message handlers to provide the desired editing features for the end-user.

1.2.3 Creating a graphics data structure in which to put graphics primitive display objects.

1.2.4 Creating the primitives (usually corresponding to elements of your

application's data) and adding them to the graphics data structure.

1.2.5 Assigning the graphics data structure to the editor.

1.2.6 <The end-user now may edit the data in the editor>.

1.2.7 Saving the changes the end-user made to the application data either by:

1.2.7.1 By examining the graphics data structure and converting the primitives back into application data.

1.2.7.2 By having taken care of updates to the application data each time the end-user made a change (this is often the only alternative if application design-rule checking is required in order to force constraints on the end-user in what the end-user may or may not do to the application data).

CHAPTER 2

Magnifiers, Views and other Kinds of Graphics Editors

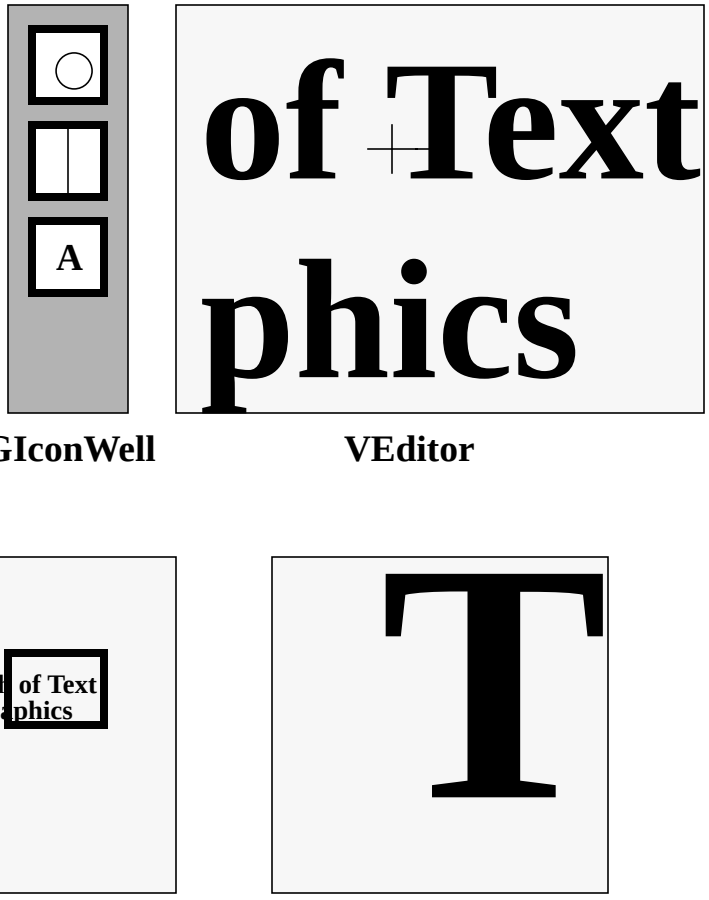

FIGURE 1 Various Editor Objects and the Iconwell

GAnotherEditorView

SYNOPSIS:

Simple constructor, call the setup method later to assign values to the editor.

GAnotherEditorView ();

Most often used constructor.

***GAnotherEditorView** (Display *display, char *container, VEditor *editor,*
G_DCOORD dxmin = 0, G_DCOORD dymin = 0,
G_DCOORD dxmax = 200, G_DCOORD dymax = 200,
G_WCOORD wxmin = 0, G_WCOORD wymin = 0,
G_WCOORD wxmax = 10000, G_WCOORD wymax = 10000);

DESCRIPTION

Creates a fully functional editor that permits simultaneous, continuously updated display and editing of another editor's graphics data.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

display	The display object for this window/ graphics system.
container	The container object specific to the underlying window system.
editor	The editor whose data this will display.
dxmin, dymin, dxmax, dymax	The size of the editor in device coordinates with respect to the parent's coordinate system.
wxmin, wymin, wxmax, wymax	The size of the editor in world coordinates.

COMPONENT NAME:

Editor

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VEditor, GMagnifier, GLocator, GLocatorTool

METHODS:

get_locator_tool

Return the address of the locator tool this editor is using. Only valid after the source editor has been assigned.

GLocatorTool ***get_locator_tool ()***

set_source_editor

Assign the editor whose graphics data this editor will also display and manipulate.

virtual void ***set_source_editor (VEditor *seditor);***

GIconWell

SYNOPSIS:

GIconWell ***(***

Display *display,
VEditor *editor,
APPLICATION_OBJECT_CREATOR applObjConstructor,
void *applObjConstructorData);

DESCRIPTION:

Manages an area that contains icons (pictorial labels) that can be dragged into the given editor. This explicitly creates a graphics object like the original icon. It also creates an application object (by calling the given applObjConstructor function, with the given applObjConstructorData information. A NEW_NODE_ACTION is then sent to the new application object. The user then is free to continue moving the new object/icon around the editing area and drop it where desired.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

display The display that will be used.

editor The editor that this iconwell supplies with new icons. May be NULL until later if the editor is unknown at the time of creation of this icon well.

applObjConstructor The function to call to create an application specific object that the new icons/objects represent.

applObjConstructorData Data that the applObjConstructor may need to do its job.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

There is currently no support at this time for other events to drag an icon into the editor other than moving the mouse with the left mouse button pressed down.

SEE ALSO:

VEditor

METHODS:

addIconToWell

Add an icon to the icon well as specified by the pixmap in the given filename and the specified iconwell container. The userdata parameter supplies data to be sent to the actions () method when it is called with the NEW_NODE_ACTION message.

void addIconToWell (char *iconwellContainer, char *iconFilename, char *userdata);

set_editor

Specify the editor into which icons will be dragged.

void set_editor (VEditor *e)

GLocator

SYNOPSIS:

Simple constructor, call the setup method later to assign values to the editor.

GLocator ();

Most often used constructor.

***GLocator** (Display *display, char *container, VEditor *editor, G_DCOORD dxmin = 0, G_DCOORD dymin = 0, G_DCOORD dxmax = 200, G_DCOORD dymax = 200);*

DESCRIPTION

Creates an type of editor that displays another editor's entire amount of graphics data at a reduced scale. In conjunction with the GLocatorTool, a interactive rectangle is displayed in the locator window which indicates the current location (continuously updated) of the field of view of the main (source) editor.
The GLocatorTool, which is automatically created in the GMagnifier constructor, can be configured in many ways.

This is often referred to as a birds-eye-view as well as a locator window in the literature.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

display	The display object for this window/ graphics system.
container	The container object specific to the underlying window system.
editor	The editor whose data this will display.
dxmin, dymin, dxmax, dymax	The size of the editor in device coordinates with respect to the parent's coordinate system.

COMPONENT NAME:

Editor

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VEditor, GMagnifier, GAnotherEditorView, GLocatorTool

METHODS:

get_locatorTool

Return the address of the locator tool this editor is using. Only valid after the source editor has been assigned.

GLocatorTool* get_locatorTool ()

set_source_editor

Assign the editor whose graphics data this editor will also display.

virtual void set_source_editor (VEditor *seditor);

GLocatorTool

SYNOPSIS:

GLocatorTool ***(***
VEditor *srceditor,
VEditor *locwin,
int options,
Boolean filled = True,
char *colorname = “red”);

DESCRIPTION:

This object links two editors together in a way that provides the functionality necessary to make the locator window and magnifier work. This tool can be configured in many ways.

PARAMETERS (Required):

srceditor The editor to map, whose data we are to display in various ways.

locwin The locator window, magnifier or some kind of editor which this tool is going to overlay to add additional interactive and visual functionality to.

PARAMETERS (Optional):

options Specifies what functionality is desired from this tool. The possibilities are:

GraphicsBackground

Specifies whether or not the source editor’s graphics will be drawn in the background of the ‘locwin’ editor. This option is visually attractive but may be quite slow on some hardware because the graphics background is continuously updated, even during interactive editing in the source editor.

DisplayAllOfSourceUniverse

Specifies that the locwin is

to be a locator window that displays all of the graphics data that is contained in the source editor.

LocatorHasBox

Specifies that the current size of the view seen in the source editor is to be drawn as a rectangle in the locwin editor.

ReadOnly

Not used at this time.

NoGraphicsInBox

Same viewport as source.

AttachBoxToMouse

An option that attaches a rectangle the size equal to the relative size of the magnifier's viewport. In a sense, this creates a dynamic magnifying glass.

AttachBoxToUnderlyingGraphics64

This option specifies that the magnifying glass not be attached to the mouse cursor but laid down on the graphics data to provide a more permanent, magnified view of an area of graphics data of interest.

SourceHasOverlay

True for magnifiers, false for locator windows. Specifies which window will be overlaid with the interactive functionality of this tool.

MaintainMagnification

Specifies whether or not the magnifier is either always magnified by a fixed amount or by a fixed amount on top of what ever the current magnification

(zoom level) is in the source editor.

MaintainSameBackground

Specifies whether or not the locwin editor should maintain the same background color as the source editor.

(The default options are for a locator window (GLocator) and are:

GraphicsBackground
DisplayAllOfSourceUniverse
LocatorHasBox
ReadOnly
NoGraphicsInBox
MaintainSameBackground

container The container object specific to the underlying window system.

filled Wether or not any box/rectangle this tool generates will be filled (i.e. a solid color).

colorname The color of any box/rectangle this tool generates.

COMPONENT NAME:

Editor

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VEditor, GMagnifier, GAnotherEditorView, GLocator

GLocatorTool

GMagnifier

SYNOPSIS:

Simple constructor, call the setup method later to assign values to the editor.

GMagnifier ();

Most often used constructor.

***GMagnifier** (Display *display, char *container, VEditor *editor, G_DCOORD dxmin = 0, G_DCOORD dymin = 0, G_DCOORD dxmax = 200, G_DCOORD dymax = 200);*

DESCRIPTION:

Creates an type of display that displays another editor's graphics, continuously updated and highly magnified, in the neighborhood of the mouse cursor. This occurs in conjunction with the GLocatorTool, which is automatically created in the GMagnifier constructor, and can be configured in many ways.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

display	The display object for this window/ graphics system.
container	The container object specific to the underlying window system.
editor	The editor whose data this will display.
dxmin, dymin, dxmax, dymax	The size of the editor in device coordinates with respect to the parent's coordinate system.

COMPONENT NAME:

Editor

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VEditor, GLocator, GAnotherEditorView, GLocatorTool

METHODS:

get_locator_tool

Return the address of the locator tool this editor is using. Only valid after the source editor has been assigned.

GLocatorTool ***get_locator_tool ()***

set_source_editor

Assign the editor whose graphics data this editor will also display.

virtual void ***set_source_editor (VEditor *seditor);***

GPrinter

SYNOPSIS:

GPrinter ***(Display *display, char *printfilename,***
GExtrema *dextrema, GExtrema *wextrema)

DESCRIPTION:

Constructs a virtual editor representing a printed page that can be drawn to same as a ordinary VEditor.

Message handlers have no effect with this write-only editor.

PARAMETERS (Required):

display	A display that can be used to determine the rgb colors assigned to color values and font characteristics assigned to text.
printfilename	The file to send the printing commands (postScript) to.
dextrema	The size of the page in pixels coordinates after any rotation.
wextrema	The size of the printer page (s) in world coordinates.

PARAMETERS (Optional):

None.

COMPONENT NAME:

Editor

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VComposite, GViewport, VEditor, GPrintPS

METHODS:

get_margins

Return page edge margins required to be left blank for the particular printer device.

void ***get_margins (float *left, float *right, float *bottom, float *top);***

initialize

Initialize printer and printer object.

void ***initialize ();***

set_bounds

Sets the bounds of the page after all calculations have been made.

void ***set_bounds (G_DCOORD xmin, G_DCOORD ymin, G_DCOORD xmax, G_DCOORD ymax);***

set_bwThreshold

Set value (in the range 0.0 to 1.0) to be the line between black and white after converting colors to a standard gray scale.

Boolean ***set_bwThreshold (float t);***

set_color_output_type

Set color output to one of: “Grey”, “Color” or “BW”.

Boolean ***set_color_output_type (char *type);***

set_font_dimensions

Set size of font in specified resolution.

Boolean ***set_font_dimensions (G_DWIDTH width, G_DWIDTH height);***

set_output_resolution

Set resolution to # dots per inch desired.

Boolean set_output_resolution (int resolution);

set_page_orientation

Set orientation of page to “LANDSCAPE” or “PORTRAIT”.

Boolean set_page_orientation (char * orientation);

set_page_size

Set page size to one of: “LEGAL” or “LETTER”.

Boolean set_page_size (char *std);

GUnClippedViewport

SYNOPSIS:

This is usually instantiated by the derived class: GViewport.

GUnClippedViewport ***(***
G_WCOORD wxmin, G_WCOORD wymin,
G_WCOORD wxmax, G_WCOORD wymax,
G_DCOORD dxmin, G_DCOORD dymin,
G_DCOORD dxmax, G_DCOORD dymax);

DESCRIPTION:

Creates a object that manages the world to/from device coordinate transformations and operations that are commonly required by the VEditor object and it's functions and GMsgHandlers.

PARAMETERS (Required):

wxmin, wymin, ***wxmax, wymax***	The size of the viewport in world coordinates.
dxmin, dymin, ***dxmax, dymax***	The size of the viewport in device coordinates with respect to the parent's coordinate system.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GUnClippedViewport

VEditor, GViewport

METHODS:

confine_proposed_world_to_constraints

Shrinks, if necessary, and translates this viewport if necessary to keep the world coordinates within the specified (if any) universe (maximum world space) and miniverse (minimum world space).

void confine_proposed_world_to_constraints ();

confine_translated_extrema_to_universe

Given the extrema (wxmin, wymin, wxmax, wymax) of an object which is being translated by the given amount (wdx, wdy), this modifies the translation, in wdx and wdy, in order to keep the object within the maximum allowed bounds of world space.

void confine_translated_extrema_to_universe (G_WWIDTH *wdx,
G_WWIDTH *wdy, G_WCOORD wxmin, G_WCOORD wymin,
G_WCOORD wxmax, G_WCOORD wymax);

copy

Make a copy of this viewport and return the address of the copy.

GUnClippedViewport *copy ();

dclip_reject

Return ‘True’ if rejected, i.e. the extrema, in the device coordinates specified, is entirely outside the current view.

virtual Boolean dclip_reject (G_DCOORD xmin, G_DCOORD ymin,
G_DCOORD xmax, G_DCOORD ymax)

dctowc

Convert the coordinates in device space to their corresponding coordinates in world space.

void dctowc (G_DCOORD x, G_DCOORD y, G_WCOORD *wx,
G_WCOORD *wy);

dctowc

Convert the coordinates in device space to their corresponding coordinates in world space, using the given scale factors.

void dctowc (double wxscale, double wyscale, G_DCOORD x, G_DCOORD y, G_WCOORD *wx, G_WCOORD *wy);

device_pan

Translate the world space by the given amounts (specified in device coordinates). Updates the scrollbars as well.

Boolean device_pan (G_DCOORD *dx, G_DCOORD *dy);

device_zoomin_around_cursor

Does a simple zoom in (magnification) around the given coordinate.

void device_zoomin_around_cursor (G_DCOORD x, G_DCOORD y);

device_zoomout_around_cursor

Does a simple zoom out (de-magnification) around the given coordinate.

void device_zoomout_around_cursor (G_DCOORD x, G_DCOORD y);

dtow

Convert the distances in device space to their corresponding distances in world space.

void dtow (G_DWIDTH dx, G_DWIDTH dy, G_WWIDTH *wdx, G_WWIDTH *wdy);

dtow

Convert the distances in device space to their corresponding distances in world space, using the given scale factors.

void dtow (double wxscale, double wyscale, G_DCOORD x, G_DCOORD y, G_WCOORD *wx, G_WCOORD *wy);

get_amount_extrema_translated_outside_world

Given the extrema (wxmin, wymin, wxmax, wymax) of an object which is being translated by the given amount (wdx, wdy), this returns, in pandx and pandy, how much the world space would also have to be translated in order for the object to remain visible. This is used in order to support autopan functionality.

void get_amount_extrema_translated_outside_world (G_WWIDTH wdx, G_WWIDTH wdy, G_WCOORD wxmin, G_WCOORD wymin,

G_WCOORD wxmax, G_WCOORD wymax, G_WWIDTH *pandx, G_WWIDTH *pandy);

get_device

Return the lower-left and upper-right coordinates of the device space.

void get_device (G_DCOORD *dxmin, G_DCOORD *dymin, G_DCOORD *dxmax, G_DCOORD *dymax);

get_device

Return the lower-left and upper-right coordinates of the device space in the given extrema object.

void get_device (GExtrema *extrema);

get_drawarea

Return the drawarea that this viewport is using.

GDrawarea *get_drawarea ()

get_dxscale

Return the scale factor which will convert horizontal world distances to horizontal device distances.

double get_dxscale ()

get_dyscale

Return the scale factor which will convert vertical world distances to vertical device distances.

double get_dyscale ()

get_relative_position_of_viewport_in_universe

Returns the location of the center of the world space within the maximum allowed size of the world space, on a scale from 0.0 to 1.0. This is usually used to set the location of the scrollbars.

void get_relative_position_of_viewport_in_universe (double *horizontal, double *vertical);

get_relative_size_of_viewport_in_universe

Returns the relative size, on a scale from 0.0 to 1.0, of the current world space relative to the maximum allowed world space. This is usually used to determine what the size of a scrollbar should be.

void ***get_relative_size_of_viewport_in_universe (double *xsize, double *ysize);***

get_universe

Return the maximum allowed size for the world space.

Boolean ***get_universe (G_WCOORD *uwxmin, G_WCOORD *uwymin, G_WCOORD *uwxmax, G_WCOORD *uwymax);***

get_universe

Return the maximum allowed size for the world space.

Boolean ***get_universe (GExtrema *extrema);***

get_world

Return the lower-left and upper-right coordinates of the world space.

void ***get_world (G_WCOORD *wxmin, G_WCOORD *wymin, G_WCOORD *wxmax, G_WCOORD *wymax);***

get_world

Return the lower-left and upper-right coordinates of the world space in the given extrema object.

void ***get_world (GExtrema *extrema);***

get_wxscale

Return the scale factor which will convert horizontal device distances to horizontal world distances.

double ***get_wxscale ()***

get_wyscale

Return the scale factor which will convert vertical device distances to vertical world distances.

double ***get_wyscale ()***

pan_to

Pans to the specified location. Supported locations are:
TOP_OF_UNIVERSE

void pan_to (int direction);

set_device

Specify the lower-left and upper-right coordinates of the device space. Note that this does not alter the size of any windows, this method merely informs the viewport what the size is.

void set_device (G_DCOORD dxmin, G_DCOORD dymin, G_DCOORD dxmax, G_DCOORD dymax);

set_device

Specify the lower-left and upper-right coordinates of the device space using the given extrema object. Note that this does not alter the size of any windows, this method merely informs the viewport what the size is.

void set_device (GExtrema *extrema);

set_drawarea

Specify the low-level drawarea object this viewport will use.

void set_drawarea (GDrawarea *drawarea)

set_editor

Specify the editor this viewport will be used by.

void set_editor (VEditor *editor)

set_horizontal_position_of_world_in_universe

Specifies the horizontal location of the center of the world space within the maximum allowed size of the world space, on a scale from 0.0 to 1.0. This is usually used to reposition the world space in response to a user moving a scrollbar.

void set_horizontal_position_of_world_in_universe (double horizontal);

set_miniverse

Specify the minimum allowed size for the world space.

void set_miniverse (GExtrema *extrema);

set_miniverse

Specify the minimum allowed size for the world space.

void set_miniverse (G_WWIDTH width, G_WWIDTH height);

set_universe

Specify the maximum allowed size for the world space.

void set_universe (G_WCOORD uwxmin, G_WCOORD uwymin, G_WCOORD uwxmax, G_WCOORD uwymax);

set_universe

Specify the maximum allowed size for the world space.

void set_universe (GExtrema *extrema);

set_vertical_position_of_world_in_universe

Specifies the vertical location of the center of the world space within the maximum allowed size of the world space, on a scale from 0.0 to 1.0. This is usually used to reposition the world space in response to a user moving a scrollbar.

void set_vertical_position_of_world_in_universe (double vertical);

set_world

Specify the lower-left and upper-right coordinates of the world space.

void set_world (G_WCOORD wxmin, G_WCOORD wymin, G_WCOORD wxmax, G_WCOORD wymax);

set_world

Specify the lower-left and upper-right coordinates of the world space using the given extrema object.

void set_world (GExtrema *extrema);

wclip_reject

Return ‘True’ if rejected, i.e. the extrema, in the world coordinates specified, is entirely outside the current view.

virtual Boolean wclip_reject (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax);

wctodc

Convert the coordinates in world space to their corresponding coordinates in device space.

void wctodc (G_WCOORD wx, G_WCOORD wy, G_DCOORD *x, G_DCOORD *y);

wctodc32

Returns the coordinates, in device space, which corresponds to the given world coordinates. The device coordinates are returned with the integer value in the top 16 bits and the fractional value in the bottom 16 bits.

void wctodc32 (G_WCOORD wx, G_WCOORD wy, long *x, long *y);

wfastclip_accept

Return ‘True’ if accepted, i.e. the extrema, in the world coordinates specified, is entirely inside the current view.

Boolean wfastclip_accept (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)

wfastclip_reject

Return ‘True’ if rejected, i.e. the extrema, in the world coordinates specified, is entirely outside the current view.

Boolean wfastclip_reject (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)

world_pan

Translate the world space by the given amounts. Updates the scrollbars as well.

Boolean world_pan (G_WWIDTH *dx, G_WWIDTH *dy);

wtod

Convert the distances in world space to their corresponding distances in device space.

void wtod (G_WWIDTH wdx, G_WWIDTH wdy, G_DWIDTH *dx, G_DWIDTH *dy);

wtod32

Returns the distance, in device space, which corresponds to the given world distance. The device coordinates are returned with the integer value in the top 16 bits and the fractional value in the bottom 16 bits.

void wtod32 (G_WWIDTH wdx, G_WWIDTH wdy, long *dx, long *dy);

zoomed_or_panned

This is called when it is desired that the window scrollbars be updated and that a message indicating that this viewport's world space has changed location and/or size be broadcast.

void zoomed_or_panned ();

GViewport

SYNOPSIS:

GViewport ***(***
G_WCOORD wxmin, G_WCOORD wymin,
G_WCOORD wxmax, G_WCOORD wymax,
G_DCOORD dxmin, G_DCOORD dymin,
G_DCOORD dxmax, G_DCOORD dymax);

GViewport ***(GExtrema *world, GExtrema *device);***

DESCRIPTION:

Creates a object that manages the world to/from device coordinate transformations and operations that are commonly required by the VEditor object and it's functions and GMsgHandlers.

This inherits all the functionality offered by the class: GUnClippedViewport.

In addition this viewport supports a clipping rectangle, which, when specified, is the only area in which drawing is allowed in this particular viewport.

PARAMETERS (Required):

wxmin, wymin, wxmax, wymax	The size of the viewport in world coordinates.
dxmin, dymin, dxmax, dymax	The size of the viewport in device coordinates with respect to the parent's coordinate system.
world	The size of the viewport in world coordinates.
device	The size of the viewport in device coordinates with respect to the parent's coordinate system.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VEditor, GUnClippedViewport

METHODS:

dclip_reject

Return 'True' if rejected, i.e. the extrema, in the device coordinates specified, is entirely outside the current view.

virtual Boolean dclip_reject (G_DCOORD xmin, G_DCOORD ymin, G_DCOORD xmax, G_DCOORD ymax)

set_clipbounds

Specifies the bounds of the only area where drawing is now allowed.

void set_clipbounds (G_WCOORD wxmin, G_WCOORD wymin, G_WCOORD wxmax, G_WCOORD wymax);

set_clipbounds

Specifies the bounds of the only area where drawing is now allowed.

void set_clipbounds (GExtrema *extrema);

unset_clipbounds

Specifies that the entire viewport is once again allowing drawing.

void unset_clipbounds ();

wclip_reject

Return ‘True’ if rejected, i.e. the extrema, in the world coordinates specified, is entirely outside the current view.

virtual Boolean wclip_reject (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax);

VEditor

SYNOPSIS:

Simple constructor, call the setup method later to assign values to the editor.

VEditor ();

Most often used constructor.

***VEditor** (Display *display,*
*char *parent_window,*
G_DCOORD dxmin, G_DCOORD dymin,
G_DCOORD dxmax, G_DCOORD dymax,
G_WCOORD wxmin, G_WCOORD wymin,
G_WCOORD wxmax, G_WCOORD wymax,
Boolean default_event_handlers = True);

Constructs an editor that overlays another editor.

***VEditor** (VEditor *overlaid, Boolean sameViewport = True);*

DESCRIPTION:

Creates an editor in which lists of objects (class VComposite) are drawn and manipulated by the user.

Any message handler with a ‘Container Component’ of ‘Editor’ can be assigned to any VEditor class. Message handlers add manipulation and display capabilities to the basic editor object.
The following message handlers are assigned to the editor: GResize and GRepaint.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

display	The display object for this window/ graphics system.
parent_window	The container object specific to the underlying window system.
dxmin, dymin, dxmax, dymax	The size of the editor in device coordinates with respect to the parent’s coordinate system.

wxmin, wymin, wxmax, wymax The size of the editor in world coordinates.

default_event_handlers If True the following message handlers are instantiated and assigned to the editor using their default translations.

GExpandCursorFootprint
GCumulativeSelect
GSmoothPan
GZoom
GSimpleDrag

COMPONENT NAME:

Editor

MESSAGES GENERATED:

CompositeChanged_name	When a new VComposite list is assigned to the editor.
BackgroundChanged_name	When the background color of the editor changes.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VComposite, GViewport

METHODS:

add_overlay

Add an editor as an overlay to this one.

void add_overlay (VEditor *overlay);

append_damaged_objarea

Append the area of the given objects extent to the list of damaged areas managed and maintained by the editor.

void append_damaged_objarea (GObject *obj);

autopan_for_moving_obj

Adjust the viewport world view (pan) so that if the given extrema (presumably of some object) moves the given amounts, it will still remain visible. Returns True only if the viewport was actually adjusted.

Boolean autopan_for_moving_obj (GExtrema *objext, G_WWIDTH *wdx, G_WWIDTH *wdy);

clear

Clear the editor to the background color.

void clear ();

deselect_all

Deselect all objects that are currently selected.

void deselect_all ();

draw

Draw the given object.

void draw (GObject *obj);

draw

Draw everything in the editor.

void draw ();

draw_damaged_areas

Draw and mark as undamaged the list of damaged areas managed and maintained by the editor.

void draw_damaged_areas ();

draw_no_clear

Draw everything in the editor but the editor background color.

void draw_no_clear ();

draw_objarea

Redraw the editor background and graphics the lie within the area of the given objects extent.

void draw_objarea (GObject *obj, Boolean flag);

drawarea

Redraw the editor background and graphics the lie within the given area.

void drawarea (GExtrema *area);

get_background_color

Return the value of the background color of the editor.

int get_background_color ();

get_composite

Return the graph assigned to this editor.

VComposite*get_composite ()

get_display

Return the display that this editor is using.

Display *get_display ()

get_drawarea

Return the drawarea which has been assigned to this editor.

GDrawareaInterface *get_drawarea ();

get_home

Return the reference (i.e. home) viewing area.

void get_home (GExtrema *h);

get_homeZoom

Return the reference (i.e. home) zoom level.

void get_homeZoom (G_WWIDTH *width, G_WWIDTH *height);

get_id

Return unique ID that identifies this editor from all other editors.

int get_id ()

get_named_color

Return the color value of the given color name.

int get_named_color (char *name);

get_selectedObjectList

Return the list of objects which are currently selected (probably by the user) within the editor.

VIndirectList*get_selectedObjectList ()

get_viewport

Return the viewport which has been assigned to this editor.

GViewport *get_viewport ()

panTo_object

Adjust (pan) the viewport so that the given object is visible.

void panTo_object (GObject *obj);

pick_node

Return the node in the assigned graph (e.g. VComposite) that lies within the area of the given extrema.

VNode *pick_node (GExtrema *extrema);

repaint

Draw everything in the editor.

void repaint ();

repeatable_pick

Return the node (s) in the assigned graph (e.g. VComposite) that lies within the area of the given extrema. Updates the iterator so that previous objects in the graph that lie under the first nodes returned can be later returned by repeatedly calling this routine.

GObject *repeatable_pick (VCompositeNode **iterator, GExtrema *extrema);

resize

Inform the editor that it must readjust any internal values that reflect the new size of the editor.

void resize ();

select_object

Make the given object one of the currently selected.

Boolean select_object (VCompositeNode *node);

set_background_color

Set the background color of the editor to the given color name.

void set_background_color (char *colorname);

set_background_color

Set the background color of the editor to the given color value.

void set_background_color (int color);

set_composite

Set the graph to be displayed in the editor.

void set_composite (VComposite *list);

set_cursorPosition

Set the mouse cursor position. The message object is only used to update the message class's static mouse object.

void set_cursorPosition (GMessage *msg, G_WCOORD wx, G_WCOORD wy);

set_display

Set the display for this editor to use.

void set_display (Display *disp)

set_home

Set home (i.e. reference) viewing area. This is a convenience so that the user may easily return to a specific view of the graphic data.

void set_home (GExtrema *h);

set_homeZoom

Set home (i.e. reference) zoom level. This is a convenience so that the user may easily return to a specific magnified view of the graphic data.

void set_homeZoom (G_WWIDTH width, G_WWIDTH height);

set_selectedObjectList

Assign to the editor then list which is to contain objects selected (probably by the user).

void set_selectedObjectList (VIndirectList *list)

set_size

Set the size of the editor in device coordinates.

void set_size (G_DCOORD width, G_DCOORD height);

update_universe_to_include_all_graphics

Adjust the viewport (pan and zoom) so that all graphics within the assigned graph (VComposite) are made visible with room to spare. If timesLargerThanAllGraphics is equal to 2.0 the 1/2 of width and height of the entire set of graphics data is added for the margin (bordering empty space.

void update_universe_to_include_all_graphics (float timesLargerThanAllGraphics);

CHAPTER 3

Message Handlers: Configuring Editor Functionality

Message handlers are objects that inherit functionality from the GMsgHandler base class. The most common way to invoke a message handler is by sending it a message object (of class GMessage). This message object has, as it's name/verb, the name of the message handler. Because of the importance of this name, this chapter lists message handlers by name, not class name. However the class name can be generated from the name by converting the first letter of the name to uppercase and prepending a 'G'. Conversely, there is a '#define' which represents each message handler name which is found by appending a '_name' to the message handler's class name.

When a message handler is creating, an editor is specified which causes the message handler to automatically register itself with the specified editor. It then, unless specified otherwise, adds translations to the editor so that the events specified in it's default translations will, when received from the end-user, trigger a message to be sent to the message handler.

For example, the following code fragment adds the GZoomAroundCursor message handler to the given editor. Now, whenever the end-user clicks the middle button of the mouse, the editor 'zooms in' on the displayed graphics and whenever the end-user clicks the right mouse button, the editor 'zooms out'.

```
Veditor *editor;
```

```
new GZoomAroundCursor(editor);
```

Now if, for example, one wanted to change the event that caused the displayed graphics to zoom in and out to, say, require the ‘shift’ key to be held down, then the following could be done:

```
// Create handler and make it without default translations.
new GNotifyNodeOfEvent(editor, False);

editor->get_translator()->add_translation(
        MIDDLE_MOUSE_CLICK_EVENT,       // type
        0,                              // key
        SHIFT_KEY_HELD,                 // shift status
        new GMessage (GZoomAroundCursor_name, ZoomIn_name));
editor->get_ translator()->add_translation(
        RIGHT_MOUSE_CLICK_EVENT,        // type
        0,                              // key
        SHIFT_KEY_HELD,                 // shift status
        new GMessage (GZoomAroundCursor_name, ZoomOut_name))
```

Similarly, if one wanted to be informed about each time the end-user, say, pressed the letter ‘a’ while the mouse cursor was over a graphics display primitive then the following could be done:

```
#define USER_PRESSED_A “pressed an a”
Veditor *editor;

// Create handler and make it without default translations.
GNotifyNodeOfEvent *handler = new GNotifyNodeOfEvent(editor, False);
editor->get_translator()->add_translation(
        KEY_EVENT,                      /* type */
        ‘a’,                            /* key */
        0,                              /* shift status */
        new GMessage (GNotifyNodeOfEvent_name));
handler->set_actionToSendToNode (USER_PRESSED_A);
```

Then, whenever the end-user pressed the letter ‘a’ over an object, the GNotifyNodeOfEvent handler would send a USER_PRESSED_A action to the object’s actions () method (within which the programmer would do a strcmp to determine it is indeed the USER_PRESSED_A action that occurred).

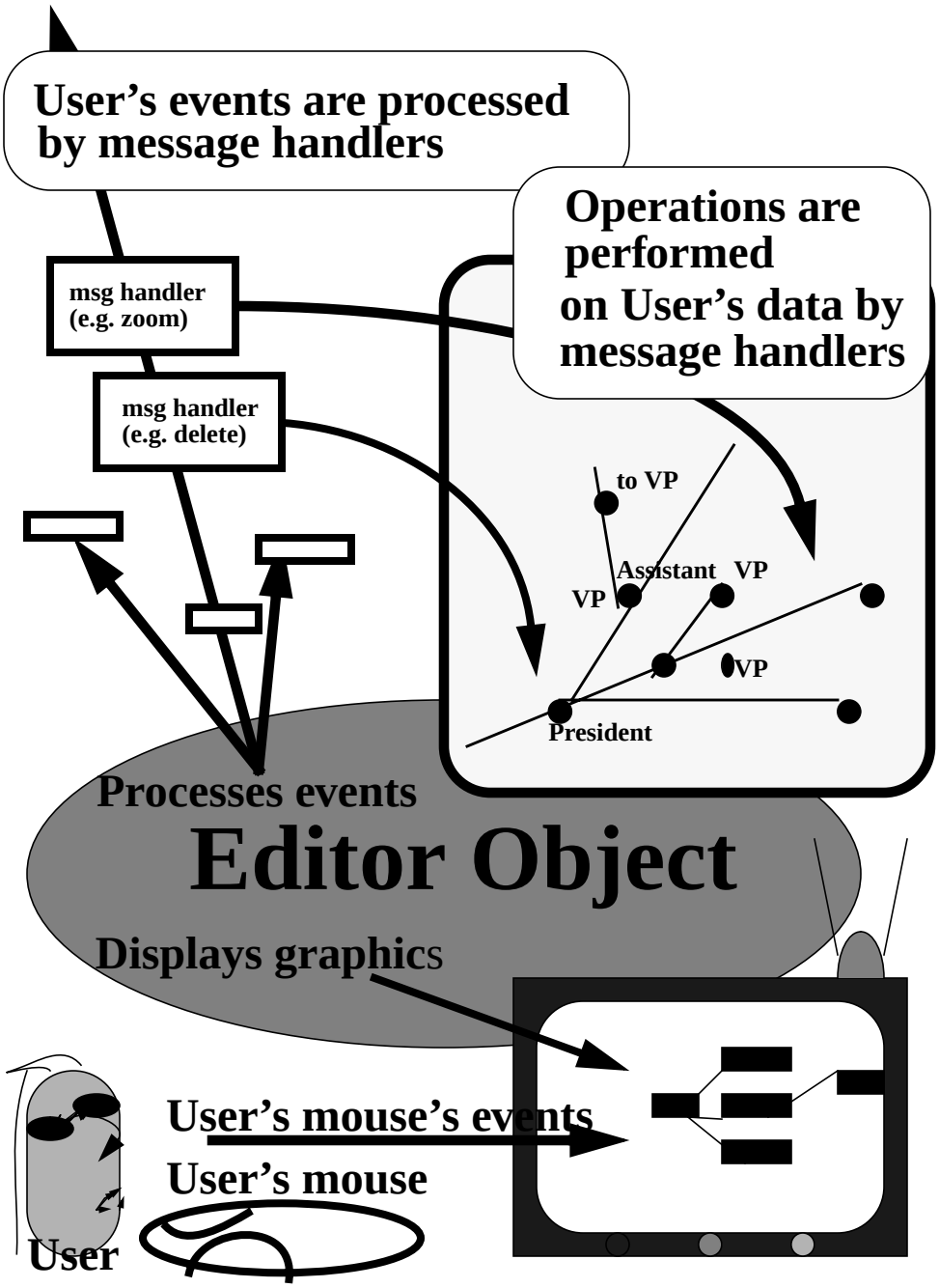

FIGURE 2 Overview of GMsgHandlers and their Behavior

alternatingSelect

SYNOPSIS:

alternatingSelect ***([keepOneSelectedAtAllTimes)***

DESCRIPTION:

The user selects a group of objects by repeatedly selecting (by generating the event as specified in the translation) each object in turn. If a selected object is selected again, it is deselected.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

keepOneSelectedAtAllTimes If true, then this handlers selection method assures that one-and-only-one item is selected. If false then this handler assures that at most one item is selected. Note that other handlers may select or deselect items (e.g. by deletion) and break the rules this handler tries to enforce. (Default is equal to false).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn1Click> alternatingSelect()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

alternatingSelect

SEE ALSO:

deselectAll, selectArea, cumulativeSelect, doubleSelect

METHODS:

set_keepOneSelectedAtAllTimes

Specify whether or not to prohibit this handlers de-selection of all items.

void set_keepOneSelectedAtAllTimes (Boolean flag)

callFunctionWhenEventOccurs

SYNOPSIS:

***callFunctionWhenEventOccurs** ([userData])*

DESCRIPTION:

Notifies a previously registered function whenever the event to which this Message Handler has been assigned has occurred.

If False is returned from the function, no other message handlers will see this event. If True is returned then The event will be passed on to any other handlers looking for it.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

userData	The 2nd parameter of the function to call (the first is specified when the function is registered). (Default userData is NULL).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

notifyNodeOfEvent, sendMsgToObjectUnderCursor

METHODS:

callFunctionWhenEventOccurs

set_functionToCall

Specifies what the function to call is as well as the first parameter.

void set_functionToCall
(CALL_FUNCTION_WHEN_EVENT_OCCURS_FNPTR fn,
void *objData);

centerCursor

SYNOPSIS:

centerCursor ()

DESCRIPTION:

Sets the mouse cursor position to the center of the editor.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>a centerCursor()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

setCursor, setCursorAppearance, cursorArm, centerLast, centerLastUnderCursor

centerLast

SYNOPSIS:

centerLast ()

DESCRIPTION:

Sets the position of the last graphics primitive created to the center of the editor.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

centerLastUnderCursor, centerCursor, setCursor, setCursorAppearance, cursorArm

centerLastUnderCursor

SYNOPSIS:

centerLastUnderCursor ()

DESCRIPTION:

Sets the position of the last graphics primitive created to the position of the mouse cursor.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

centerLast, centerCursor, setCursor, setCursorAppearance, cursorArm

cumulativeSelect

SYNOPSIS:

cumulativeSelect ()

DESCRIPTION:

The user selects a group of objects by repeatedly selecting (by generating the event as specified in the translation) each object in turn. If a selected object is selected again, it is deselected.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn1Click> cumulativeSelect()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

deselectAll, selectArea, alternatingSelect, doubleSelect

cursorArm

SYNOPSIS:

***cursorArm** (Motion | WinExit)*

DESCRIPTION:

'Arms' the graphics object underneath of the mouse cursor (if the graphics object is armable). This consists of a visual highlighting of the graphics object, indicating the fact that the mouse cursor is within the graphics object's selectable area.

PARAMETERS (Required):

Motion	The event that updates the currently armed object, if it has changed.
WinExit	The event that assures there is no currently armed object.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Motion> cursorArm(Motion)
<WinExit> cursorArm(WinExit)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

setCursor, setCursorAppearance, centerCursor

deSelectAll

SYNOPSIS:

deSelectAll ()

DESCRIPTION:

Deselect (Un select) all objects currently selected.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Esc> deselectAll()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

cumulativeSelect, selectArea, alternatingSelect, doubleSelect

delete

SYNOPSIS:

delete ()

DESCRIPTION:

Deletes all selected objects.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>Delete delete ()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

deleteObjUnderCursor

deleteObjUnderCursor

SYNOPSIS:

deleteObjUnderCursor ()

DESCRIPTION:

Deletes object underneath cursor if selectable.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>Delete delete ()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

delete

doubleSelect

SYNOPSIS:

doubleSelect ()

DESCRIPTION:

The user selects an object by double selecting it (by generating the event as specified in the translation, which is expected to be two select events). The object is sent a select event do restore it to the state that the first half of this double select changed.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn1DblClick> doubleSelect()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

deselectAll, selectArea, alternatingSelect, cumulativeSelect

draw

SYNOPSIS:

draw ()

DESCRIPTION:

Draw the graphics in the editing area, and all views, locator, scopes and magnifiers of the graphics.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>d draw()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

drawLast, showNodes, showOnlyRelatives

drawLast

SYNOPSIS:

drawLast ()

DESCRIPTION:

Draw the last graphics primitive created.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

draw, showNodes, showOnlyRelatives

expandCursorFootprint

SYNOPSIS:

expandCursorFootprint ()

DESCRIPTION:

Expands the apparent size of the cursor from a single pixel into a 2 by 2 pixel square. This allows the user's cursor work (picks) that are close to work as well as those that are exact.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<All> expandCursorFootprint().

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

hide

SYNOPSIS:

hide ***(True | False)***

DESCRIPTION:

Hides or displays the entire editor.

PARAMETERS (Required):

True	The editor is hidden, the containing window is resized to compensate.
False	The editor is displayed, the containing window is resized to compensate.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

setSize, setBackgroundColor

hideBitmapsIfSmaller

SYNOPSIS:

hideBitmapsIfSmaller ()

DESCRIPTION:

Tells all bitmaps graphics to draw only if bigger than or the same size as the bitmaps at the current zoom level. This is usually invoked when the window is first mapped. It allows the nodes in a graph, for example, to not appear overly large when zoomed way out by not drawing the unscalable bitmaps.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<WinMap> hideBitmapsIfSmaller()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

hideTextIfTooSmall

hideTextIfTooSmall

SYNOPSIS:

hideTextIfTooSmall ()

DESCRIPTION:

Tells all text graphics to draw only if bigger than or the same size as the text at the current zoom level. This is usually invoked when the window is first mapped. It allows the nodes in a graph, for example, to shrink when zoomed way out by not drawing the fixed-size-font text.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<WinMap> hideTextIfTooSmall()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

hideBitmapsIfSmaller

iCreateLine

SYNOPSIS:

***iCreateLine** (Start | Move | End)*

DESCRIPTION:

Interactively creates a line by rubber-banding.

PARAMETERS (Required):

Start	The events that starts the interactive creation of the line.
Move	The events that interactively changes the size of the line.
End	The event that terminates the interactive creation of the line.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

Shift<Btn1StartDrag>iCreateLine (Start)
Shift<Btn1Drag> iCreateLine (Move)
Shift<Btn1Up> iCreateLine (End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

iCreateRect, iCreateSimpleConn

iCreateRect

SYNOPSIS:

***iCreateRect** (Start | Move | End)*

DESCRIPTION:

Interactively creates a rectangle by rubber-banding.

PARAMETERS (Required):

Start	The events that starts the interactive creation of the rectangle.
Move	The events that interactively changes the size of the rectangle.
End	The event that terminates the interactive creation of the rectangle.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

Shift<Btn1StartDrag>iCreateRect(Start)
Shift<Btn1Drag> iCreateRect(Move)
Shift<Btn1Up> iCreateRect(End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

iCreateSimpleConn, iCreateLine

iCreateRect

iCreateSimpleConn

SYNOPSIS:

***iCreateSimpleConn** (Start | Move | End)*

DESCRIPTION:

Interactively creates a line connection by rubber-banding. The endpoints snap to the closest nodes available.

PARAMETERS (Required):

Start	The events that starts the interactive creation of the line.
Move	The events that interactively changes the size of the line.
End	The event that terminates the interactive creation of the line.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

Btn2StartDragiCreateSimpleConn(Start)
Btn2DragiCreateSimpleConn(Move)
Btn2Up iCreateSimpleConn(End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

METHODS:

The constructor contains a few extra parameters to be used to specify the appearance of the connecting line during

its creation.

```
GICreateSimpleConn(GMsgManager *editor = NULL,
        Boolean use_default_translations = True,
        G_WWIDTH lineWidth = 0, // Set to -1 to ignore.
        G_DWIDTH maxDeviceLWidth = 1,// Set to -1 to ignore.
        char *color = “red”)
```

Before a connection is finalized a REQUEST_NEW_CONNECTION_ACTION message is sent to the destination nodes actions () method for confirmation. If False is returned the connection is deleted. Otherwise it is created and a NEW_CONNECTION_ACTION message is sent to the destination node.

SEE ALSO:

iCreateLine

jumpPan

SYNOPSIS:

jumpPan ***(Left | Right | Up | Down | LeftSide | RightSide | Top | Bottom | LowerLeft | LowerRight | UpperLeft | UpperRight | QuarterLeft | QuarterRight | QuarterUp | QuarterDown)***

DESCRIPTION:

The editing area is panned by one of a number of fixed increments in one of a number of fixed directions.

PARAMETERS (Required):

Left	Pans one whole area left.
Right	Pans one whole area right.
Up	Pans one whole area up.
Down	Pans one whole area down.
LeftSide	Pans all the way to the left edge.
RightSide	Pans all the way to the right edge.
Top	Pans all the way to the top edge.
Bottom	Pans all the way to the bottom edge.
LowerLeft	Pans one whole area to the lower left.
LowerRight	Pans one whole area to the lower right.
UpperLeft	Pans one whole area to the upper left.
UpperRight	Pans one whole area to the upper right.
QuarterLeft	Pans one quarter area left.
QuarterRight	Pans one quarter area right.
QuarterUp	Pans one quarter area up.
QuarterDown	Pans one quarter area down.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

jumpPan(Left)
jumpPan(Right)
jumpPan(Up)
jumpPan(Down)

Ctrl<LeftArrow>	jumpPan(LeftSide)
Ctrl<RightArrow>	jumpPan(RightSide)
Ctrl<UpArrow>	jumpPan(Top)
Ctrl<DownArrow>	jumpPan(Bottom)
<End>	jumpPan(LowerLeft)
<PgDn>	jumpPan(LowerRight)
<Home>	jumpPan(UpperLeft)
<PgUp>	jumpPan(UpperRight)

<LeftArrow>	QuarterLeft”
<RightArrow>	QuarterRight”
<UpArrow>	QuarterUp”
<DownArrow>	QuarterDown”

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

smoothPan, smoothPan2, onePointPan

locatorToolOptions

SYNOPSIS:

locatorToolOptions ***(optionname [, True | False | Toggle])***

DESCRIPTION:

Modifies the behavior of Locators and Scopes and Views.

PARAMETERS (Required):

optionname Name of an option. The following are currently supported:

attachBoxToMouse
locatorHasBox
maintainMagnification
anotherGraphicsView

attachBoxToMouse:
Specifically for editors with a Scope. Causes the scope to be continuously updated to display the graphics in the area of the current mouse pointer position.

locatorHasBox:
Specifically for editors with a Scope. Causes the scope to be represented by a visible box (rectangle) in the source editor.

maintainMagnification:
Specifically for editors with a Scope. Causes the scope to be automatically adjusted after zooms in the source editor so that a constant magnification factor is applied.

anotherGraphicsView:
Specifically for editors with a Locator. Causes the locator to have a copy of the source editor graphics as a background.

PARAMETERS (Optional):

True	The option is applied to the editor(s).
False	The option is not applied to the editor(s).
Toggle	The option is toggled from True to False and back again each time Toggle is invoked. (when not specified is set to Toggle).

CONTAINER COMPONENT:

Editor
Locator
Scope
Magnifier
View

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

Class: PANEL, Error: Missing Parameter
Class: PANEL, Error: Bad Parameter.

CAVEATS:

None.

SEE ALSO:

GLocator, GMagnifier

notifyNodeOfEvent

SYNOPSIS:

notifyNodeOfEvent ()

DESCRIPTION:

Notifies all graphical display objects underneath the mouse cursor that the event to which this Message Handler has been assigned has occurred. This is used primarily by the ‘C’ based interface programmer.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

sendMsgToObjectUnderCursor

METHODS:

set_actionToSendToNode

Specifies what the name of the action is to be. This action name is sent to the nodes actions () method when the above event occurs.

notifyNodeOfEvent

void set_actionToSendToNode (char *act)

onePointPan

SYNOPSIS:

onePointPan ()

DESCRIPTION:

The window is panned such that the graphics that were underneath the mouse cursor position before the pan is now in the center of the editing area.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>g onePointPan()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

smoothPan, smoothPan2, jumpPan

selectArea

SYNOPSIS:

selectArea ***(Start | Move | End)***

DESCRIPTION:

The user rubberbands a rectangle. All the graphics inside the rectangle is selected.

PARAMETERS (Required):

Start	The event that starts the interactive creation of the rectangle.
Move	The events that interactively changes the size of the rectangle.
End	The event that terminates the interactive creation of the rectangle.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

Ctrl<Btn1StartDrag> selectArea(Start)
Ctrl<Btn1Drag> selectArea(Move)
Ctrl<Btn1Up> selectArea(End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

cumulativeSelect, deselectAll, alternatingSelect, doubleSelect

sendMsgToObjectUnderCursor

SYNOPSIS:

sendMsgToObjectUnderCursor ***([msgToSend])***

DESCRIPTION:

Notifies the topmost graphical display object underneath the mouse cursor that the event to which this Message Handler has been assigned has occurred. The object must be visible and selectable. The named action is sent to the object's actions () method.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

msgToSend	The action name to send to the object's actions () method. (Default action name is PICKED_ACTION).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

notifyNodeOfEvent

setBackgroundColor

SYNOPSIS:

setBackgroundColor** **(color name)

DESCRIPTION:

Sets the background color of the editing area to the color specified by the color name.

PARAMETERS (Required):

color name The name of the color to use.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

^<key>b setBackgroundColor(black)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

setSize, hide

setSize

SYNOPSIS:

***setSize** (width, height)*

DESCRIPTION:

Sets the size (width and height) of the editor in device coordinates.

PARAMETERS (Required):

width	The width of editor in pixels.
height	The height of editor in pixels.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

Class: EDITOR, Error: Missing Parameter

CAVEATS:

None.

SEE ALSO:

hide, setBackgroundColor

setSourceEditor

SYNOPSIS:

***setSourceEditor** (name of editor)*

DESCRIPTION:

Tells a Locator, Scope, View or Magnifier where the editor is whose graphics it is viewing. I.e. Specifies the editor which contains the graphics which is to have another ‘view’.

PARAMETERS (Required):

name	The name of an Editor (as specified in the description file).

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Locator
Scope
Magnifier
View

DEFAULT TRANSLATIONS:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

GLocator, GMagnifier

showNodes

SYNOPSIS:

***showNodes** (True, False, Toggle)*

DESCRIPTION:

To display or not to display the nodes of a graph. If not displayed, then only the connections between the nodes will be visible.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

True	The nodes are displayed.
False	The nodes are not displayed.
Toggle	If the nodes are not displayed then they are drawn, if they are displayed then they are hidden. (when not specified Toggle is chosen).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>n showNodes (Toggle)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

draw, showOnlyRelatives

showNodes

showOnlyRelatives

SYNOPSIS:

showOnlyRelatives ***(Children | Parents | All)***

DESCRIPTION:

For graphics composed of Graphs, this will hide all but the specified relatives of any selected graphic objects. Invoking this twice will display all the graphics again (i.e. this toggles the display automatically upon repeated invocation).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

Children	The nodes that are not children of the selected nodes are hidden.
Parents	The nodes that are not parents of the selected nodes are hidden.
All	The nodes that are not parents of or children of the selected nodes are hidden. T (when not specified All is chosen).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<key>k	showOnlyRelatives (Children)
<key>p	showOnlyRelatives (Parents)
<key>h	showOnlyRelatives (All)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

showOnlyRelatives

None.

SEE ALSO:

draw, showNodes

simpleDrag

SYNOPSIS:

simpleDrag ***(Start | Move | End)***

DESCRIPTION:

Moves the graphics object underneath the mouse cursor interactively in response to the mouse. It assumes there are no connections to the graphics object and if there are, the connections are not automatically rubber-banded to maintain connectivity.

PARAMETERS (Required):

Start	The event that starts the move.
Move	The event that actual causes the movement.
End	The event that terminates the drag operation.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn1StartDrag> simpleDrag(Start)
<Btn1Drag> simpleDrag(Move)
<Btn1Up> simpleDrag(End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

simpleDrag

treeNodeDrag

smoothPan

SYNOPSIS:

smoothPan ()

DESCRIPTION:

The window is panned interactively in the direction opposite mouse cursor movement as if one was dragging around the viewport which was looking down on a section of the graphics.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn3Drag> smoothPan()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

smoothPan2, onePointPan, jumpPan

smoothPan2

SYNOPSIS:

smoothPan2 ()

DESCRIPTION:

The window is panned interactively in the direction opposite mouse cursor movement as if one was dragging around the viewport which was looking down on a area of the graphics. The area of the graphics that is displayed is proportional to the distance the mouse cursor is from the edge of the editor viewport (i.e. the scrollbar 'buttons' are continuously updated such that they center on the mouse cursor position).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn3Drag> smoothPan()

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

smoothPan, onePointPan, jumpPan

treeNodeDrag

SYNOPSIS:

treeNodeDrag ***(Start | Move | End)***

DESCRIPTION:

Moves the graphics object underneath the mouse cursor interactively in response to the mouse. It takes account of the possibility that there are connections to the graphics object and 'rubberbands' the connections as the move occurs, keeping the object connected at all times.

PARAMETERS (Required):

Start	The event that starts the move.
Move	The event that actual causes the movement.
End	The event that terminates the drag operation.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn1StartDrag> treeNodeDrag(Start)
<Btn1Drag> treeNodeDrag(Move)
<Btn1Up> treeNodeDrag(End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

treeNodeDrag

simpleDrag

zoom

SYNOPSIS:

zoom ***(In | Out)***

DESCRIPTION:

Zoom (in: magnify, out: de-magnify) graphics, centering the result around the graphics that was underneath the mouse cursor position before the zoom.

PARAMETERS (Required):

In	The event that causes magnification.
Out	The event that causes demagnification.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn2Click> zoom(In)
<Btn3Click> zoom(Out)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

zoomAroundCursor, zoomThruArea

zoomAroundCursor

SYNOPSIS:

***zoomAroundCursor** (In | Out [, amount])*

DESCRIPTION:

Zoom, preserving the current mouse cursor position with respect to the underlying graphics. I.E. the graphics object(s) underneath the cursor before the zoom is underneath the cursor after the zoom.

PARAMETERS (Required):

In	The event that causes magnification.
Out	The event that causes demagnification.

PARAMETERS (Optional):

amount	A floating point number indicating the amount of magnification involved in each zoom operation. (Default amount equals 1.5).

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn2Click> zoom(In)
<Btn3Click> zoom(Out)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

zoom, zoomThruArea

adjust_universeToBeAnIntegralZoomLevel

Return # of zoom levels between zoomed all the way out (all of universe visible) and current zoom level.

int adjust_universeToBeAnIntegralZoomLevel (VEditor *editor);

get_zoomFactor

Return amount of magnification between each zoom level.

float get_zoomFactor ()

set_maxZoomLevel

Specify maximum number of zoom levels allowed between zoomed all the way out (all of universe visible) and zoomed all the way in (maximum zoom level).

void set_maxZoomLevel (int level)

set_zoomFactor

Set amount of magnification between each zoom level.

void set_zoomFactor (float zf)

zoomThruArea

SYNOPSIS:

***zoomThruArea** (In | Out, Start | Move | End)*

DESCRIPTION:

The user rubberbands a rectangle. If it is a zoom in, then the window is zoomed such that the graphics in the rectangle fills the editing area. If it is a zoom out, the current window is zoomed such that the previous window contents now fill only the area of the rectangle.

PARAMETERS (Required):

In	The event that causes magnification.
Out	The event that causes demagnification.
Start	The events that starts the interactive creation of the rectangle that is to be used.
Move	The events that interactively changes the size of the rectangle.
End	The event that terminates the interactive creation of the rectangle that is to be used. This the triggers the actual zooming operation.

PARAMETERS (Optional):

None.

CONTAINER COMPONENT:

Editor

DEFAULT TRANSLATIONS:

<Btn2StartDrag> zoomThruArea(In, Start)
<Btn2Drag> zoomThruArea(In, Move)
<Btn2Up> zoomThruArea(In, End)
<Btn3StartDrag> zoomThruArea(Out, Start)
<Btn3Drag> zoomThruArea(Out, Move)
<Btn3Up> zoomThruArea(Out, End)

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS:

None.

CAVEATS:

None.

SEE ALSO:

zoomAroundCursor, zoomThruArea

CHAPTER 4

Graphics Data Structures: Graphs, Trees and Lists

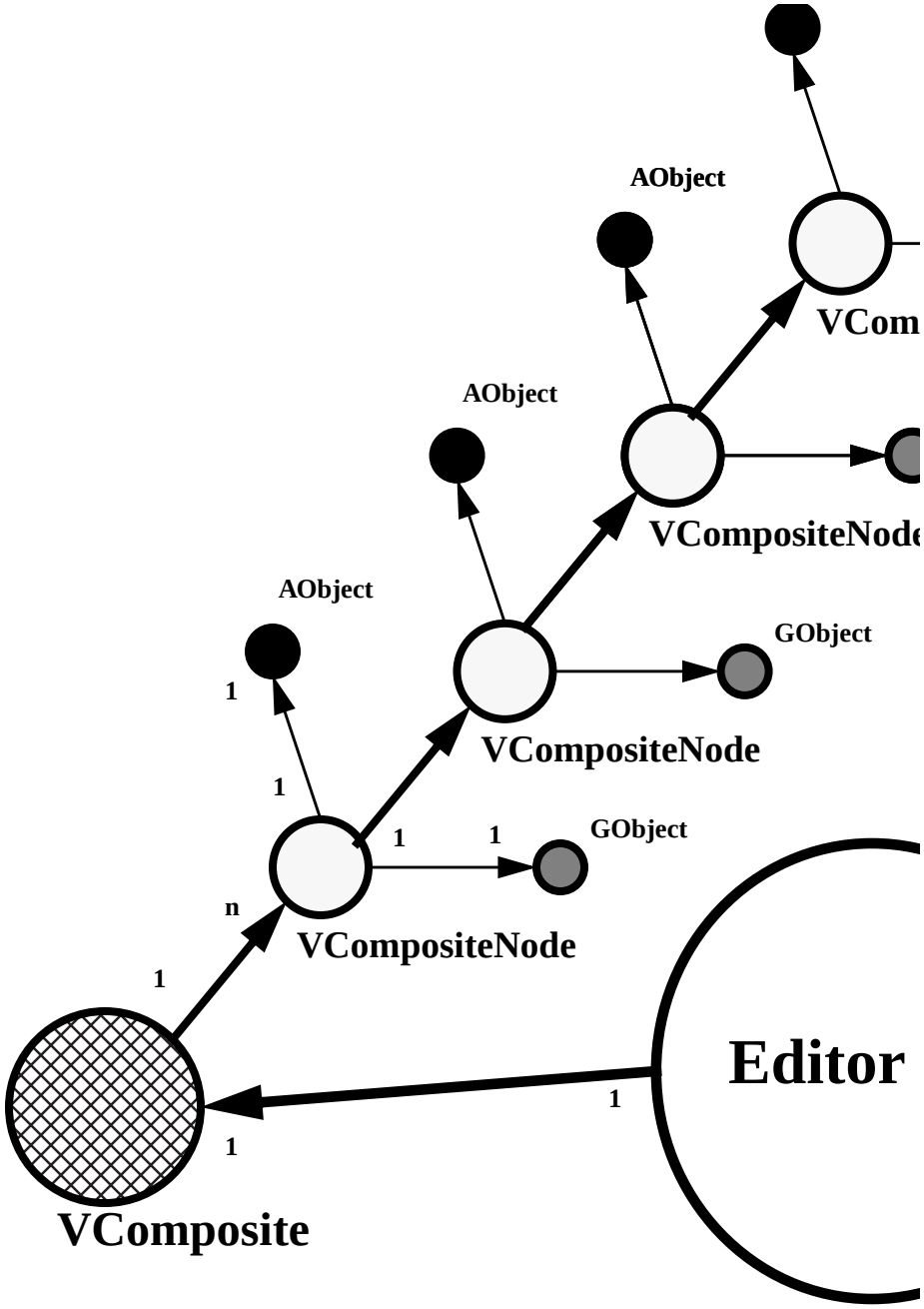

FIGURE 3 The Graphics Data Structure Diagram

AObject

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

This class is the base class for all application objects in this architecture. Application objects are those data objects that are specific to the particular application and are typically outside the domain of a graphics library.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VComposite, GObject, VObject

METHODS:

actions

This method is called when some type of event has occurred. This call is usually from a GMsgHandler. A return of True indicates that all is well. This method allows the main application to be informed of events during which it needs to do additional processing.

src	Is the manager/container of the calling message handler.
node	Is the node in the graph that is being

acted on by the message handler.

action Is what the message handler is about to do or has done.

userdata Is non-null if the particular message handler supports the concept of userdata.

The types of actions currently supported by most message handlers are (see vobject.h for the most up-to-date information):

SELECTED_ACTION
DESELECTED_ACTION
DOUBLE_SELECTED_ACTION
REQUEST_DELETE_ACTION
DELETE_ACTION
MOVED_ACTION
PICKED_ACTION
NEW_NODE_ACTION
REQUEST_NEW_CONNECTION_ACTION
NEW_CONNECTION_ACTION

virtual Boolean actions (GMsgManager *src, VCompositeNode *node, char * action, char *userdata = NULL)

actions

This method is called when some type of event has occurred. This call is usually from a GMsgHandler. A return of True indicates that all is well. This method allows the main application to be informed of events during which it needs to do additional processing. This particular method is called when action that has or is about to be taken by the message handler involves two nodes (a source and destination) and the object that connects them together.

src Is the manager/container of the calling message handler.

action Is what the message handler is about to do or has done.

srcnode Is a node in the graph that is involved in the action by the message handler.

destnode Is a node in the graph that is involved in the action by the message handler.

connection Is the connection that links the srcnode and destnode.

virtual Boolean actions (GMsgManager *src, char *action, VCompositeNode **srcnode, VCompositeNode **destnode, VCompositeNode *connection)

AObject

copy

Makes a copy of this application object and returns a pointer to the copy.

virtual AObject *copy ()

get_name

Returns the name of this application object.

virtual char *get_name ()

GFaceFunctor

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class that specifies the interface to functors
that the programmer passes to various GFaceComposite
methods to manipulate elements in a VComposite graph.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VCompositeGFace, VObject, VComposite, VCompositeNode

METHODS:

VComposite

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of graphs (i.e. lists, directed graphs, etc.). This therefore contains a list of nodes in a graph and which reference a VObject, which manages the relationships between the application and graphics domain. The methods (mainly of traversal) of VComposite are supported by all graphs in the system.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

VObject*	The VObject that this graph will reference.

COMPONENT NAME:

VComposite.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace

METHODS:

append_node

Add node to the end of the graph.

virtual void append_node (VCompositeNode *node);

append_obj

Creates a new node and assigns the VObject obj to it and then appends the node to the graph.

virtual void append_obj (VObject *obj);

del_node

Remove a node from the graph and delete it without deleting its contents (i.e. the vobject and its associated graphics, application and linkage are all untouched).

virtual void del_node (VCompositeNode *node);

del_obj

Delete node containing the given data and the data.

virtual void del_obj (VObject *data);

find_data

Returns the node in the graph that has been assigned the given VObject or NULL if no node is found.

virtual VCompositeNode*find_data (VObject *obj);

get_appobj

Return the address of the application object associated with this graph.

virtual AObject *get_appobj ();

get_extrema

Returns the total extrema of all graphics objects referred to by all nodes in the graph.

virtual void get_extrema (VEditor *veditor, GExtrema *extrema);

get_first

Return the first element in the graph.

virtual VCompositeNode *get_first ();

get_grobj

Return the address of the graphics object associated with this graph. Note that this graphics object is NOT a GObject. Most drawing operations should go through the VCompositeGFace interface if this graph is being displayed in multiple editors.

virtual VCompositeGFace *get_grobj ();

get_hints

Return hints about the type of graph this is. Used by the placement routines to detect differences like single parent and multiple parent trees in a VDirGraph implementation. Used only for efficiency.

int get_hints ();

get_last

Return the last element in the graph.

virtual VCompositeNode *get_last ();

get_length

Return the number of elements in the graph.

virtual int get_length ();

get_linkobj

Return the address of the linkage object associated with this graph.

virtual LObject *get_linkobj ();

get_next

Return the next element in the graph.

virtual VCompositeNode *get_next (VCompositeNode *node);

get_node

Return the nth node where n = index.

virtual VCompositeNode *get_node (int index);

get_nodes

Return a list that contains only nodes of the graph (no connections or ‘arcs’).

virtual VComposite*get_nodes ();

get_object_index

Return the index of the node that has been assigned the given VObject ‘obj’.

virtual int get_object_index (VObject *obj);

get_placer

Return the automatic placement routine that has been assigned to the graph.

VPlacer *get_placer ();

get_prev

Return the previous element in the graph.

virtual VCompositeNode *get_prev (VCompositeNode *node);

get_vobject

Return the address of the VObject associated with this graph.

virtual VObject *get_vobject ();

make_node

Make and return the address of a VComposite node compatible with the type of this VComposite graph. For example, this makes and returns a VUnDirGraphNode * when this is the method of the VUnDirGraph class.

virtual VCompositeNode*make_node ();

purge

Delete all nodes in the graph, leave their contents intact.

virtual void purge ();

remove_node

Remove a node from the graph without deleting it or its contents (i.e. the vobject and its associated graphics, application and linkage are all untouched).

virtual void remove_node (VCompositeNode *node);

VComposite

set_appobj

Assign a application object to this graph.

virtual void set_appobj (AObject *obj);

set_grobj

Assign a graphics object to this graph.

virtual void set_grobj (GObject *obj);

set_hints

Assign hints to the graph.

void set_hints (int hint);

set_linkobj

Assign a linkage object to this graph.

virtual void set_linkobj (LObject *obj);

set_placer

Assign to the graph an automatic placement routine.

void set_placer (VPlacer *placer);

set_vobject

Assign a VObject to this graph.

virtual void set_vobject (VObject *obj);

VCompositeGFace

SYNOPSIS:

***VCompositeGFace** (VComposite *list);*

DESCRIPTION:

Provides graphics support to for VComposite objects. Maintains a list of all the editors (i.e. views) that the associated VComposite appears in. Using this classes methods to act upon and draw objects in the associated list assures that all views are updated simultaneously.

PARAMETERS (Required):

list	The associated VComposite object.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VEditor, GFaceFunctor

METHODS:

append_damaged_objarea

Takes the area of the given object and appends the area to the list of damaged areas in each VEditor the associated VComposite appears in. This is used to improve redraw time when lots of GObjects are being modified at one time and only one redraw at the end of the modifications is actually necessary.

void append_damaged_objarea (GObject *obj);

copy

Unimplemented.

void copy (VEditor *editor, GExtrema *area, VComposite *dest);

copy

Unimplemented.

virtual GObject *copy (VEditor *veditor);

delete_node

Delete the given node (or connection) which resides in the associated VComposite.

void delete_node (VNode *node);

draw

Draw the given editors contents.

void draw (VEditor *editor);

draw

Draw the contents of all the editors the associated VComposite appears in.

void draw ();

draw

Draw the given display object in each of the editors the associated VComposite appears in.

void draw (GObject *obj);

draw_damaged_areas

Redraw the list of damaged areas in each VEditor the associated VComposite appears in.

void draw_damaged_areas ();

draw_objarea

Draws (undraws if drawobj_flag = False) the given object and all objects in the obj's extrema in correct order.

void draw_objarea (GObject *obj, Boolean drawobj_flag = True);

drawarea

Draw the given area in each of the editors the associated VComposite appears in.

void drawarea (GExtrema *area);

get_picklist

Creates and returns address of a list (actually a VIndirectList) of all objects in the associated list that overlay the given area. The caller should destroy the list when done with it.

VComposite * get_picklist (GExtrema *area);

get_source_editor

Return the address of the first registered editor.

VEditor *get_source_editor ();

make_connections_visible_between_visible_nodes

This does what it says. It is used by routines that selectively display and hide various nodes of a graph based on varying heuristics. This routine is then used to display the connections between the nodes that are visible.

void make_connections_visible_between_visible_nodes (Boolean flag);

process_children_of_node

For each child of the given item, invoke the execproc method of the given functor.

void process_children_of_node (VNode *node, GFaceFunctor *func);

process_connections

For each connection in the associated VComposite, invoke the execproc method of the given functor.

void process_connections (GFaceFunctor *func);

process_node

For the given item, invoke the execproc method of the given functor.

void process_node (VNode *node, GFaceFunctor *func);

process_nodes

For each node in the associated VComposite, invoke the execproc method of the given functor.

void process_nodes (GFaceFunctor *func);

process_objs_in_area

For each item in the associated VComposite found that overlays the given area in the given editor, invoke the execproc method of the given functor.

void process_objs_in_area (VEditor *editor, GExtrema *area, GFaceFunctor *func);

process_parents_of_node

For each parent of the given item, invoke the execproc method of the given functor.

void process_parents_of_node (VNode *node, GFaceFunctor *func);

register_editor

Add the given editor to the list of editors the associated VComposite appears in.

void register_editor (VEditor *editor);

set_armable

Set each item in the associated VComposite to be armable.

virtual void set_armable (Boolean flag);

set_selectable

Set each item in the associated VComposite to be selectable.

virtual void set_selectable (Boolean flag);

set_visible

Set each item in the associated VComposite to be visible.

virtual void set_visible (Boolean flag);

undraw

Remove the given editor from the list of editors the associated VComposite appears in.

void undraw (VEditor *editor);

undraw

Undraw the given display object in each of the editors the associated VComposite appears in.

void undraw (GObject *obj);

unregister_editor

Remove the given editor from the list of editors the associated VComposite appears in.

void unregister_editor (VEditor *editor);

VCompositeNode

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of nodes in all kinds of graphs (i.e. VComposites). This node, therefore exists as a node in a graph and references a VObject, which manages the relationships between the application and graphics domain. The methods of this class specifies the interface for all nodes in whatever type of graph they may reside. Note that this node class may also wrap/interface to connections, as well as true node, in a particular implementation of a graph.

S

None.

PARAMETERS (Optional):

VObject*	The VObject that this node will reference.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite

METHODS:

actions

Forward all actions to the action handler of the associated VObject if it exists. Return True in the absence of any errors.

virtual Boolean actions (GMsgManager *src, VCompositeNode *node, char * action);

actions

Forward all actions to the action handler of the associated VObject if it exists. Return True in the absence of any errors.

virtual Boolean actions (GMsgManager *src, VCompositeNode **srcnode, char * action, VCompositeNode **destnode, VCompositeNode *connection);

get_appobj

Return the application object associated with this node.

virtual AObject *get_appobj ();

get_grobj

Return the graphics object associated with this node.

virtual GObject *get_grobj ();

get_linkobj

Return the linkage object associated with this node.

virtual LObject *get_linkobj ();

get_next

Return the next node in the VComposite graph structure. Note this only works with some graphs, use VComposite get_next() for consistent results across different VComposite graph types.

virtual VCompositeNode *get_next () = 0;

get_prev

Return the previous node in the VComposite graph structure. Note this only works with some graphs, use VComposite get_prev() for consistent results across different VComposite graph types.

virtual VCompositeNode *get_prev () = 0;

VCompositeNode

get_type

Return the type of the node. The type can be anything that the implementor of the derived VCompositeNode has chosen to use.

virtual char *get_type () = 0;

get_vobject

Return the VObject associated with this node.

virtual VObject *get_vobject ();

set_appobj

Assign the application object to this node.

virtual void set_appobj (AObject *obj);

set_grobj

Assign the graphics object to this node.

virtual void set_grobj (GObject *obj);

set_linkobj

Assign the linkage object to this node.

virtual void set_linkobj (LObject *obj);

set_vobject

Assign a VObject to this node.

virtual void set_vobject (VObject *obj);

VConnection

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of connections in all kinds of graphs (i.e. VGraphs) that support connectivity. This node inherits functionality from the base class VCompositeNode. The methods of this class specifies the interface for all connections in whatever type of graph they may reside. Note that what is referred to as connections is also referred to as arcs in some of the literature.
This connection object is intrinsically directional, from source node to destination node.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x1, y1, x2, y2 Often a connection is represented visually by assigning a GLine to the VObject associated with this connection. This constructor automates setting the lines endpoints.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VNode

METHODS:

VConnection

get_destination_node

Returns the node on the ‘destination’ side of this connection.

virtual VNode *get_destination_node ()

get_other

Return the node on the other side of this connection from the given node.

VNode *get_other (VNode *node)

get_source_node

Returns the node on the ‘source’ side of this connection.

virtual VNode *get_source_node ()

get_type

Returns the type of this object. Often used as a isNode () method.

virtual char *get_type ()

update_endpoint

Informs this connection that the given node has moved and that the connection should update its graphical representation. This, of course, assumes that the given node is either the source or destination of this connection.

virtual void update_endpoint (VNode *node)

VUnDirGraph

SYNOPSIS:

VUnDirGraph ()

DESCRIPTION:

A graph with the nodes arranged internally in a doubly-linked list but with the semantics of a undirected graph. There is direct support for the connections between the nodes that define the topology of the graph. This node inherits functionality from the base class VGraph.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

VGraph.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection VDirGraph, VDirGraphNode, VDirGraphConnection, VUnDirGraphNode, VUnDirGraphConnection

METHODS:

VDblLinkList

SYNOPSIS:

VDblLinkList ()

DESCRIPTION:

A graph with the nodes arranged internally in a doubly-linked list. There can be connections between the nodes but there is no direct support for it. This node inherits functionality from the base class VGraph.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

VGraph.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection, VDblLinkListNode

METHODS:

VDblLinkListNode

SYNOPSIS:

VDblLinkListNode (VObject *)

DESCRIPTION:

A node for a graph with the nodes arranged internally in a doubly-linked list. There can be connections between the nodes but there is no direct support for it. This node inherits functionality from the base class VNode.

PARAMETERS (Required):

VObject*	The VObject that this node will reference.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VConnection, VDblLinkList

METHODS:

VDirGraph

SYNOPSIS:

VDirGraph ()

DESCRIPTION:

A graph with the nodes arranged internally in a doubly-linked list but with the semantics of a directed graph. There is direct support for the connections between the nodes that define the topology of the graph. This graph inherits functionality from the base class VGraph.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

VGraph.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection VUnDirGraph, VUnDirGraphNode, VUnDirGraphConnection, VDirGraphNode, VDirGraphConnection

METHODS:

VDirGraphConnection

SYNOPSIS:

VDirGraphConnection (
VDirGraphNode *parent, VDirGraphNode *child)

DESCRIPTION:

A connection for a graph with the semantics of a undirected graph. There is direct support for the connections between the nodes that define the topology of the graph. This inherits functionality from the base class VConnection.

PARAMETERS (Required):

parent	The node at the start of the connection.
child	The node at the end of the connection.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection, VUnDirGraph, VUnDirGraphNode, VUnDirGraphConnection, VDirGraphNode, VDirGraph

METHODS:

VDirGraphNode

SYNOPSIS:

VDirGraphNode (VObject *)

DESCRIPTION:

A node for a graph with the nodes arranged internally in a doubly-linked list but with the semantics of a directed graph. There is direct support for the connections between the nodes that define the topology of the graph. This node inherits functionality from the base class VNode.

PARAMETERS (Required):

VObject*	The VObject that this node will reference.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VConnection, VDblLinkList, VDirGraphConnection, VDirGraph

METHODS:

VDirGraphPlacer

SYNOPSIS:

VDirGraphPlacer ();

DESCRIPTION:

Automatic (computerized) placement class for directed graphs. This includes graphs that are trees, directed acyclic graphs (DAG) and graphs which contain multiple instances of trees and DAGs.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VPlacer, VComposite

METHODS:

place

Places the given graph in the given editor according to the given specifications. Objects are placed starting at the upper left hand side of the current editor viewing area and extend down and to the right, most likely off of the viewable area.

Layout: the possible layouts are:

TOPDOWN_LAYOUT
(Root(s) at top and leaves at bottom of editor).

LEFTRIGHT_LAYOUT
(Root(s) at left and leaves at right of editor).

OUTLINE_LAYOUT
(Root(s) at top and left and leaves at right and bottom of editor).

ConnectionStyle: the possible connection styles are:

STRAIGHTLINE_CONNSTYLE_LAYOUT
(Connections are drawn as straight lines from node to node).

RIGHTANGLE_CONNSTYLE_LAYOUT
(Connections are drawn as vertical and horizontal lines from node to node).

STRAIGHT_THEN_FLAIR_CONNSTYLE_LAYOUT
(Connections are drawn as vertical and horizontal lines to a point midway between nodes and then as sloping lines the rest of the way to the child node).

FLAIR_THEN_STRAIGHT_CONNSTYLE_LAYOUT
(Connections are drawn as sloping lines to a point midway between nodes and then as vertical and horizontal lines from node to node).

debuglevel: the possible values for debuglevel are:

0	No debugging.
!=0	An attempt is made to draw each node as it is placed.

virtual void place (VEditor *editor, VComposite *graph, int layout = TOPDOWN_LAYOUT, int connectionStyle = STRAIGHTLINE_CONNSTYLE_LAYOUT, int debuglevel = 0)

VGraph

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of graphs (i.e. lists, directed graphs, etc.). This therefore contains a list of nodes in a graph and which reference a VObject, which manages the relationships between the application and graphics domain. The methods (mainly of traversal) of VComposite are supported by all graphs in the system.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

VObject*	The VObject that this graph will reference.

COMPONENT NAME:

VGraph.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection

METHODS:

del_node

Remove a node from the graph and delete it without deleting its contents (i.e. the vobject and its associated graphics, application and linkage are all untouched).

virtual void del_node (VCompositeNode *node);

get_appobjNode

Return the node that has an application object with the given name (assigned to the nodes VObject).

VNode *get_appobjNode (char *name);

get_connections

Return a list of the connections in this graph.

virtual VComposite*get_connections ()

get_nodes

Return a list of the nodes in this graph.

virtual VComposite*get_nodes ()

VIndirectList

SYNOPSIS:

VIndirectList ()

DESCRIPTION:

A graph with the nodes that reference nodes in another graph. This is used when one wants to make a sublist of VComposite nodes without having to copy them. This graph inherits functionality from the base class VComposite. Most often this is utilized by using the append_obj() and del_node () methods and not by creating the VCompositeNodeWrapper object directly.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

VGraph.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNodeWrapper

METHODS:

VIndirectListNode

SYNOPSIS:

VIndirectListNode ()

DESCRIPTION:

A node that references a node in another VComposite list.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VConnection, VDblLinkList, VDirGraphConnection, VDirGraph, VIndirectList

METHODS:

VNode

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of nodes in all kinds of graphs (i.e. VGraphs) that support connectivity. This node inherits functionality from the base class VCompositeNode. The methods of this class specifies the interface for all nodes in whatever type of graph they may reside. Note that what is referred to as connections is also referred to as arcs in some of the literature.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

VObject*	The VObject that this node will reference.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeNode, VConnection

METHODS:

get_child

Returns the child of this node. If there is more than one this returns an arbitrary child. A child is defined as sharing a connection with this object with the child on the ‘destination’ side of the connection and this node on the ‘source’ side of the connection.

virtual VNode *get_child ()

get_connectionTo

Returns the connection, if any, between this node and the given node.

virtual VConnectionNode *get_connectionTo (VNode *other);

get_connections

Returns the list of connections to this node.

virtual VConnectionList*get_connections ()

get_nextConnection

Return the connection following the given one in an internally maintained list.

virtual VConnectionNode*get_nextConnection (VConnectionNode *conn)

get_nextParent

Returns the parent following the given one in an internally maintained list.

virtual VNode *get_nextParent (VNode *n)

get_nextSibling

Returns the child following the given one in an internally maintained list.

virtual VNode *get_nextSibling (VNode *n)

get_parent

Returns the parent of this node. If there is more than one this returns an arbitrary parent. A parent is defined as sharing a connection with this object with the parent on the ‘source’ side of the connection and this node on the ‘destination’ side of the connection.

virtual VNode *get_parent ()

get_prevConnection

Return the connection preceding the given one in an internally maintained list.

virtual VConnectionNode*get_prevConnection (VConnectionNode *conn)

get_prevParent

Returns the parent preceding the given one in an internally maintained list.

virtual VNode *get_prevParent (VNode *n)

get_prevSibling

Returns the child preceding the given one in an internally maintained list.

virtual VNode *get_prevSibling (VNode *n)

get_type

Returns the type of this object. Often used as a isNode () method.

virtual char *get_type ()

is_connectedTo

Returns whether this node is connected to the given node (i.e. whether the two nodes share a connection.

virtual Boolean is_connectedTo (VNode *other);

set_position

Set the position of the node.

virtual void set_position (VEditor *veditor, G_WCOORD x, G_WCOORD y);

translate

Translate the position of the node.

virtual void translate (VEditor *veditor, G_WCOORD tx, G_WCOORD ty);

VObject

SYNOPSIS:

VObject ()

DESCRIPTION:

This class is the basis for every atomic entity in this architecture. Its objects reference a application aspect (of the class ‘AObject’), a graphics aspect (of the class ‘GObject’), and a constraint, link, aspect (of class ‘Lobject’).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

LObject is undefined and is only a opaque handle. LObject is used only in the Software Farm VisualADE product.

SEE ALSO:

VComposite, GObject, AObject, LObject

METHODS:

actions

Forward all actions to the action handler of the associated AObject if it exists. Return True in the absence of any errors.

virtual Boolean ***actions (GMsgManager *src, VCompositeNode *node, char * action, char *userdata = NULL);***

actions

Forward all actions to the action handler of the associated AObject if it exists. Return True in the absence of any errors.

virtual Boolean ***actions (GMsgManager *src, VCompositeNode **srcnode, char * action, VCompositeNode **destnode, VCompositeNode *connection);***

get_appobj

Returns the previously assigned application object.

virtual AObject ****get_appobj ()***

get_grobj

Returns the previously assigned graphics object.

virtual GObject ****get_grobj ()***

get_linkobj

Returns the previously assigned linkage object.

virtual LObject ****get_linkobj ()***

set_appobj

Assigns a application object.

virtual void ***set_appobj (AObject *app)***

set_grobj

Assigns a graphics object.

virtual void ***set_grobj (GObject *gr)***

set_linkobj

Assigns a linkage object.

virtual void set_linkobj (LObject *l)

VPlacer

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class for all kinds of automatic placement classes whose instances are assigned to all kinds of graphs (i.e. VComposites). Often, though, placers are specific to a specific kind of VComposite.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VDirGraphPlacer, VComposite

METHODS:

VUnDirGraphConnection

SYNOPSIS:

VUnDirGraphConnection (
VUnDirGraphNode *source, VUnDirGraphNode *dest)

DESCRIPTION:

A connection for a graph with the semantics of a undirected graph. There is direct support for the connections between the nodes that define the topology of the graph. This inherits functionality from the base class VConnection.

PARAMETERS (Required):

source	The node at the start of the connection.
destination	The node at the end of the connection.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection, VDirGraph, VDirGraphNode, VDirGraphConnection, VUnDirGraphNode, VUnDirGraph

METHODS:

VUnDirGraphNode

SYNOPSIS:

VUnDirGraphNode (VObject *)

DESCRIPTION:

A node for a graph with the nodes arranged internally in a doubly-linked list but with the semantics of a undirected graph. There is direct support for the connections between the nodes that define the topology of the graph. This node inherits functionality from the base class VNode.

PARAMETERS (Required):

VObject*	The VObject that this node will reference.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

VObject, VComposite, VCompositeGFace, VNode, VConnection VDirGraph, VDirGraphNode, VDirGraphConnection, VUnDirGraph, VUnDirGraphConnection

METHODS:

VUnDirGraphNode

CHAPTER 5

Graphics Display Primitives: Lines, Circles and the Rest

GAnnotatedIcon

SYNOPSIS:

GAnnotatedIcon ()

DESCRIPTION:

An icon with associated text. The text may appear in any of 8 positions near the icon. The icon and text are considered one entity and are moved and drawn together.
The eight positions currently supported are:

CENTER_POSITION
LEFT_POSITION
RIGHT_POSITION
BOTTOM_POSITION
TOP_POSITION
TOPLEFT_POSITION
BOTTOMLEFT_POSITION
TOPRIGHT_POSITION
BOTTOMRIGHT_POSITION

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GText, GImage, GAnnotatedObj, GIcon, GObjInBox

SPECIAL METHODS:

get_annotation

Return any text that may exist at the given position.

virtual char *get_annotation (int location);

get_extrema

Return the union of the extrema of the non-NULL text strings.

virtual void get_extrema (VEditor *veditor, GExtrema *extrema);

get_hasbox

Return whether the icon has a box around it.

Boolean get_hasbox ()

get_icon

Return the associated icon.

GIcon *get_icon ()

get_image

Return the image associated with the icon.

virtual GImage *get_image ()

get_text

Return any text that may exist at the given position. (This method is provided for backwards-compatibility only).

GText *get_text (int location);

set_annotation

Assign text to the icon at the given position.

virtual void set_annotation (VEditor *veditor, char *string, int location);

set_hasbox

Specify whether the icon has a box around it.

void set_hasbox (Boolean flag)

set_image

Assign an image for the icon.

virtual void set_image (VEditor *ed, GImage *image);

set_image

Assign an image for the icon and position it.

void set_image (VEditor *veditor, GImage *image, G_WCOORD x, G_WCOORD y);

set_text

Assign text to the icon at the given position. (This method is provided for backwards-compatibility only).

void set_text (VEditor *veditor, char *string, int location);

GAnnotatedObj

SYNOPSIS:

GAnnotatedObj ()

DESCRIPTION:

A display object with an associated display object(s). The associated display object (annotation) may appear in any of 8 positions near the icon. The icon and text are considered one entity and are moved and drawn together. The eight positions currently supported are:

CENTER_POSITION
LEFT_POSITION
RIGHT_POSITION
BOTTOM_POSITION
TOP_POSITION
TOPLEFT_POSITION
BOTTOMLEFT_POSITION
TOPRIGHT_POSITION
BOTTOMRIGHT_POSITION

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GShadowRect, GStyledRect

SPECIAL METHODS:

get_extrema

Return the union of the extrema of the non-NULL Annotations.

virtual void get_extrema (VEditor *veditor, GExtrema *extrema);

GAttributes

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class attribute functionality for all graphics display primitives. This class manages the storage and read/write access to attributes such as color, line width, write mode that each graphics display primitive object has associated with it.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Inherited classes:

GGenAttrInterface, GLineAttrInterface

Other base classes:

GGeometry, GBehavior, ObjWithClassType

METHODS:

GBehavior

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class behavior functionality for all graphics display primitives. Manages storage and read/write access to information about the current state and allowed behavior that each graphics display primitive object has associated with it.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Other base classes:

GGeometry, GAttributes, ObjWithClassType

METHODS:

is_armable

Return whether or not this object visually reacts to the mouse as it passes over it.

virtual Boolean is_armable ()

is_armed

Return whether or not this object is visibly reacting to the presence of the mouse pointer.

virtual Boolean is_armed ()

is_droponable

Return whether or not this object allows another object to be moved and dropped on it by the user.

virtual Boolean is_droponable ()

is_locked

Return whether or not this objects attributes are locked against any changes (only visibility can be locked at this time).

virtual Boolean is_locked ()

is_movable

Return whether or not the user can move this object.

virtual Boolean is_movable ()

is_selectable

Return whether or not the user can select this object.

virtual Boolean is_selectable ()

is_selected

Return whether or not this object is selected.

virtual Boolean is_selected ()

is_visible

Return whether or not this object is visible.

virtual Boolean is_visible ()

set_armable

Specify whether or not this object visually reacts to the mouse as it passes over it.

virtual void set_armable (Boolean flag)

set_armed

Specify whether or not this object is visibly reacting to the presence of the mouse pointer.

virtual void set_armed (Boolean flag)

set_droponable

Specify whether or not this object allows another object to be moved and dropped on it by the user.

virtual void set_droponable (Boolean flag)

set_locked

Specify whether or not this objects attributes are locked against any changes (only visibility can be locked at this time).

virtual void set_locked (Boolean flag)

set_movable

Specify whether or not the user can move this object.

virtual void set_movable (Boolean flag)

set_selectable

Specify whether or not the user can select this object.

virtual void set_selectable (Boolean flag)

set_selected

Specify whether or not this object is selected.

virtual void set_selected (Boolean flag)

set_visible

Specify whether or not this object is visible.

virtual void set_visible (Boolean flag)

GBehavior

GCircle

SYNOPSIS:

GCircle ()

***GCircle** (G_WCOORD x, G_WCOORD y, G_WCOORD radius)*

DESCRIPTION:

A circle display object. The circle may appear in any size, color, line style, or filled. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x, y	The coordinates of the center of the circle in world coordinates.
radius	The radius of the circle in world coordinates.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject

SPECIAL METHODS:

get_radius

Return the current value of the radius.

G_WWIDTH get_radius (VEditor *veditor)

pick

Return TRUE if the location touches the rectangle formed by the bounds of the circle.

virtual Boolean pick (VEditor *veditor, G_WCOORD pickx, G_WCOORD picky);

set_radius

Assign the given value to the radius of the circle.

void set_radius (VEditor *veditor, G_WWIDTH r)

GGeometry

SYNOPSIS:

Not instantiated directly.

DESCRIPTION:

Base class geometric functionality for all graphics display primitives. Collects the methods that inquire and modify the geometric quantities that each graphics display primitive object has associated with it.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Other base classes:

GAttributes, GBehavior, ObjWithClassType

METHODS:

get_extrema

Get the extent of the object, in the given editor, in world coordinates.

virtual void get_extrema (VEditor *veditor,
G_WCOORD *xmin, G_WCOORD *ymin,
G_WCOORD *xmax, G_WCOORD *ymax);

get_extrema

Get the extent of the object, in the given editor, into a given extrema object.

virtual void get_extrema (VEditor *veditor, GExtrema *extrema);

get_position

Return the position of the object in world coordinates.

virtual void get_position (VEditor *veditor, G_WCOORD *tx, G_WCOORD *ty);

get_position

Return the position of the specified reference point of the object in world coordinates.

virtual void get_position (VEditor *veditor, int position_number,
G_WCOORD *tx, G_WCOORD *ty);

get_positionX

Return the X position of the specified reference point of the object in world coordinates.

virtual G_WCOORD get_positionX (VEditor *veditor, int position_number);

get_positionY

Return the Y position of the specified reference point of the object in world coordinates.

virtual G_WCOORD get_positionY (VEditor *veditor, int position_number);

pick

Return True if the point (x, y) touches this object.

virtual Boolean pick (VEditor *veditor, G_WCOORD x, G_WCOORD y);

pick

Return True if the given area touches this object.

virtual Boolean pick (VEditor *veditor, GExtrema *extrema);

set_extrema

Set the extent of the object, with respect to the given editor, in world coordinates.

virtual void set_extrema (VEditor *veditor,
G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax,
G_WCOORD ymax);

set_extrema

Set the extent of the object, with respect to the given editor, to a given extrema object.

virtual void set_extrema (VEditor *veditor, GExtrema *extrema);

set_position

Position the object to the given world coordinates.

virtual void set_position (VEditor *veditor, G_WCOORD tx, G_WCOORD ty);

set_position

Set the position of the specified reference point of the object in world coordinates. position_number == -1 is last position (i.e. of polyline).

virtual void set_position (VEditor *veditor, int position_number,
G_WCOORD tx, G_WCOORD ty);

set_positionX

Set the X position of the specified reference point of the object in world coordinates.

virtual void set_positionX (VEditor *veditor, int position_number,
G_WCOORD x);

set_positionY

Set the Y position of the specified reference point of the object in world coordinates.

virtual void set_positionY (VEditor *veditor, int position_number,
G_WCOORD y);

translate

Translate the object by the given world coordinates.

virtual void translate (VEditor *veditor, G_WCOORD tx, G_WCOORD ty);

translate

Translate the specified reference point of the object by the given world coordinates.

virtual void translate (VEditor *veditor, int position_number, G_WCOORD tx, G_WCOORD ty);

GGrid

SYNOPSIS:

GGrid ()

GGrid (int orientation, G_WWIDTH hstep, G_WWIDTH vstep, G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)

DESCRIPTION:

A rectangular grid display object. The grid may appear in any size, color, line style. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

orientation Whether the lines of the grid are be drawn horizontally, vertically or both.
Possible values are:

HORIZONTAL_GRID
(lines are drawn horizontally).

VERTICAL_GRID
(lines are drawn vertically).

HORIZONTAL_GRID + VERTICAL_GRID

hstep The distance between the vertical lines in world coordinates.

vstep The distance between the horizontal lines in world coordinates.

xmin, ymin, xmax, ymax The coordinates of the lower left and upper right corners of the grid's rectangular boundaries.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GRect, GLine

SPECIAL METHODS:

get_hstep

Return the current distance between the vertical lines.

G_WWIDTH get_hstep ()

get_orientation

Return the current orientation of the grid lines.

int get_orientation ()

get_vstep

Return the current distance between the horizontal lines.

G_WWIDTH get_vstep ()

get_vstep

Return the current distance between the horizontal lines.

void get_vstep (G_WWIDTH step)

set_hstep

Assign the current distance between the vertical lines.

void set_hstep (G_WWIDTH step)

set_orientation

Set the current orientation of the grid lines.

void set_orientation (int orient)

GIcon

SYNOPSIS:

GIcon ()

GIcon (GImage *image, G_WCOORD x, G_WCOORD y)

DESCRIPTION

A rectangular bitmap/pixmap display object. The icon is not resizable at this time. In general, all the methods specified by the GObject base class have been implemented for this class. There is also an optional rectangle (box) that is drawn around the outside of the icon in the foreground color.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

image	The image object that contains the pixmap or bitmap that is to be drawn.
x, y	The location of the center of the icon in world coordinates.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GImage

SPECIAL METHODS:

GIcon

get_inbox

Return whether or not there is a rectangle around the outside of the icon.

Boolean get_inbox ()

set_inbox

Set whether or not there is a rectangle around the outside of the icon.

void set_inbox (Boolean flag)

GImage

SYNOPSIS:

GImage (Display *display, char *name);

GImage (Display *display, char *imagedata,
G_DWIDTH width, G_DWIDTH height, int depth, int bytewidth);

DESCRIPTION

A bitmap/pixmap object. This is usually not displayed directly
but is contained within (i.e. referenced by) a GIcon object.
resizable at this time. Note that this is NOT a class derived
from GObject, and all the methods specified
by the GObject class have NOT been implemented for this class.

PARAMETERS (Required):

display	The display object to be used.

PARAMETERS (Optional):

name	The name of the file that contains bitmap/pixmap data to be loaded into this GImage object. X Window system standard .xpm and .xbm file formats are supported.
imagedata	The data representing the bitmap. This is most often used when a non-standard bitmap is to be read, in which case the programmer writes a file loader/bitmap extractor routine and then passes the data in as the 'imagedata' parameter.
width, height	The width and height of the bitmap in pixels.
depth	The depth of the bitmap (i.e. number of bits per pixel).
bytewidth	The number of bytes it takes to define a pixel row in the bitmap data.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GIcon

SPECIAL METHODS:

None.

GLine

SYNOPSIS:

GLine ()

GLine (G_WCOORD x1, G_WCOORD y1, G_WCOORD x2, G_WCOORD y2)

DESCRIPTION

A line display object. The line may appear in any size, color, line style, or width. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x1, y1, x2, y2	The coordinates of the start and end points of the line in world coordinates.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GPLine, GOrthoPline, GPolygon

SPECIAL METHODS:

GObjInBox

SYNOPSIS:

GObjInBox ()

GObjInBox (VEditor *veditor, GObject *obj);

DESCRIPTION

A autoplace display object. This object takes two display objects, one a general display object and one is a rectangle. It places the general display object inside the rectangle and constrains its location to the center of the rectangle. A margin between the outside of the center object and the rectangle may be specified (it defaults to zero).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

veditor	The editor this will be displayed in. May be NULL if unknown.
obj	The object that will be in the 'box'.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GRect, GWidth, GAnnotatedObj, GAnnotatedIcon, GStyledRect, GShadowRect

SPECIAL METHODS:

get_box

Return the associated rectangle (box).

GRect *get_box ()

get_boxShadowWidth

Returns the width object of the box if the box is a ShadowRect object.

GWidth *get_boxShadowWidth ()

get_object

Return the centered object.

GObject *get_object ()

set_box

Assign the associated rectangle (box).

void set_box (GRect *o)

set_boxIndented

Assigns the given style to the box if the box is a StyledRect object.

void set_boxIndented (int style)

set_boxShadowColor

Assigns the given color to the box if the box is a ShadowRect object.

void set_boxShadowColor (int style)

set_boxShadowWidth

Assigns the given width to the box if the box is a ShadowRect object.

void set_boxShadowWidth (G_DWIDTH width)

set_boxWidth

Assigns the given width to the box.

void set_boxWidth (int width)

set_object

Assign the centered object.

void set_object (GObject *o)

GObject

SYNOPSIS:

Not instantiated directly.

DESCRIPTION

Base class functionality for all graphics display primitives.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Base classes:

GGeometry, GAttributes, GBehavior, ObjWithClassType

Heavily used classes:

GExtrema, VEditor

and the primitives:

GAnnotatedIcon, GAnnotatedObj, GCircle, GGrid, GIcon, GLine, GObjInBox, GPline, GOrthoPline, GPolygon, GRect, GShadowRect, GStyledRect, GText, GTextFixedSizeFont

GObject

METHODS:

action

Method to handle actions at the graphics level instead of the application level (unsupported functionality).

virtual void action (char *actiontype);

arm

Visually highlight the object (i.e. in response to the user positioning the mouse cursor over this object).

virtual void arm (VEditor *veditor);

arm_dehighlight

Remove mouse cursor highlighting from this object.

virtual void arm_dehighlight (VEditor *veditor);

arm_highlight

Visually alter (highlight) the object to indicate that the mouse cursor is over it.

virtual void arm_highlight (VEditor *veditor);

copy

Make a copy of this object and return a pointer to it.

virtual GObject *copy (VEditor *veditor);

dearm

Visually restore the object (i.e. in response to the user moving the mouse cursor away from this object).

virtual void dearm (VEditor *veditor);

deselect

Restore the object (i.e. in response to the user clicking the mouse select button over this object in a ‘selected’ state).

virtual void deselect (VEditor *veditor);

draw

Draw the object and any associated annotations in the appropriate write mode.

virtual void draw (VEditor *veditor);

g_get_centerx

Return the X position of the center of the object.

virtual G_WCOORD g_get_centerx ();

g_get_centery

Return the Y position of the center of the object.

virtual G_WCOORD g_get_centery ();

g_set_centerx

Set the position of the center of the object to the given X coordinate, ignoring any constraints.

virtual void g_set_centerx (G_WCOORD x);

g_set_centery

Set the position of the center of the object to the given Y coordinate, ignoring any constraints.

virtual void g_set_centery (G_WCOORD y);

g_set_height

Set the height of the object using world coordinates, ignoring any constraints.

virtual void g_set_height (G_WWIDTH w);

g_set_width

Set the width of the object using world coordinates, ignoring any constraints.

virtual void g_set_width (G_WWIDTH w);

g_set_width

Set the width of the object using device coordinates, ignoring any constraints.

virtual void g_set_width (G_DWIDTH w);

g_set_x1

Set the first reference point of the object to the given X coordinate, ignoring any constraints.

virtual void g_set_x1(G_WCOORD x);

g_set_x2

Set the second reference point of the object to the given X coordinate, ignoring any constraints.

virtual void g_set_x2(G_WCOORD x);

g_set_y1

Set the first reference point of the object to the given Y coordinate, ignoring any constraints.

virtual void g_set_y1(G_WCOORD y);

g_set_y2

Set the second reference point of the object to the given Y coordinate, ignoring any constraints.

virtual void g_set_y2(G_WCOORD y);

get_annotation

Return the given text from the object at the specified position (useful only for GAnnotatedIcon and GAnnotatedObj at this time).

virtual char *get_annotation (int location);

get_font

Return the font of the object.

virtual G_FONT get_font ();

get_image

Return the given image of the object (useful only for GIcon at this time).

virtual GImage *get_image ();

get_numberedColor

Return the color of the specified part of the object.

virtual G_COLORget_numberedColor (int num);

get_style

Return the style of display of the object.

virtual int get_style ();

hide

Make an object invisible by drawing all other objects in its extrema.

virtual void hide (VEditor *editor);

is_hidden

Whether or not this object is hidden.

virtual Boolean is_hidden (VEditor *veditor);

pick

Return True if the point (x, y) touches this object.

virtual Boolean pick (VEditor *veditor, G_WCOORD x, G_WCOORD y);

pick

Return True if the given area touches this object.

virtual Boolean pick (VEditor *veditor, GExtrema *extrema);

reverse_video

Swap foreground and background color. Note that this does nothing with the fillcolor and some objects (i.e. filled rectangles) may not change appearance much using this base class implementation and so they override this.

virtual void reverse_video (VEditor *veditor);

select

Select the object (i.e. in response to the user clicking the mouse select button over this object).

virtual void select (VEditor *veditor);

selection_dehighlight

Remove selection highlighting from this object.

virtual void selection_dehighlight (VEditor *veditor);

selection_highlight

Draw attention to a selected object.

virtual void selection_highlight (VEditor *veditor);

set_annotation

Attach the given text ‘string’ to the object in the specified position (useful only for GAnnotatedIcon and GAnnotatedObj at this time).

virtual void set_annotation (VEditor *veditor, char *string, int location);

set_backgroundcolor

set the background color of the object to the given color.

virtual void set_backgroundcolor (G_COLOR c);

set_color

Set the foreground color of this object to the given color.

virtual void set_color (G_COLOR color);

set_fillcolor

set the fill color of the object to the given color.

virtual void set_fillcolor (G_COLOR c);

set_filled

Set whether the object is filled or not.

virtual void set_filled (Boolean flag);

set_font

Set the font of the object.

virtual void set_font (G_FONT fontID);

set_graphics_annotation

Attach the given object ‘obj’ to the object in the specified position (useful only for GAnnotatedIcon and GAnnotatedObj at this time).

virtual void set_graphics_annotation (VEditor *veditor, GObject *obj, int location);

set_image

Assign the given image to the object (useful only for GIcon at this time).

virtual void set_image (VEditor *ed, GImage *image);

set_numberedColor

Used mainly for rectangles with shadow and/or depth to set individual sides to the given color.

virtual void set_numberedColor (G_COLOR color, int num);

set_style

Set the style of display of the object. This style is unique for each object type and is usually something like ‘shadow’ or ‘depth’.

virtual void set_style (int style);

set_style

Set the style of display of the object. This style is unique for each object type and is usually something like ‘shadow’ or ‘depth’.

virtual void set_style (char * style);

set_writemode

Set the write mode of the object (i.e. XOR, REPLACE,...).

virtual void set_writemode (int wmode);

undraw

Undraw the object.

virtual void undraw (VEditor *veditor);

unhide

Make this object visible if hidden.

virtual void unhide (VEditor *veditor);

GOrthoPline

SYNOPSIS:

GOrthoPline ()

GOrthoPline (G_WPOINT *pts, int npoints)

DESCRIPTION

A polyline (multiple line) display object which constrains the horizontal lines to stay horizontal and vertical lines to stay vertical. The programmer may also supply an optional grid to which the endpoints of the lines are forced to coincide. The lines may appear in any size, color, line style, or width. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

pts	The coordinates of the start point and successive endpoints of the lines.
npoints	The number of points in the 'pts' point array. (The number of points is equal to the number of lines plus one (+1)).

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

The optional grid is not supported at this time.

SEE ALSO:

GOrthoPline

GObject, GLine, GPline, GPolygon

SPECIAL METHODS:

get_autoOrthoMaintenance

Return whether or not orthogonality is maintained.

Boolean get_autoOrthoMaintenance ()

get_endPtOnGrid

Return whether or not the end point is forced to coincide with a grid point.

Boolean get_endPtOnGrid ()

get_startPtOnGrid

Return whether or not the start point is forced to coincide with a grid point.

Boolean get_startPtOnGrid ()

set_autoOrthoMaintenance

Specify whether or not orthogonality is maintained.

void set_autoOrthoMaintenance (Boolean flag)

set_endPtOnGrid

Specify whether or not the end point is forced to coincide with a grid point.

void set_endPtOnGrid (Boolean flag)

set_grid

Assign a layout grid for the endpoints of the lines. Any point = -1 is ignored.

void set_grid (G_WPOINT *ptgrid, int ngrids);

set_startPtOnGrid

Specify whether or not the start point is forced to coincide with a grid point.

void set_startPtOnGrid (Boolean flag)

GPLine

SYNOPSIS:

GPLine ()

GPline (G_WPOINT *pts, int npoints)

DESCRIPTION

A polyline (multiple line) display object. The lines may appear in any size, color, line style, or width. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

pts	The coordinates of the start point and successive endpoints of the lines.
npoints	The number of points in the 'pts' point array. (The number of points is equal to the number of lines plus one (+1)).

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GLine, GOrthoPline, GPolygon

SPECIAL METHODS:

get_numPoints

Return the current number of points in the polyline.

int get_numPoints ()

get_points

Return the current point array for the polyline.

G_WPOINT*get_points ()

set_points

Assign a new point array for the polyline.

void set_points (G_WPOINT *pts, int npts);

GPolygon

SYNOPSIS:

GPolygon ()

***GPolygon** (G_WCOORD x1, G_WCOORD y1, G_WCOORD x2, G_WCOORD y2, G_WCOORD x3, G_WCOORD y3)*

DESCRIPTION

A polygon (closed multiple-line) display object. The lines may appear in any size, color, line style, width, or filled. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x1, y1, x2, y2, x3, y3 — The coordinates of the endpoints of the first 2 lines of the polygon.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GLine, GPline, GOrthoPline

SPECIAL METHODS:

add_point

Add a point to the polygon.

void add_point (G_WCOORD x, G_WCOORD y);

get_num_points

Return number of points in the polygon.

int get_num_points ();

set_position

Set center of polygon to given position.

virtual void set_position (VEditor *veditor, G_WCOORD x, G_WCOORD y);

set_position

Set line endpoints to the indicated value:

virtual void set_position (VEditor *veditor, int posnum, G_WCOORD x, G_WCOORD y);

GRect

SYNOPSIS:

GRect ()

***GRect** (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)*

DESCRIPTION

A rectangle display object. The rectangle may appear in any size, color, line style, or filled. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

xmin, ymin, xmax, ymax	The coordinates of the lower left and upper right corners of the rectangle in world coordinates.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GShadowRect, GStyledRect

SPECIAL METHODS:

GShadowRect

SYNOPSIS:

GShadowRect ()

***GShadowRect** (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)*

DESCRIPTION

A rectangle display object. The rectangle may appear in any size, color, line style, or filled. In general, all the methods specified by the GObject base class have been implemented for this class. The rectangle has a shadow which appears to the lower right of the rectangle. The width (and width behavior using the GWidth object) and color of the shadow may be specified.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x***min, ymin, xmax, ymax*** The coordinates of the lower left and upper right corners of the rectangle.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GRect, GStyledRect, GWidth

SPECIAL METHODS:

get_shadowcolor

Return the shadow color.

G_COLOR get_shadowcolor ()

get_shadowwidth

Return the shadow width object.

GWidth *get_shadowwidth ()

set_shadowcolor

Assign the shadow color.

void set_shadowcolor (G_COLOR color)

set_shadowwidth

Assign the width of the shadow in pixels.

void set_shadowwidth (G_DWIDTH width)

GStyledRect

SYNOPSIS:

GStyledRect ()

***GStyledRect** (G_WCOORD xmin, G_WCOORD ymin, G_WCOORD xmax, G_WCOORD ymax)*

DESCRIPTION

A rectangle display object. The rectangle may appear in any size, color, line style, or filled. In general, all the methods specified by the GObject base class have been implemented for this class. The rectangle has a styled border which lends a number of beveled looks to the rectangle similar to the popular looks of OSF/Motif and NeXT. The particular style, width (and width behavior using the GWidth object) and color of the style may be specified. Four colors are used to render the stylized border. These colors are referred to as ‘black’, ‘dark’, ‘light’ and ‘white’, going from darkest to lightest. These four colors can be assigned any color.

There are currently 3 styles. Refer to groot.h for the complete set.

RECTANGLE_STYLE_TYPE_0
: Simple, 2 color bevel.

RECTANGLE_STYLE_TYPE_1
: NeXT-like.

RECTANGLE_STYLE_TYPE_2
: Motif-like.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

xmin, ymin, xmax, ymax — The coordinates of the lower left and upper right corners of the rectangle.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GObject, GRect, GShadowRect, GWidth

SPECIAL METHODS:

get_colorblack

Return the blackest color.

G_COLOR get_colorblack ()

get_colordark

Return the dark color.

G_COLOR get_colordark ()

get_colorlight

Return the light color.

G_COLOR get_colorlight ()

get_colorwhite

Return the white color.

G_COLOR get_colorwhite ()

get_indented

Return whether the rectangle appears to be indented or raised.

Boolean get_indented ()

GStyledRect

get_style

Return the current style of the rectangle.

int ***get_style ()***

get_stylewidth

Return the width object of the style.

GWidth ****get_stylewidth ()***

set_colorblack

Set the blackest color.

void ***set_colorblack (G_COLOR cb)***

set_colordark

Set the dark color.

void ***set_colordark (G_COLOR cd)***

set_colorlight

Set the light color.

void ***set_colorlight (G_COLOR cl)***

set_colorwhite

Set the white color.

void ***set_colorwhite (G_COLOR cw)***

set_indented

Set whether the rectangle appears to be indented or raised.

void ***set_indented (Boolean flag)***

set_numberedColor

Set the particular stylized border color to the given color. The colors are numbered: 0 and 1 are black, 2 is dark, 3 is light, 4 is white

virtual void set_numberedColor (G_COLOR col, int num);

set_style

Assign a particular style to the rectangle.

virtual void set_style (int s)

set_style

Assign a particular style to the rectangle.

virtual void set_style (char * style)

set_stylewidth

Set the width if the stylized border in pixels.

void set_stylewidth (G_DWIDTH w)

GText

SYNOPSIS:

GText ()

***GText** (G_WCOORD x, G_WCOORD y, char *string)*

DESCRIPTION

A text display object. The text may appear in any supported font or color. In general, all the methods specified by the GObject base class have been implemented for this class.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x, y,	The coordinates of the center of the text string.
string	The text to be displayed.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

The methods selection_highlight and deselection_highlight are not implemented as they were found to be more annoying than useful.

SEE ALSO:

GObject, GTextFixedSizeFont

SPECIAL METHODS:

get_font

Return the current font.

virtual G_FONTget_font ()

get_text

Return the text string this displays.

char *get_text ()

set_font

Assign a font to the text. Can be any font or charset name.

virtual void set_font (G_FONT textfont)

set_text

Specify the text string to display.

virtual void set_text (char *string)

GTextFixedSizeFont

SYNOPSIS:

GTextFixedSizeFont ()

GTextFixedSizeFont (G_WCOORD x, G_WCOORD y, char *string)

DESCRIPTION

A text display object. The text may appear in any supported font or color. In general, all of the methods specified by the GObject base class have been implemented for this class. This class is specifically designed for bitmap fonts that come in only one size and so do not change size dynamically during zooms (changes in magnification).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

x, y,	The coordinates of the center of the text string.
string	The text to be displayed.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

The methods selection_highlight and deselection_highlight are not implemented as they were found to be more annoying than useful.

SEE ALSO:

GObject, GText

SPECIAL METHODS:

get_invalid_width_height

Return whether the width and height of text font will be recalculated next time this is drawn or extrema inquired.

Boolean get_invalid_width_height ()

get_invalidate_all_width_height

Return whether width and height of text font is recalculated each time this is drawn or extrema inquired.

static Boolean get_invalidate_all_width_height ()

get_threshold_scales

Return magnification below which text is not drawn.

static void get_threshold_scales (double *xscale, double *yscale)

invalidate_width_height

Recalculate width and height of text font next time this is drawn or extrema inquired.

void invalidate_width_height ()

set_font

Assign a font to the text. Can be any font or charset name.

virtual void set_font (G_FONT textfont)

set_invalidate_all_width_height

Calculate width and height of text font each time this is drawn or extrema inquired.

static void set_invalidate_all_width_height (Boolean flag)

set_text

Specify the text string to display.

virtual void set_text (char *string)

set_threshold_scales

GTextFixedSizeFont

Set magnification below which text is not drawn.

static void set_threshold_scales (double xscale, double yscale)

GWidth

SYNOPSIS:

GWidth ()

DESCRIPTION

GWidth objects describe the size and behavior of a geometric width. The size can be specified in device or world coordinates. If device coordinates are specified, the width does not appear to change under magnification (i.e. different zoom levels) unless the width exceeds the maximum world width (if specified using the set_max_world_width() method). If the width is specified in world coordinates then the width appears to change naturally under various magnifications unless the width exceeds the maximum device width (if specified with the set_max_device_width() method).

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GAttributes

SPECIAL METHODS:

copy_to

Copy this width and its specifications to the given ‘other’ width.

void copy_to (GWidth *other)

get_device_width

Return the width in device (pixel) coordinates given the current world/device viewport transformations in the given editor.

G_DWIDTHget_device_width (VEditor *veditor);

get_specified_device_width

Returns the width in device coordinate space.

G_DWIDTHget_specified_device_width ()

get_specified_world_width

Returns the width in world coordinate space.

G_WWIDTHget_specified_world_width ()

get_world_width

Return the width in world coordinates given the current world/device viewport transformations in the given editor.

G_WWIDTHget_world_width (VEditor *veditor);

is_equal_to

Returns whether this and the given width are equal.

Boolean is_equal_to (GWidth *other)

set_device_width

Specifies that the width is to be a constant width in device coordinates (therefore the world width may change to enforce this constancy). Also specifies what the device width is to be.

void set_device_width (G_DWIDTH dw)

set_device_width

Specifies that the width is to be a constant width in device coordinates (therefore the world width may change to enforce this constancy). Also specifies what the device width is to be using the specified world

width and transforming this width into device coordinates using the given editor's current world to device transformations.

void set_device_width (VEditor *veditor, G_WWIDTH ww);

set_max_device_width

Specifies what the maximum allowed device width is to be when the device width is being updated to maintain a prespecified constant world width.

void set_max_device_width (G_DWIDTH dw)

set_max_world_width

Specifies what the maximum allowed world width is to be when the world width is being updated to maintain a prespecified constant device width.

void set_max_world_width (G_WWIDTH ww)

set_world_width

Specifies that the width is to be a constant width in world coordinates (therefore the device width may change to enforce this constancy). Also specifies what the world width is to be.

void set_world_width (G_WWIDTH ww)

set_world_width

Specifies that the width is to be a constant width in world coordinates (therefore the device width may change to enforce this constancy). Also specifies what the world width is to be using the specified device width and transforming this width into world coordinates using the given editor's current device to world transformations.

void set_world_width (VEditor *veditor, G_DWIDTH dw);

width_specified_in_device_coordinates

Returns whether or not the width has been specified to be a constant device width.

Boolean width_specified_in_device_coordinates ()

ObjWithClassType

SYNOPSIS:

ObjWithClassType ***(char *name, ClassType *parent, int value = -1)***

DESCRIPTION

Base class naming functionality for all graphics display primitives. This class manages the storage and read/write access to the name each type/class/kind of graphics display primitive object has associated with it.

Each class that uses this class has these data and methods:

```
private:

        static  ClassType       *type;

protected:

        static  ClassType       *get_static_type ()
                                        {
                                        return type;
                                        }

        virtual ClassType       *get_type ()
                                        {
                                        return(type);
                                        }

ClassType *GMsgHandler::type = new ClassType (
                                MSGHANDLER_TYPE_NAME,
                                NULL,
                                MSGHANDLER_TYPE_VALUE);

ClassType *GMsgManager::type = new ClassType (
                                MSGMANAGER_TYPE_NAME,
                                GMsgHandler::get_static_type (),
                                MSGMANAGER_TYPE_VALUE);
```

PARAMETERS (Required):

name	Name of the class.
parent	ClassType of the parent of the class.
value	Value to be associated with this class, used for faster

classType searches.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Other base classes:

GGeometry, GAttributes, GBehavior

METHODS:

get_type

Returns the class-typing object for this class.

virtual ClassType *get_type ()

is

Returns whether this class is of the given name.

Boolean is (char *name)

is

Returns whether this class is of the given type.

Boolean is (ClassType *t)

is

Returns whether this class is of the given type value.

Boolean is (int value)

isa

Returns whether this class, or one of this classes base classes, is of the given name.

Boolean isa (char *name)

isa

Returns whether this class, or one of this classes base classes, is of the given type.

Boolean isa (ClassType *t)

isa

Returns whether this class, or one of this classes base classes, is of the given type value.

Boolean isa (int value)

CHAPTER 6

Advanced Topics: Interface to the Low-Level Windowing/Graphics Standards

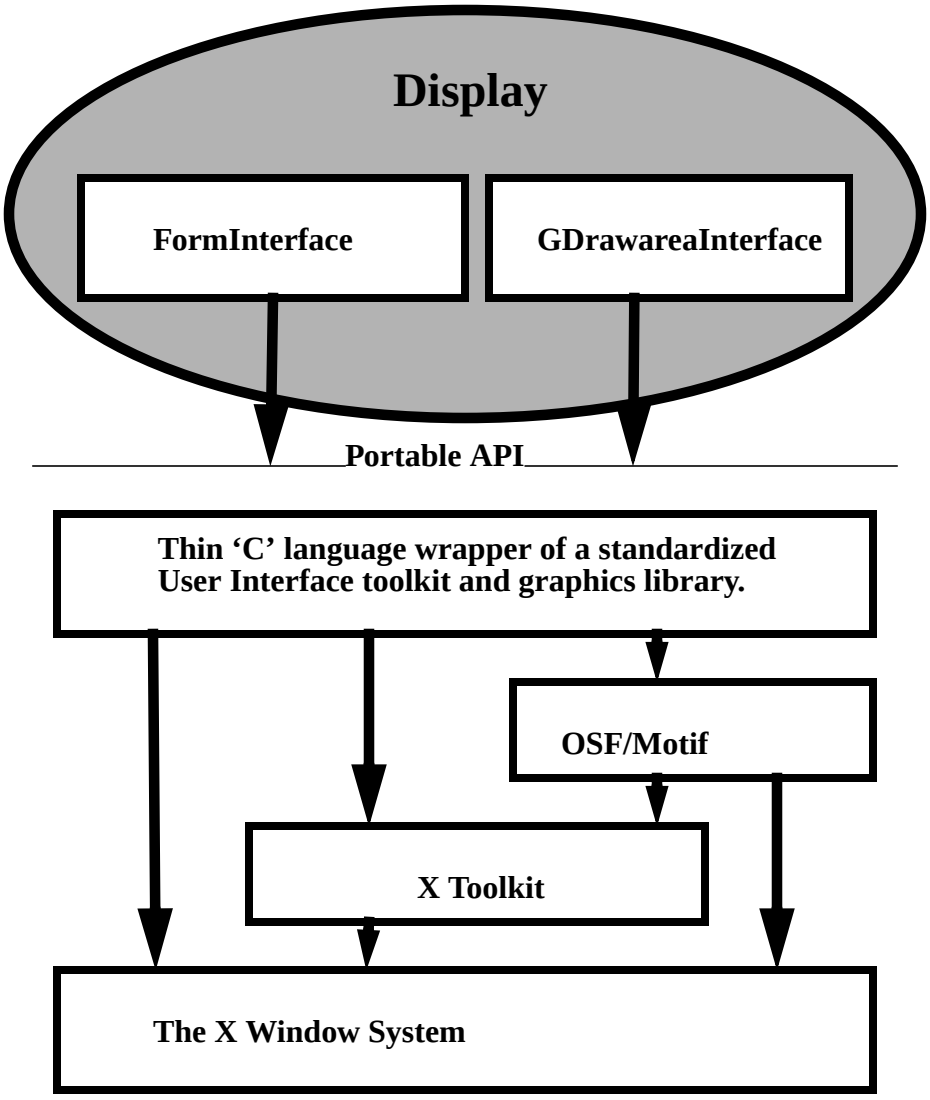

FIGURE 4 The Low-Level Layering of the Editor Object System.

Display

SYNOPSIS:

Create a display object passing down any command line options to the low-level window system.

Display (int argc, char **argv);

Create a display object, using the given low-level window system object to reuse an already existing display connection.

Display (char *widget, char *appContext);

DESCRIPTION

Creates an display object which provides the interface to the low-level window system.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

argc	The number of arguments present in the argv parameter.
argv	The command-like arguments that the low-level window system may be able to interpret.
widget	An existing object in the low-level window system. This is used when this display object is to coexist with other previously existing interfaces to the low-level window system in the current overall application.
appContext	Provides additional information to the low-level window system about the interface used by the rest of the application.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

Display

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

FormInterface, GDrawareaInterface

METHODS:

get_form_interface

Returns the address of a C++ wrapper around the lowest level interface to the low-level user interface toolkit system.

FormInterface *get_form_interface ()

get_graphics_interface

Returns the lowest level interface to the low-level window/graphics system in the form a a ‘C’ language function pointer table.

struct graphics_interface *get_graphics_interface ()

get_interface

Returns the address of a minimal C++ wrapper around the lowest level interface to the low-level window/graphics system. Note this is a interface without a corresponding window, so many functions are not available. It is recommended that the GDrawareaInterface be accessed through the VEditor object which does indeed have an associated window.

GDrawareaInterface *get_interface ()

FormInterface

SYNOPSIS:

Create an interface to the low-level user interface toolkit.

FormInterface (struct graphics_interface *graphics);

DESCRIPTION

Creates an interface to the low-level user interface toolkit. This is not usually instantiated directly but as a by-product of creating a Display object.

Note that in this context containers are objects into which widgets (i.e. labels, text fields, ...) are created. These widgets are then automatically arranged in various aesthetic positions. These arrangements can be varied by specifying various attributes of the container. Widgets are removed either by deletion or unmanaging them. This is the OSF/Motif paradigm which is also used here. Note that a container is sometimes called a buttonarea here.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

graphics	The address of the 'C' function pointer table interface to the low-level window/graphics system.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

Display, GDrawareaInterface

FormInterface

METHODS:

add_close_callback

Specifies that the given callback routine is to be called whenever the given window is about to be closed (i.e. exited, usually by the user selecting a close window option on a window manager menu).

```
void        add_close_callback (
                char *window,
                void *callback_obj,
                PUSHBUTTONFNPTR callback,
                long userdata
                );
```

add_event_handler

Specifies that he given callback routine is to be called whenever the given event occurs above the given widget. The event types supported are:

L_BUTTON_START_DRAG
L_BUTTON_DRAG
WINDOW_ENTER
WINDOW_EXIT
RESIZE
ASCII_EVENT
L_BUTTON_DOWN
L_BUTTON_CLICK
M_BUTTON_DOWN
M_BUTTON_CLICK
R_BUTTON_DOWN
R_BUTTON_CLICK
L_BUTTON_UP
M_BUTTON_UP
R_BUTTON_UP
M_BUTTON_DRAG
M_BUTTON_START_DRAG
R_BUTTON_DRAG
R_BUTTON_START_DRAG

```
void        add_event_handler (
                void *callback_obj,
                char *widget,
                int eventtype,
                EVENTFNPTR callback,
                long userdata
                );
```

add_timer

Starts a periodic invocation of the given callback every ‘interval’ milliseconds. A handle is returned which can be used to end the callbacks.

```
char *      add_timer (void *callback_obj,
                PUSHBUTTONFNPTR callback,
                long interval,
                long userdata);
```

attach_layoutChildToLayoutParent

Arranges the previously created child of the previously created parent in the specified orientation. This routine may then attach/align the child to the given previousChild and/or the parent. The lastOne parameter specifies whether there will be any more children to arrange in this parent.

```
void        attach_layoutChildToLayoutParent (
                char *parent,
                char *child,
                char *previousChild,
                int orientation,
                Boolean lastOne);
```

clear_drawarea_background

Fill drawarea with it’s current background color.

```
void        clear_drawarea_background (char *drawarea);
```

clear_scrolledlist

Remove all text items from the given list.

```
void        clear_scrolledlist (char *list);
```

create_buttonarea

Create a horizontally or vertically filling container. Direction is one of FHORIZONTAL or FVERTICAL.

```
char    *   create_buttonarea (char *container, Boolean direction);
```

create_drawarea

Create an area within the given window in which graphics may be drawn. The area is created at the given size and a handle of the drawarea is returned. The given callback ptr is invoked whenever events occur within the drawarea.

```
char        *create_drawarea (void *callback_object,
                char *window,
                EVENTFNPTR callback,
                G_DCOORD xmin, G_DCOORD ymin,
                G_DCOORD xmax, G_DCOORD ymax, long userdata);
```

create_editor

Create an area within the given window in which graphics may be drawn. The area is created at the given size and a handle of the drawarea is returned. The given callback ptr is invoked whenever events occur within the drawarea. The area has scrollbars along bottom and/or right sides if the scrollbars flag is set to:

BOTTOM_SIDE_SCROLLBAR and/or
RIGHT_SIDE_SCROLLBAR

```
char        *create_editor (void *callback_object,
                char *window,
                EVENTFNPTR drawarea_callback,
                SCROLLBAR_EVENTFNPTR scrollbar_callback,
                int scrollbars_flag,
                G_DCOORD xmin, G_DCOORD ymin,
                G_DCOORD xmax, G_DCOORD ymax, long userdata);
```

create_formarea

Create a horizontally or vertically oriented container, whose handle is returned, inside the given parent container, below which is a row containing the possible four standard buttons of OSF/Motif. Each buttondata structure allows the customization of a button; its name, sensitivity, and callback method in the given callback object.

```
char *      create_formarea (
                char *parent,
                void *callback_object,
                struct buttondata *ok_button,
                struct buttondata *apply_button,
                struct buttondata *cancel_button,
                struct buttondata *help_button,
                Boolean horizontal_form_area);
```

create_formwindow

Create a standard window with a row containing the possible four standard buttons of OSF/Motif. Each buttondata structure allows the customization of a button; its name, sensitivity, and callback method in the given callback object. The window title is set to the given ‘bordername’ parameter and the window name is set to

the given ‘widgetname’ parameter. The area in-between the top of the window and the 4 possible buttons on the bottom is filled with a horizontally or vertically oriented container, whose handle is returned.

```
char *      create_formwindow (
                char *bordername,
                char *widgetname,
                void *callback_object,
                struct buttondata *ok_button,
                struct buttondata *apply_button,
                struct buttondata *cancel_button,
                struct buttondata *help_button,
                Boolean horizontal_form_area);
```

create_frame

Create a rectangular decoration inside the given container of the given width and type style. Returns a handle to the frame within which other widgets/containers may be created. The width is specified in pixels and the type may be one of:

FRAME_SHADOW_IN
FRAME_SHADOW_OUT
FRAME_SHADOW_ETCHED_IN
FRAME_SHADOW_ETCHED_OUT

```
char *      create_frame (char *container, int shadow_width, int type =
                FRAME_SHADOW_IN);
```

create_label

Create a textual label in the given container.

```
char        *create_label (char *container, char *name);
```

create_lowLevelLayoutWidget

Creates a layout widget as a child of the given container.

```
char        *create_lowLevelLayoutWidget (char *container);
```

create_menubar

Create a menubar for the given window with pulldown menus and options specified by the given ‘pulldown’ structure. Submenus, accelerators, mnemonics, sensitivity and other options are available. All the users selections in the menubar go to the given caller object to the callback as specified in the pulldown structure. (see xmdraw.h).

```
char *      create_menubar (char *window, void *callerobj, struct pulldown **pds);
```

create_option_menu

Create an option menu for the given window (or container) with options specified by the given 'pulldown' structure. Submenus, accelerators, mnemonics, sensitivity and other options are available. All the users selections in the option menu go to the given callback object and callback. The 'default_option_index' parameter specifies which option in the option menu is to initially appear selected.

```
char        *create_option_menu (
                void *callback_obj,
                char *window,
                PULLDOWNFNPTR callback,
                struct pulldown *options,
                int default_option_index,
                long userdata);
```

create_popup_menu

Create a popup menu for the given window (or container) with options specified by the given 'pulldown' structure. Submenus, accelerators, mnemonics, sensitivity and other options are available. All the users selections in the popup menu go to the given callback object and callback.

```
char        *create_popup_menu (
                void *callback_obj,
                char *window,
                PULLDOWNFNPTR callback,
                struct pulldown *options,
                long userdata);
```

create_pulldown_menu

Create a pulldown menu for the given window (or container) with options specified by the given 'pulldown' structure. Submenus, accelerators, mnemonics, sensitivity and other options are available. All the users selections in the pulldown menu go to the given callback object and callback. The 'default_option_index' parameter specifies which option in the option menu is to initially appear selected.

```
char        *create_pulldown_menu (
                void *callback_obj,
                char *window,
                PULLDOWNFNPTR callback,
                struct pulldown *options,
                int default_option_index,
                long userdata);
```

create_pushbutton

Create a pushbutton, which when selected by the user, invokes the specified callback.

```
char        *create_pushbutton (
                void *callback_obj,
                char *container,
                char *label,
                PUSHBUTTONFNPTR callback,
                long userinfo);
```

create_radioarea

Create a horizontally or vertically filling container that is to be filled with toggle buttons which it is desired to work in a radio button manner.

```
char    *   create_radioarea (char *container, Boolean direction);
```

create_radiobuttons

Create a area filled with a group of radio buttons. The number and labels of the radio buttons are given and the individual radio button handles are returned in the given 'widgets' array. The buttons are placed, if possible, in an array such that there are the given number of columns. The 'default_index' parameter specifies which radio button is to be 'on' first.

```
void        create_radiobuttons (
                void *callback_obj,
                char *container,
                char *labels[],
                RADIOBUTTONFNPTR callback,
                char *widgets[],
                int number,
                int default_index,
                int numcolumns,
                long userinfo);
```

create_scrollbox

Create a scrolling area in the given parent container of the given size.

```
char        *create_scrollbox (char *parent,
                int scrollbars,
                int width,
                int height);
```

create_scrolledlist

Create a scrolled list of the given text strings in the given window/container. List configuration options are:

SCRLIST_SELECT_WHEN_BROWSED or,
SCRLIST_PM_DRAG_AND_BROWSE or,
SCRLIST_SINGLE_SELECTION (the default).

The given callback in the given callback object is called whenever an item is selected or 'double clicked'. The seltype parameter to the callback is then LIST_ITEM_SELECTED or LIST_ITEM_DBLSELECTED.

```
char        *create_scrolledlist (
                void *callback_obj,
                char *window,
                SCROLLLISTFNPTR callback,
                int numrows_displayed_in_list,
                char ** strings_in_list,
                long userinfo,
                int configmask = SCRLIST_SINGLE_SELECTION);
```

create_scrolledtext

Create a scrolling text editor with the given number of displayed rows and columns. Sets the maximum number of characters allowed in the editor at any one time (from the bufsize parameter) and sets the contents of the editor to the 'value' parameter.

```
char        *create_scrolledtext (char *parent,
                void *callback_object,
                TEXTFIELDFNPTR callback,
                int numrows,
                int numcolumns,
                int bufsize,
                char *value,
                long userinfo);
```

create_selectionlist

Create a standard selection list within the given container. The list is initialized to contain the given list of text items. The ok and apply buttons may be overridden by specifying their names here. The types of selection boxes supported are:

SELBOX_NO_APPLY
SELBOX_STANDARD (the default)
SELBOX_COMMAND
SELBOX_TEXTENTRY

The list interaction configurations supported are:

SCRLIST_SELECT_WHEN_BROWSED
SCRLIST_PM_DRAG_AND_BROWSE
SCRLIST_SINGLE_SELECTION (the default).

```
char *      create_selectionlist (
                void *callback_object,
                char *container,
                char **list,
                char *ok_button_name,
                char *apply_button_name,
                SCROLLLISTFNPTR callback,
                long userdata,
                int config = SCRLIST_SINGLE_SELECTION,
                int type = SELBOX_STANDARD);
```

create_separator

Create a separator (horizontal line) in the given container.

```
char        *create_separator (char *container);
```

create_stdbuttons

Create a horizontal area containing the possible four standard buttons of OSF/Motif. Each buttondata structure allows the customization of a button; its name, sensitivity, and callback method in the given callback object.

```
void        create_stdbuttons (char *parent,
                void *callback_object,
                struct buttondata *ok_button,
                struct buttondata *apply_button,
                struct buttondata *cancel_button,
                struct buttondata *help_button);
```

create_textfield

Create a text entry field within the given parent container. If the given ‘name’ parameter is not NULL then a label is also created to the left of the text field set to the given name. The text field is populated with the given text value and the given callback is called whenever the user presses <cr> (the enter key).

```
char *      create_textfield (
                void *callback_object,
                char *parent,
                char *name,
                char *value,
                TEXTFIELDFNPTR callback,
                long userinfo);
```

FormInterface

create_togglebutton

Create a toggle button, which when toggled by the user, invokes the specified callback.

```
char        *create_togglebutton (
                void *callback_obj,
                char *container,
                char *label,
                TOGGLEBUTTONFNPTR callback,
                long userinfo);
```

delete_widget

Delete the given widget. The widget can be a window or any other low level handle.

```
void        delete_widget (char *widget);
```

deselect_all_items

Deselects all list items in a scrolled list.

```
void        deselect_all_items (char *widget);
```

get_button_state

Return the current state of the given toggle button.

```
Boolean     get_button_state (char *togglebutton);
```

get_pixmap

Return a handle to a pixmap created from the given filename. The widget is used for reference purposes only.

```
char *      get_pixmap (char *widget, char *filename);
```

get_textfield

Return the contents of the given text field.

```
char        *get_textfield (char *tf);
```

get_textstring_size

Return the size, in pixels, of the given string of text. If the given fontname is not specified, then the current font will be used to do the calculations.

void get_textstring_size (char *textstring, int *width, int *height, char *font);

get_widget_size

Return the size of a widget, in pixels.

void get_widget_size (char *container, int *xmin, int *ymin, int *xmax, int *ymax);

manage

Manage (i.e. position and display) or unManage (i.e. hide and re-position everything nearby) the given widget.

void manage (char *widget, int flag);

modify_frame

Modify an existing frame's appearance. See create_frame ().

void modify_frame (char *frame, int shadow_width, int type);

popdown

Close the given window without deleting it.

void popdown (char *window);

popup

Popup and/or raise to the front the given window.

void popup (char *window, int position = WM_DEFAULT_POSITION);

post_message_form

Post a standard message window (dialog box) which displays the given message. The kinds of standard message dialogs supported are:

ERRORMSG
INFOMSG
JUSTAMSG
QUERYMSG
WARNINGMSG
WORKINGMSG

This routine does not return until the user closes the dialog box. Returns either FORM_OKD or FORM_CANCELED.

```
int         post_message_form (
                char *message,
                int kind,
                void *callback_obj,
                VOIDFNPTR help_callback,
                long userdata);
```

post_textentry_form

Post a dialog box which displays the given message and prompts the user to enter textual input. This routine does not return until the user closes the dialog box. Returns either FORM_OKD or FORM_CANCELED.

```
int post_textentry_form (
                char *message,
                char *buffer,
                int buflen,
                void *callback_obj,
                VOIDFNPTR help_callback,
                long userdata);
```

remove_timer

Removes the repetitive invocation of the callback. The given timer is a handle previously returned from the add_timer() method.

```
void        remove_timer (char *timer);
```

select_listitem

Display as selected the given item in the given list.

```
void        select_listitem (char *list, char *item);
```

set_background_color

Set the background color of a widget.

```
void        set_background_color (char *window, char *colorname);
```

set_bordername

Set the title text of a window.

```
void        set_bordername (char *window, char *name);
```

set_button_state

Set the given toggle button state to on or off.

void set_button_state (char *togglebutton, Boolean flag);

set_buttonarea_alignment

Specifies placement constraints/hints for contents of the given container. Possible alignments supported are:

FALIGNMENT_NONE
FALIGNMENT_LEFT
FALIGNMENT_RIGHT
FALIGNMENT_CENTER

void set_buttonarea_alignment (char *container, int alignment);

set_buttonarea_orientation_and_numcolumns

Specifies placement constraints/hints for contents of the given container.

```
void set_buttonarea_orientation_and_numcolumns (
        char *container,
        Boolean horizontal,
        int numcol);
```

set_color

Set the foreground color of a widget.

void set_color (char *window, char *colorname);

set_current_menu_item

Set the current displayed option in an option menu to the given item. If no option with the same name as the item exists, then nothing is done.

void set_current_menu_item (char *menu, char *item);

set_label

Replace the text in the given label with the new text.

void set_label (char *item, char *newlabel);

set_label_margins

Set the margins, in pixels, for a widget.

void set_label_margins (char *label, int width, int height);

set_pixmap

Assign the given pixmap to the given widget. Usually used on label widgets.

void set_pixmap (char *widget, char *pixmap);

set_sensitivity

Set the sensitivity of the given widget to the user.

void set_sensitivity (char *widget, Boolean flag);

set_size

Set the width and height of a widget.

void set_size (char *window, G_DCOORD width, G_DCOORD height);

set_textfield

Replace the contents of the given text field with the new contents.

void set_textfield (char *tf, char *newcontents);

set_widget_size

Assign a size to a widget.

void set_widget_size (char *w, int xmin, int ymin, int xmax, int ymax);

update_items_in_scrolledlist

Replace the specified text items in the given list with the specified new items.

```
void update_items_in_scrolledlist (
        char *list,
        char ** strings_to_delete, // NULL terminated list.
        char ** strings_to_add); // NULL terminated list.
```

GLayoutContainer

SYNOPSIS:

***GLayoutContainer** (Display *display, char *container, int orientation);*

DESCRIPTION:

Creates a container in which subsequent GLayoutContainers are created and arranged in horizontal or vertical rows/columns. This arrangement is preserved through resizes of the original container.

PARAMETERS (Required):

display The display object for this window/ graphics system.

container The container object specific to the underlying window system.

orientation The manner in which subsequent 'child' GLayoutContainers are to be arranged within this GLayoutContainer. Supported orientations at this time are:

FVERTICAL
(Children are placed top to bottom).

FHORIZONTAL
(Children are placed left to right).

PARAMETERS (Optional):

None.

COMPONENT NAME:

None

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

If the given container is a XmRowColumn (i.e. buttonarea) of the X Window System, resize events will probably not be passed down to this GLayoutContainer.

SEE ALSO:

FormInterface

METHODS:

create_andAppendChildLayoutContainer

Creates a GLayoutContainer, whose address is returned and assigns it the given orientation for ITS children.

GLayoutContainer*create_andAppendChildLayoutContainer (int orientation);

doLayout

Perform the actual arrangement of this class's children, and their children, etc.

void doLayout ();

get_lowLevelLayoutWidget

Returns the handle of a low-level window system widget into which other widgets can be inserted.

char *get_lowLevelLayoutWidget ()

CHAPTER 7

Advanced Topics: The Messaging System

Messaging in supported in this system in such a way as to conform to the following assumptions:

- Messaging systems are more flexible and easier to program (and make it easier to support end-user programming).
- Messaging systems are slow.

Therefore:

Operations that occur many times a second are written in the ‘C’ or ‘C++’ languages such as the routine responsible for the drawing of 100,000 lines in an editor.

Operations that occur relatively infrequently (with respect to machine speeds) are written as message handlers such as a user zooming in on something in the editor.

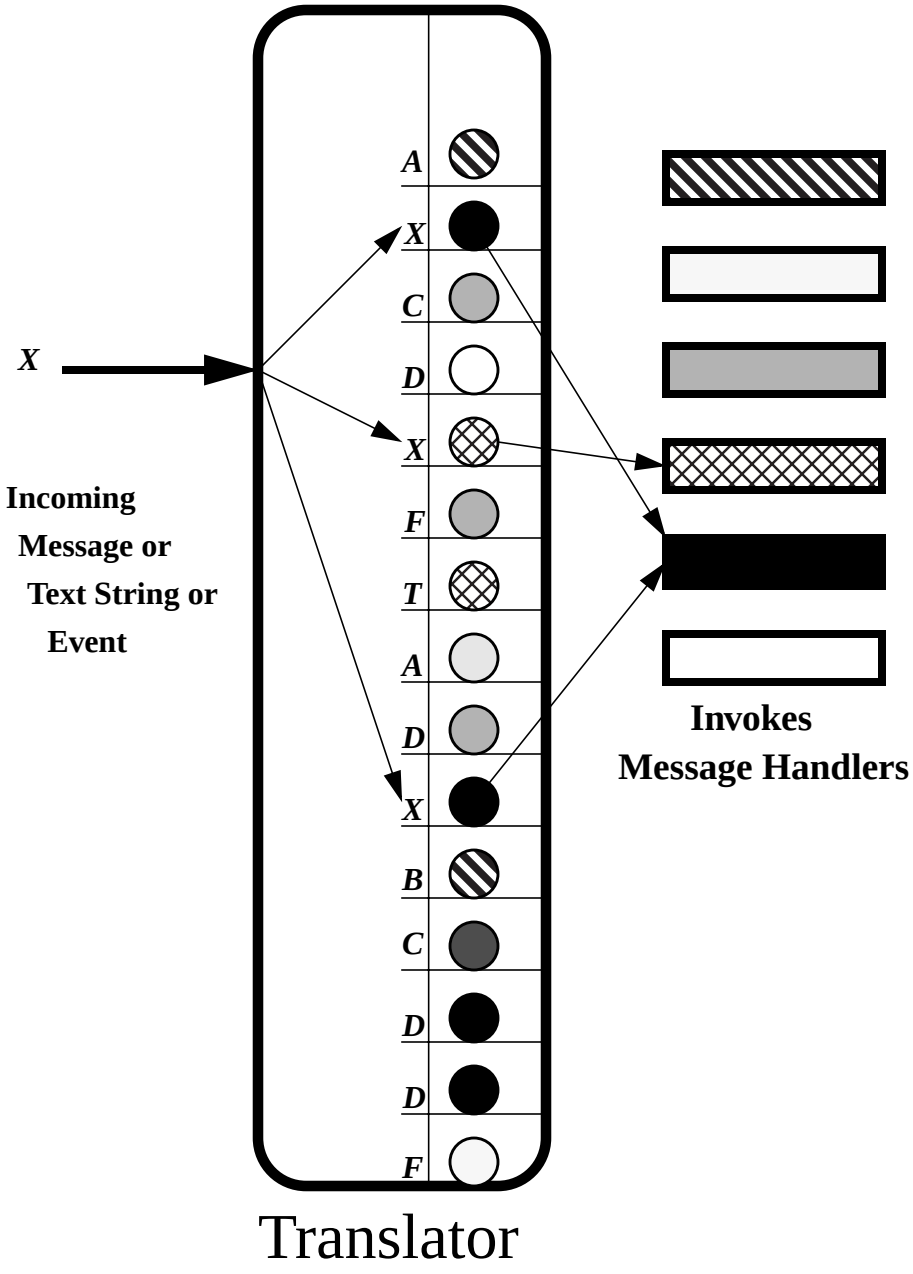

FIGURE 5 The Translator broadcasts messages that have been assigned to the given message, text string or event.

GMessage

SYNOPSIS:

GMessage (char *verb = NULL, char *noun = NULL, GMsgManager *dest = NULL);

DESCRIPTION

Creates an message object. The message object represents an action (verb) and parameters for the action. Messages are typically sent a GMsgManager (like a VEditor) and responded to by one or more GMsgHandlers that were previously assigned to the GMsgManager. The verb is also typically the name of the GMsgHandler that will execute the message. Translations can be assigned to GMsgManagers so that when programmer specified events, text or messages are received, prerecorded messages are then broadcast or sent to specified GMsgManagers.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

verb	The name of the action/GMsgHandler that the message represents.
noun	The first parameter of the action.
dest	The GMsgManager that this message is to be sent to.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

GMessage

SEE ALSO:

GMsgManager, GMsgHandler, GTranslator

METHODS:

add_parm

Add a parameter value to this message.

void add_parm (char *parmname);

copy

Make and return a copy of this message.

GMessage*copy ();

disablePercentReplacement

Disable the substitution of the values of any variables into their assigned parameters, if any. Available in VisualADE only.

Boolean disablePercentReplacement ()

disablePercentReplacement

Enable/Disable the substitution of the values of any variables into their assigned parameters, if any. Available in VisualADE only.

void disablePercentReplacement (Boolean flag)

get_caller

Return the calling GMsgManager.

GMsgManager*get_caller ()

get_creator

Return the previously specified creator of the message. Useful for debugging.

char * get_creator ()

get_destination

Return the GMsgManager this will be sent to.

GMsgManager*get_destination ()

get_destname

Return the wild card-like search string which is used to determine the destination GMsgManagers to send this message to. Available in VisualADE only.

char *get_destname ()

get_event

Return the event that prompted this message to be generated in a translation table. This event corresponds to a low-level window system event and is usually NULL for messages generated from other causes.

GEvent *get_event ()

get_instance

Return the application object that is to be executed on. Available in VisualADE only.

GObjOrFnInstance *get_instance ()

get_mouse

Return the information about the state of the mouse recorded during the last event received.

GEvent *get_mouse ()

get_msgHandler

Return the GMsgHandler assigned to handle this message. Not used at this time.

GMsgHandler*get_msgHandler ()

get_msgclasstype

Return class type. At this time there are only two class types. One for errors and one for normal messages. These types are: GMESSAGE_CLASS_TYPE and GMESSAGE_ERROR_CLASS_TYPE.

int get_msgclasstype ()

get_numparms

Return the number of parameters this message has.

int get_numparms ()

GMessage

get_parm

Return a parameter value of this message.

char** ***get_parm (int parmnum);

get_searchSpec

Return the wild card-like search string which is used to determine the destination GMsgManagers to send this message to. Available in VisualADE only.

char** ***get_searchSpec ()

get_verb

Return the name of the message (i.e. its verb).

char** ***get_verb ()

get_verb_code

Return the integer code that corresponds to the verb name.

int** **get_verb_code ()

hasPercents

Return whether the searchSpec has any percent signs which indicate a presence of a variable in at least one of the parameters. Available in VisualADE only.

Boolean** **hasPercents ()

is

Return whether this message has the given code. If no code has been assigned to this message then the given name is compared with the message's verb name.

Boolean is (int code, char *name);

is

Return whether this message has the given code and 1st parameter. If no code has been assigned to this message then the given name is compared with the message's verb name.

Boolean is (int code, char *name, char *parm);

is

Return whether this message has the given code and 1st and 2nd parameter. If no code has been assigned to this message then the given name is compared with the message's verb name.

Boolean is (int code, char *name, char *noun, char *parm2);

isAnError

Return whether this is an error message.

Boolean isAnError ()

parm_is

Return whether the i-th (specified by index) parameter is equal to the given name.

Boolean parm_is (int index, char *name);

set_caller

Specify the calling GMsgManager. Some GMsgHandlers may look at this in order to ascertain more contextual information.

void set_caller (GMsgManager *callerobj)

set_creator

Specify the creator of the message. Useful for debugging.

void set_creator (char *n)

set_destination

Assign the GMsgManager this will be sent to. This is often done automatically by setting up translations. See GTranslate.

void set_destination (GMsgManager *destination)

set_destname

Specify a wild card-like search string which is used to determine the destination GMsgManagers to send this message to. Available in VisualADE only.

void set_destname (char *name)

set_event

Specify the event that prompted this message to be generated.

void set_event (GEvent *e);

set_instance

Specify the application object that is to be executed on. Available in VisualADE only.

void set_instance (GObjOrFnInstance *i)

set_isAnError

Specify whether this is an error message. If so, VisualADE will propagate this to the nearest GMsgErrorHandler. Available in VisualADE only.

void set_isAnError (Boolean flag)

set_msgHandler

Assign the GMsgHandler assigned to handle this message. Not used at this time.

void set_msgHandler (GMsgHandler *mh)

set_msgclasstype

Assign class type. See get_msgclasstype () for more information.

void set_msgclasstype (int type)

set_numparms

Assign the number of parameters this message has.

void set_numparms (int nparms)

set_parm

Assign a parameter value to this message.

void set_parm (int parmnum, char *parmname);

set_searchSpec

Specify a wild card-like search string which is used to determine the destination GMsgManagers to send this message to. Available in VisualADE only.

void set_searchSpec (char *i)

set_verb

Assign the name to the message (i.e. its verb).

void set_verb (char *n)

set_verb_code

Assign the integer code that corresponds to the verb name. This is used to reduce the time taken when the message searches through lots of GMsgHandlers looking for one that can handle this messages verb.

void set_verb_code (int code)

GMsgCentral

SYNOPSIS:

GMsgCentral ()

DESCRIPTION:

This is a global object that provides services for messages GMessage and message handlers GMsgHandlers. These services range from debugging and tracing aids, to error logging, to maintaining the list of all possible message handlers.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GMsgManager, GMsgHandler, GTranslator

METHODS:

assign_code_to_msg

Assigns an integer value to the given message based on its name (i.e. its targeted message handler).

void assign_code_to_msg (GMessage *msg)

find_msg_handler

Return the first message handler found with the given name that can be assigned to the given message manager type.

GMsgHandler*find_msg_handler (char *name, char *required_component_type);

find_msg_handler

Return the first message handler found with the given name that is assigned to the given message manager.

GMsgHandler *find_msg_handler (GMsgManager *mm, char *name);

find_msg_manager

Return the first message manager found with the given name.

GMsgManager*find_msg_manager (char *name);

get_code_for_msg

Returns the integer value for the given message based on its name (i.e. its targeted message handler).

int get_code_for_msg (char *name);

get_list_of_handlers

Returns a list of all the message handlers in the system. Unimplemented.

char **get_list_of_handlers (GMsgManager *mm);

get_list_of_managers

Returns the list of all the message managers in the system.

GMsgManagerList*get_list_of_managers ();

get_manager_that_handles_msg

Returns (a list?) of message managers who 1) have subscribed to receive the given message or 2) have an associated message handler capable of processing the given message. Not implemented.

GMsgManager*get_manager_that_handles_msg (GMessage *msg)

get_msg_handlers

Returns the list of all the message handlers in the system.

GMsgHandlerArray *get_msg_handlers ()

print_list_of_handlers

Prints the list of all message handlers assigned to the given message manager.

void print_list_of_handlers (GMsgManager *mm);

print_list_of_handlers

Prints the list of all message handlers in the system.

void print_list_of_handlers ();

print_list_of_managers_and_handlers

Prints the list of all message managers and handlers in the system.

void print_list_of_managers_and_handlers ();

publishMsg

Sends the given message to all message managers that have subscribed (to this message central object) and had indicated that they want to receive any published messages with this name.

Boolean publishMsg (GMessage *msg);

pushOnCatchStack

Appends the given message handler to the list of message handlers that errors are forwarded to.

void pushOnCatchStack (GMsgHandler *mh);

register_handler

Register the given message handler as one of the handlers in the system.

int register_handler (GMsgHandler *mh);

register_manager

Register the given message manager as one of the managers in the system.

void register_manager (GMsgManager *mm);

setCatchStackIndex

Pops off a number of message handlers off the list of message handlers that errors are forwarded to.

void setCatchStackIndex (int index);

subscribeToMsg

Registers the given message manager as indicating the desire to receive all published messages of the given name.

void subscribeToMsg (GMsgManager *mm, char *msgname);

throwError

Using a throw and catch exception handling system, this method forwards the given message containing the error to a list of registered error handlers until one is found that can deal with it or printing out a textual description of the error to the spawning window. This is used heavily by message handlers which may encounter an error during event/message processing. Available in VisualADE only.

Boolean throwError (GMessage *msgCausedError,
char *classname,
char *handlername,
char *errorname,
int severity,
char *errormsg);

GMsgHandler

SYNOPSIS:

GMsgHandler ();

DESCRIPTION

Creates an object which executes any GMessage object specifically sent to it. The GMessage object provides the needed data required by this GMsgHandler. GMsgHandlers are typically assigned to a GMsgManager so that the GMsgHandler adds functionality to the GMsgManager. This handler is analogous to the event handler concept except that the idea of an event has been expanded to include any type of message. Another analogous concept is the following:

Object	<---> GMsgManager
Function	<---> GMsgHandler
Function call	<---> GMessage
Switch statement	<---> GTranslator

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

Those raised by the individual GMsgHandlers derived from this base class.

CAVEATS:

None.

SEE ALSO:

GMsgManager, GMessage, GTranslator, GMsgCentral

METHODS:

allowMultipleCopiesInAManager

Return whether multiple instances of this handler can be assigned to a GMsgManager.

Boolean allowMultipleCopiesInAManager ()

allowMultipleCopiesInAManager

Specify whether multiple instances of this handler can be assigned to a GMsgManager.

void allowMultipleCopiesInAManager (Boolean flag)

convert_toDestType

Convert an incoming type to the given required for some specific parameter type supported by this handler. Available in VisualADE only.

virtual Boolean convert_toDestType (char **newValue, int srctype, int desttype);

create

Create and return a pointer to a copy of this handler.

virtual GMsgHandler *create (GMsgManager *mm = NULL, Boolean use_default_translations = True)

disablePercentReplacement

Disable the substitution of the values of any variables into their assigned parameters, if any. Available in VisualADE only.

Boolean disablePercentReplacement ()

disablePercentReplacement

Enable/Disable the substitution of the values of any variables into their assigned parameters, if any. Available in VisualADE only.

void disablePercentReplacement (Boolean flag)

enable

Specify whether this handler is enabled.

void enable (Boolean flag)

get_code

Return the code (i.e. verb code) assigned to this handler. (of the messages that this GMsgHandler handles).

int get_code ()

get_defaults

Return the default values (parameters, etc.) for this handler. Not used.

GMessage*get_defaults ()

get_invokingMsg

Returns which GMessage object is invoking the GMsgHandler. Used for debugging usually.

GMessage *get_invokingMsg ()

get_name

Return the name (i.e. verb) assigned to this handler (of the messages that this GMsgHandler handles).

virtual char *get_name ()

get_parmName

Return the name of the i-th parameter. This is only supported by a few handlers.

virtual char *get_parmName (int pnum)

get_parmType

Return the type of the i-th parameter. This is only supported by a few handlers.

virtual int get_parmType (int pnum)

get_required_component_type

Return the component name of the GMsgManagers this handler may be assigned to.

char *get_required_component_type ()

is_enabled

Returns whether this handler is enabled.

Boolean is_enabled ()

process_msg

Process the given GMessage if it is directed at this handler.

virtual Boolean process_msg (GMessage *msg)

register_me

Register this handler and it's name with the global list of handlers maintained in GMsgCentral.

void register_me ();

set_default_translations

Assign to this handler the default translations as specified in GMsgHandlers derived from this that have overridden this method to provide such translations.

virtual void set_default_translations (GTranslator *translator)

set_defaults

Specify the default values (parameters, etc.) for this handler. Not used.

void set_defaults (GMessage *msg)

set_invokingMsg

Specifies which GMessage object is invoking the GMsgHandler. Used for debugging usually.

void set_invokingMsg (GMessage * msg)

set_name

Assign the name (i.e. verb) assigned to this handler

void set_name (char *n)

set_required_component_type

Assign the component name of the GMsgManagers this handler may be assigned to.

void set_required_component_type (char *t)

GMsgHandler

throwError

Routine that is called when this handler detects an error. GMsgCentral is then called with the given data.

Boolean throwError (GMessage *msgCausedError,
char *classname,
char *handlername,
char *errorname,
int severity,
char *errormsg);

update_parmValue

Force update of a constrained parameter value of this handler and invoke the handler with the new values. Available in VisualADE only.

virtual Boolean update_parmValue (GMessage *msg, int pnum, char *newValue)

GMsgManager

SYNOPSIS:

GMsgManager ();

DESCRIPTION

Rarely instantiated directly.

Creates an object which is typically larger than most objects in an application. This manager contains resident methods which supply minimal functionality and a GTranslator and a list of GMsgHandlers to augment this functionality. This class is the base class of all Components in the system.

Object	<---> GMsgManager
Function	<---> GMsgHandler
Function call	<---> GMessage
Switch statement	<---> GTranslator

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

Specified by the classes derived from GMsgManager.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

None.

SEE ALSO:

GMsgHandler, GMessage, GTranslator, GMsgCentral

METHODS:

GMsgManager

add_changes

Ors the given change flag into this manager's change state.

void add_changes (int achange)

add_overlay

Specifies that the given 'overlay' manager intercept all messages sent to this manager and therefore gets to process them first.

void add_overlay (GMsgManager *overlay);

dispatch

Dispatch the given message to the destination GMsgManager.

Boolean dispatch (GMessage *msg, GMsgManager *mm);

find_instMatchingSearchSpec

Returns application data objects matching the given wild card search string. Available in VisualADE only.

virtual GObjOrFnInstance *find_instMatchingSearchSpec (char *search_spec, GSearchContext *context);

find_msg_handler

Returns the first GMsgHandler of the given name assigned to this manager. If none assigned then NULL is returned.

GMsgHandler *find_msg_handler (char *name);

get_changes

Returns whether this manager has changed or not. The manager usually looks at this if it receives something like an 'UpdateYourself' message.

int get_changes ()

get_instantiated

Return whether this has been internally instantiated or not. This indicates whether the internal initialization routines for this manager have been invoked yet.

Boolean get_instantiated ()

get_msgHandlerList

Return the list of all GMsgHandlers assigned to this manager.

virtual GMsgHandlerList *get_msgHandlerList ()

get_numparms

Return the number of parameters (i.e. variables) that have been assigned to this manager.

int get_numparms ();

get_parm

Returns the value of the i-th parameter.

char *get_parm (int parmnum);

get_translator

Return the translator assigned to this manager.

GTranslator*get_translator ();

get_type

Return the component name/type of this manager.

char *get_type ()

isa

Returns whether or not this manager is of the given component type.

Boolean isa (char *type);

replace_percents

Replace the variable references in the given message with their current values, if they exist in this manager's variable data. Available in VisualADE only.

GMessage * replace_percents (GMessage *msg);

replace_percents

Replace the variable references in the given string with their current values, if they exist in this manager's variable data. Available in VisualADE only.

Boolean replace_percents (char *origstr, String *newstr);

set_changes

Specifies whether this manager has changed or not.

void set_changes (int achange)

set_instantiated

Specify whether this has been internally instantiated or not.

void set_instantiated (int flag)

set_msgHandlerList

Specify the list of GMsgHandlers assigned to this manager.

void set_msgHandlerList (GMsgHandlerList *list)

set_parm

Assigns the given value to the i-th parameter.

void set_parm (int parmnum, char *parmname);

set_type

Specify the component name/type of this manager.

void set_type (char *t)

translate_and_broadcast

For any messages assigned in this manager's translator to the given event, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (GEvent *src, GMsgManager *mm);

translate_and_broadcast

For any messages assigned in this manager's translator to the given name, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (char *src, GMsgManager *mm);

translate_and_broadcast

For any messages assigned in this manager's translator to the given message, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (GMessage *src, GMsgManager *mm);

GTranslator

SYNOPSIS:

GTranslator ();

DESCRIPTION

Creates an table which maps text strings, GMessage names and GEvents to GMessages. Translators are typically created inside GMsgManager constructors to be used by the application as a convenient interface to get at the GMsgHandlers assigned to the GMsgManager.

PARAMETERS (Required):

None.

PARAMETERS (Optional):

None.

COMPONENT NAME:

None.

MESSAGES GENERATED:

None.

VARIABLES SET:

None.

EXCEPTIONS RAISED:

None.

CAVEATS:

Note that incoming GMessages are treated as if their name (i.e. verb) is an incoming text string and translated as such.

SEE ALSO:

GMsgHandler, GMessage, GMsgManager, GMsgCentral

METHODS:

add_translation

Add a text-string-to-message translation. The given id is optional and can be used later to remove the translation.

int add_translation (char *src, GMessage *dest, char *id = NULL);

add_translation

Add a event-to-message translation. The given id is optional and can be used later to remove the translation.

```
int     add_translation (
            G_EVENTTYPE type,
            G_EVENTKEY key,
            G_SHIFTSTATUS shiftstatus,
            GMessage *msg,
            char *id = NULL);
```

debug_msg_print

Prints the contents (name, parameters, etc.) of the given message in an easy-to-read format.

void debug_msg_print (GMessage *msg);

dispatch

Dispatch the given message to the destination GMsgManager.

Boolean dispatch (GMessage *msg, GMsgManager *mm);

find_translation

Return the index of the translation that translates the given string into a message. If none are found, then -1 is returned.

int find_translation (char *srcstring);

find_translation

Return the index of the translation that translates the given message into another message. If none are found, then -1 is returned.

int find_translation (GMessage *msg);

get_event_translation

Returns the number of event-to-message translations. If the given index ‘num’ is less than the number of event translations, then the associated event and message of the given index are also returned.

int ***get_event_translation (***
int num,
G_EVENTTYPE *type,
G_EVENTKEY *key,
G_SHIFTSTATUS *shiftstatus,
GMessage **msg);

get_string_translation

Returns the number of string-to-message translations. If the given index ‘num’ is less than the number of event translations, then the associated event and message of the given index are also returned.

int ***get_string_translation (***
int num,
char **srcstring,
GMessage **msg);

print_all_msgs

Whether to print all messages passing through this translator or not.

void ***print_all_msgs (Boolean flag)***

remove_event_translation

Remove the event-to-message translation with the given index.

void ***remove_event_translation (int index);***

remove_event_translations

Remove the event-to-message translation with the given id.

void ***remove_event_translations (char *id);***

remove_translation

Remove the string-to-message translation with the given index.

void ***remove_translation (int index);***

remove_translations

Remove the string-to-message translation with the given id.

void remove_translations (char *id);

translate_and_broadcast

For any messages assigned in this manager's translator to the given event, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (GEvent *src, GMsgManager *mm);

translate_and_broadcast

For any messages assigned in this manager's translator to the given string, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (char *src, GMsgManager *mm, GObjOrFnInstance *instance = NULL);

translate_and_broadcast

For any messages assigned in this manager's translator to the given message, broadcast them to the given GMsgManager. If, however the particular message already has an assigned destination, then this parameter 'mm' is ignored for this particular message.

Boolean translate_and_broadcast (GMessage *src, GMsgManager *mm);

Index

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z