OPaC: A Portable Graphical User Interface

Portability Guide
and
Semester Project Report on Port to Unix/X11

Laurent Bovet

February 9, 1998

Contents

Chapter 1

Introduction

1.1 Audience

This document is intended to readers who want to know how to port the OPaC Class Library to a new platform.

The reader should know the oriented object programming concepts and have some basics of graphical user interface programming and X window.

It also describes the way it was done for Unix/X11 as a semester project at the Laboratoire de Microinformatique of the Ecole Polytechnique Fédérale de Lausanne, Switzerland.

This document is not a reference, nor a guide for development with OPaC. The reader shall refer to the source code for information about the architecture of OPaC.

1.2 About this document

This document is divided into two chapters that have the same structure. The first chapter is the portability guide. It describes the way to implement each feature of the system dependent part of OPaC. The second chapter explains how each feature of the system dependent part has been implemented for the port to Unix/X11, as far it is not trivial.

1.3 OPaC in a word

The OPaC Class Library has been written in C++. It enables programmers to dynamically edit their applications' user interfaces at run time, modifying their look and behaviour in a few mouse clicks. This has been achieved thanks to dynamic object allocation and dynamic message passing, thus suppressing the need for scripting.

The OPaC class library has been under development since 1990. OPaC started as a wrapper library aimed at writing highly portable code by abstracting the system's API into classes.

OPaC expanded to totally encompass the graphical user interface (GUI) : a collection of widgets was implemented based only on low level graphic operations and event handling was abstracted by defining a set of generic user events. In order to reduce development time, a dynamic user interface editor was embedded into OPaC, which allows the programmer and the end user to modify the interface even while the application is running, changing not only its look but also its behaviour.

The OPaC Library has been designed to be portable. The source code is split into two categories: system dependent and system independent. This reduces porting efforts.

A lot of interesting features such as automatic memory recycling, garbage collection, smart references, dynamic method invocation, automatic class registration, etc. have been added in the process.

Chapter 2

Portability Guide

To port OPaC, one must create adequate data structure and write methods that access these structures. These structures are described in this chapter, highlighting what information it should content. Each method is referenced and its behaviour described.

The system dependent files are located in a sub-directory of `sd/` named after the specific system (e.g `sd/linux`). These files are:

`sd_colman.cxx`	Color Manager
`sd_event.cxx`	Event Handling
`sd_file_io.cxx`	File Input/Output
`sd_fontman.cxx`	Font Manager
`sd_grail.cxx`	Graphical Interface Layer (Grail) Drawing Methods
`sd_grailcur.cxx`	Grail Mouse Cursor
`sd_grailfnt.cxx`	Grail Font
`sd_grailpxm.cxx`	Grail Pixmap
`sd_grailwdo.cxx`	Grail Window
`sd_interface.h`	System dependent include file
`sd_main.cxx`	Startup code
`sd_memory.cxx`	Memory

2.1 Interface

`sd_interface.h`

This file should include all header files of the system API used by the system dependent part. Put in this file all global definitions and declaration the implementation will use.

It also declares the following underlaying structures that are encapsulated in system independent structures:

1. `OPaC_GrPort`
   This contains all informations about a context. A context is a set of properties for graphic outputs.

   The fields should include:

   - A reference to the graphic area receiving the output, i.e. a window or an off-screen area
   - Foreground drawing color

- Background drawing color
- Clipping region, i.e. a rectangular mask that constrains drawing inside its bounds
- Current brush properties
- Current font properties

2. `OPaC_GrPixmap`
   This contains a reference to a pixmap, i.e. an off-screen area where it is possible to draw and store temporary pieces of graphics.

3. `OPaC_GrWindow`
   This contains a reference to a window. It also may contain information shared by all the windows, such as a name used to identify which application a window is from.

4. `OPaC_ColorMan`
   This should contain a reference to a color map or a palette for old indexed color systems.

5. `OPaC_GrCursor`
   This contains a reference to a mouse cursor structure.

6. `OPaC_GrFont`
   This contains a reference to a font structure.

2.2 Graphic Interface Layer

OPaC uses double buffering for every output to the screen. Every drawing operation is performed on a pixmap having the same size than the actual window. The pixmap is copied to the window each time it is needed; in case of new drawing and if the window must be repaint.

So, the Grail must implement two kinds of "drawable" graphic area. One that is off-screen and commonly called pixmap and the other that is the actual window on the visible screen. This one is never given drawing operations directly.

The `OPGrailWindow` class inherits from `OPGrailPixmap` which inherits from `OPGrailPort` which is somewhat abstract.

The drawing methods apply to `OPGrailPort` objects. Because of double buffering, the window class is just a pixmap class that adds window specific methods.

The class definitions are in `OPaC/grail.h`.

2.2.1 Drawing

`sd_grail.cxx`

Global functions

```
void
OPaC_GrailPort_Initialise (OPaC_GrPort*& info,
                           OPColorPixel f,
                           OPColorPixel b)
```

Initialise an empty Grail Port.

`info` is a reference to the Initialised graphic context. More generally, info is a pointer to the encapsulated system dependent structure.
`f` is the foreground color
`b` is the background color

```
void
OPaC_GrailPort_Kill (OPaC_GrPort*& info)
```

Free resources allocated in `OPaC_GrailPort_Init`

```
void
OPaC_GrailPort_BeginDraw (OPaC_GrPort* info,
                          int x1, int y1, int x2, int y2,
                          OPaC_ColorMan* cman)
```

Allocate resources and prepare the port for drawing output. This does not set the clip mask or any attribute of the graphic context.
`x1, y1, x2, y2` are coordinates of the rectangle where the drawing will take place.
`cman` is a reference to the color manager used for this port and the following drawing operations

---

```
void
OPaC_GrailPort_EndDraw (OPaC_GrPort* info)
```

Free resources allocated in `OPaC_GrailPort_BeginDraw`

```
void
OPaC_GrailPort_Flush (OPaC_GrPort* info)
```

Flush the graphical output. This forces the drawing operations to be executed in case of queuing. This will not display the result on the screen, because of the double buffering.

```
void
OPaC_GrailPort_SetClipRect (OPaC_GrPort* info,
                            int x1, int y1, int x2, int y2)
```

Define a clipping rectangle. This will be called only between a `OPaC_GrailPort_BeginDraw` and a `OPaC_GrailPort_EndDraw`.

The clipping rectangle is a sub-area of the port that will accept a drawing operation. Only the part of the drawing inside the clipping rectangle will be visible.

```
void
OPaC_GrailPort_SetForeColor (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPColorManIn cman)
```

Define the foreground color. This will be called only between a `OPaC_GrailPort_BeginDraw` and a `OPaC_GrailPort_EndDraw`.

```
void
OPaC_GrailPort_SetBackColor (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPColorManIn cman)
```

Define the background color. This will be called only between a `OPaC_GrailPort_BeginDraw` and a `OPaC_GrailPort_EndDraw`.

```
void
OPaC_GrailPort_SetLineStyle (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPLineStyle style,
                             OPColorManIn cman)
```

Define the line style. This will be called only between a `OPaC_GrailPort_BeginDraw` and a `OPaC_GrailPort_EndDraw`.

`style` can have the following values:

```
OP_LINE_SOLID ________
OP_LINE_DOT   ........
OP_LINE_DASH  __ __ __
```

```
void
OPaC_GrailPort_SetFont (OPaC_GrPort* info,
                        OPaC_GrFont* font_info)
```

Define the current font. This will be called only between a `OPaC_GrailPort_BeginDraw` and a `OPaC_GrailPort_EndDraw`.

Target specific methods

```
void
OPGrailPort::DrawDot (OPCoord x, OPCoord y)

void
OPGrailPort::DrawDot (OPCoord x, OPCoord y, OPColorPixel color)
```

Draw a single pixel.

```
void
OPGrailPort::DrawLineDX (OPCoord x, OPCoord y, OPCoord dx)

void
OPGrailPort::DrawLineDY (OPCoord x, OPCoord y, OPCoord dy)
```

Draw a line (segment), using one of the predefined dash styles. The segment is always hair thin... The last dot will be drawn.
A line length of 0 won't draw anything !

```
void
OPGrailPort::DrawLineToX (OPCoord x1, OPCoord y1, OPCoord x2)

void
OPGrailPort::DrawLineToY (OPCoord x1, OPCoord y1, OPCoord y2)
```

Draw a line (segment), using one of the predefined dash styles. The segment is always hair thin... The last dot will be drawn.
A line with both extremities at the same location will draw just a single dot.

```
void
OPGrailPort::DrawRect (OPRectIn box, OPRelCoord margin)
```

```
void
OPGrailPort::DrawRect (OPCoord x1, OPCoord y1,
                                   OPCoord x2, OPCoord y2)
```

Draw rectangular box. The bottom left corner will be drawn at [x2;y2]. This might not be the case on every system, so be careful to implent this the OPaC way.

```
void
OPGrailPort::FillRect (OPRectIn box, OPRelCoord margin)

void
OPGrailPort::FillRect (OPCoord x1, OPCoord y1,
                       OPCoord x2, OPCoord y2)
```

Fill a rectangular box. The surface drawn is exactly the same as with DrawRect. Beware, this is usually not the case on other systems !!!

Be careful to implement this method the right way.

```
void
OPGrailPort::DrawHairLine (OPCoord x, OPCoord y,
                           OPCoord dx, OPCoord dy)

void
OPGrailPort::DrawHairLineTo (OPCoord x1, OPCoord y1,
                             OPCoord x2, OPCoord y2)
```

Draw a line segment. The segment is always hair thin and is not dashed... The last dot will be drawn.

```
void
OPGrailPort::DrawAALine (OPCoord x, OPCoord y,
                         OPCoord dx,OPCoord dy)

void
OPGrailPort::DrawAALineTo (OPCoord x1, OPCoord y1,
                           OPCoord x2, OPCoord y2)
```

Draw a line segment with anti-aliasing. The segment is always hair thin and is not dashed... The last dot will be drawn.

```
void
OPGrailPort::ShowText (OPCoord x, OPCoord y, OPStringIn text,
                       Count start, Count length,
                       Card8 pos,
                       OPCoord dx)

void
OPGrailPort::ShowText (OPRectIn box, OPStringIn text,
                       Count start, Count length,
                       Card8 pos, OPRelCoord margin)

void
OPGrailPort::ShowText (OPCoord x1, OPCoord y1,
                       OPCoord x2, OPCoord y2,
                       OPStringIn text,
                       Count start, Count length,
                       Card8 pos)
```

Draw a piece of text, which will be extracted from a string starting at character `start` and up to `length` characters. The way to position the text is specified by `pos` and `dx` is used if aligned positioning has been requested.

`pos` is a bitwise-ored combination of the following values:

```
OP_TEXT_HPOS_LEFT       >Hello    <
OP_TEXT_HPOS_CENTER     >  Hello  <
OP_TEXT_HPOS_RIGHT      >    Hello<
OP_TEXT_HPOS_ALIGN      >H e l l o<

OP_TEXT_VPOS_BASE     use baseline as vertical ref.
OP_TEXT_VPOS_TOP      use top border as vertical ref.
OP_TEXT_VPOS_CENTER   use text middle as vertical ref.
OP_TEXT_VPOS_BOTTOM   use bottom border as vertical ref.
```

`pos` analysis can be done using the following masks:

```
OP_TEXT_HPOS_MASK
OP_TEXT_VPOS_MASK
```

```
void
OPGrailPort::CopyRect (OPGrailPortIn source,
                       OPCoord source_x, OPCoord source_y,
                       OPCoord dest_x1, OPCoord dest_y1,
                       OPCoord dest_x2, OPCoord dest_y2)

OPGrailPort::CopyRect (OPGrailPortIn source,
                       OPCoord source_x, OPCoord source_y,
                       OPRectIn dest_box, OPRelCoord dest_margin)
```

Copy a rectangle from the source port to the current port, into the specified box, from the source position given by its top/left corner.

This will copy the pixels, including the one of the bottom right corner. This is still the same policy as with the rectangle.

If the attribute `use_transparency` of the port is set to `TRUE`, the color given by the attribute `transparent_color` must be considered as transparent.

```
void
OPGrailPort::SetColorPixels (OPCoord x, OPCoord y, OPCoord dx,
                             const OPColorPixel* pixels)

void
OPGrailPort::SetGrayPixels (OPCoord x, OPCoord y, OPCoord dx,
                            const Card8* intensity)

void
OPGrailPort::SetRGBPixels (OPCoord x, OPCoord y, OPCoord dx,
                           const Card8* red,
                           const Card8* green,
                           const Card8* blue)
```

`SetColorPixels` takes an array of `dx` or more pixels and draw them on a line of the port beginning at `x`, `y`

`SetGrayPixels` takes an array of `dx` or more intensity values (0-255) draw them on a line of the port beginning at `x`, `y`

`SetRGBPixels` takes three arrays of `dx` or more intensity values correponding to red, green and blue components of a color, then draw them on a line of the port beginning at `x`, `y`.

2.2.2 Pixmap

`sd_grailpxm.cxx`

```
OPGrailPixmap::OPGrailPixmap ()
```

Initialise the Grail Pixmap instance. This won't allocate any bitmap; do this using `OPGrailPixmap::Initialise` !

```
OPGrailPixmap::~OPGrailPixmap ()
```

Destroy cleanly a Pixmap : this will free an existing bitmap and associated data.

```
void
OPGrailPixmap::BeginDraw (OPRectIn box, OPRelCoord margin)

void
OPGrailPixmap::BeginDraw (OPCoord x1, OPCoord y1,
                          OPCoord x2, OPCoord y2)
```

Start drawing into the pixel map. Put here all allocations or initializations needed by this operation. One can draw within a rectangle given an inner margin or in a rectangle given by its coordinates.

```
void
OPGrailPixmap::EndDraw ()
```

End of drawing. Frees the resources allocated by BeginDraw.

```
void
OPGrailPixmap::Initialise (OPGrailPixmapIn window,
                           OPCoord dx,
                           OPCoord dy)

void
OPGrailPixmap::Initialise (Count depth, OPCoord dx, OPCoord dy)
```

Initialise a new pixel map. If one already existed, just delete it and start a new. If `window` is not specified, the default screen will be used as model. `window` is actually from type `OPGrailWindow` that is inherited from `OPGrailPixmap`, hence the type in the parameter list. The depth of the new pixmap must be set to the depth of `window` or equal to `depth` in the second method.

```
Bool
OPGrailPixmap::SystemGetRGBPixels (OPCoord x, OPCoord y,
                                   OPCoord dx,
                                   Card8* r, Card8* g, Card8* b)
```

Reads `dx` pixels from a line of the pixmap beginning at `x`, `y`. The RGB values of the pixels must be stored in the arrays `r`, `g` and `b`, respectively.

2.2.3 Window

`sd_grailwdo.cxx`

```
OPGrailWindow::OPGrailWindow ()
```

Construct a new window instance. Allocate memory for encapsulated system dependent structure and set empty values to attributes.

Initialise name attributes.

```
OPGrailWindow::~OPGrailWindow ()
```

Delete the window. Free any resourcees allocated by `OPGrailWindow`.

```
void
OPGrailWindow::Initialise (OPWindowIn view,
                           OPWindowStyle style,
                           OPStringIn name,
                           OPRectIn box, OPRelCoord margin)

void
OPGrailWindow::Initialise (OPWindowIn view,
                           OPWindowStyle style, OPStringIn name,
                           OPCoord x1, OPCoord y1,
                           OPCoord x2, OPCoord y2)
```

Create the real window but do not open it. `style` is a bitwise-ored value of the following items:
Modes:

```
OP_WIN_NAKED          naked window (no frame)
OP_WIN_NORMAL         normal window without title
OP_WIN_TITLE          normal window with title
```

Attributes:

```
OP_WIN_MOVE_OK        window may be moved
OP_WIN_SIZE_OK        window may be sized
OP_WIN_REDUCE_OK      window may be reduced (iconified)
OP_WIN_CLOSE_OK       window may be closed
```

Extended attributes:

```
OP_WIN_TRANSPARENT    window should be transparent
OP_WIN_FLOAT_TOOL     window should be considered as a tool
OP_WIN_TOP_LEVEL      window should be always on top
```

These flags can be selected through the following masks:

```
OP_WIN_MODE_MASK      mask for mode
OP_WIN_ATTR_MASK      mask for attributes
OP_WIN_EXTEND_MASK    mask for extended attributes
```

The actual system window is likely to store a pointer or a handle to the associated `OPGrailWindow` object. This can be done given that the system window structure usually allows users to store associated data. This stored pointer will be usefull to find the OPaC window structure given a system window.

```
void
OPGrailWindow::ChangeName (OPStringIn name)
```

Select another name for the current window. This is only possible if the window has been defined with a title bar.

```
void
OPGrailWindow::Open ()
```

Open an existing window on the screen. This is usually just a display command. The following attributes must be set:

```
this->is_open    = TRUE;
this->is_reduced = FALSE;
```

```
void
OPGrailWindow::Reduce ()
```

Minimize the window, i.e. reduce it to an icon. The following attributes must be set:

```
this->is_open    = TRUE;
this->is_reduced = TRUE;
```

```
void
OPGrailWindow::Close ()
```

Close the window. Do not destroy anything, just hide the window.
The following attributes must be set:

```
this->is_open    = FALSE;
this->is_reduced = FALSE;
```

```
void
OPGrailWindow::ActivateAsFrontWindow (Bool front)
```

Give the focus to the window.
`front` specifies if the window must be brought to the top.

```
void
OPGrailWindow::SysChangedActive (Bool yes)

void
OPGrailWindow::SysChangedOpen (Bool yes)

void
OPGrailWindow::SysChangedVisible (Bool yes)

void
OPGrailWindow::SysChangedReduced (Bool yes)
```

This routines must set the attributes `is_active`, `is_open` and `is_reduced` to the right value. They also have to store information about the new state if needed.

```
Bool
OPGrailWindow::IsActive () const
```

Just return the `active` attribute...

```
void
OPGrailWindow::SetOrigin (OPCoord x, OPCoord y)
```

Set the window's origin. This will be reflected immediately by the window manager so that this can be used for dragging.

```
void
OPGrailWindow::GetOrigin (OPCoordOut x, OPCoordOut y) const
```

Return the origin of the window. Just use the `bounding_box` attribute.

```
void
OPGrailWindow::SetSize (OPCoord dx, OPCoord dy)
```

Set the window size. The `bounding_box` attribute must be updated according to the new size.

```
void
OPGrailWindow::GetSize (OPCoordOut dx, OPCoordOut dy) const
```

Return the window's size. Use the `bounding_box` attribute.

```
void
OPGrailWindow::UserChangedGeometry ()
```

Get the new position and size of the window to fill in the `bounding_box` attribute and call the `UserChangedGeometry` method of the nested view if the box has changed.

```
void
OPGrailWindow::GetFrameSize (OPCoordOut xl, OPCoordOut yt,
                             OPCoordOut xr, OPCoordOut yb)


void
OPGrailWindow::GetFrameSize (OPWindowStyle style,
                             OPCoordOut xl, OPCoordOut yt,
                             OPCoordOut xr, OPCoordOut yb)
```

Return the frame size, i.e. the thickness of the decoration around the window content that is given by the window manager. The second method is static and can't currently be implemented on some systems. Some ports of OPaC don't implement these methods.

```
void
OPGrailWindow::BeginDraw (OPRectIn box, OPRelCoord margin)

void
OPGrailWindow::BeginDraw (OPCoord x1, OPCoord y1,
                          OPCoord x2, OPCoord y2)
```

Start drawing into the window. This will first draw to the double buffering pixmap. These methods have to call `super::BeginDraw` in order to transmit the message to the parent. Other information can be initialized by theses methods, as the coordinates of the rectangle for further repainting.

```
void
OPGrailWindow::EndDraw ()
```

Frees resources allocated in `BeginDraw`. It also has to call `super::EndDraw`.

```
void
OPGrailWindow::Flush ()
```

Send the graphic output to the screen. This should update the part of the screen which has been changed between `BeginDraw` and `EndDraw`.

```
void
OPGrailWindow::Redraw ()
```

Redraw the window, i.e. copy the double buffering pixmap to the window.

```
Bool
OPGrailWindow::FindOwnWindow (OPCoord x, OPCoord y,
                              OPGrailWindowOut wdo,
                              Count index)
```

Find which window belonging to the application, if any, contains the specified coordinate. It looks for the nth window (0 is the topmost). Return `FALSE` if no OPaC window is found or if an OPaC window is overlapped by a non-OPaC window.

2.2.4 Font

`sd_grailfnt.cxx`

```
OPGrailFont::OPGrailFont ()
```

Initialise the attributes to null values.

```
OPGrailFont::~OPGrailFont ()
```

Free resources allocated for the font.

```
void
OPGrailFont::StoreInstance (OPStorageIn storage)
```

Archive the properties of the font. This should save the `name`, `style` and `size` attributes.

```
void
OPGrailFont::RestoreInstance (OPStorageIn storage)
```

Restore the attributes saved by `StoreInstance`.

```
Bool
OPGrailFont::Initialise (OPStringIn name,
                         OPStringIn style, OPCoord size)
```

Set the `name`, `style` and `size` attributes. Return the result of a call to `RealizeFont` (see below).

```
Bool
OPGrailFont::RealizeFont ()
```

Allocate resources for the font. Load or choose the system font according to the attributes set by `Initialise`. Return `TRUE` if the font has been realized, `FALSE` otherwise.

```
void
OPGrailFont::GetFontName (OPStringOut name) const

void
OPGrailFont::GetFontStyle (OPStringOut name) const

OPCoord
OPGrailFont::ReturnFontSize () const
```

Just returns the attributes of the font.

```
OPRelCoord
OPGrailFont::ReturnAscender () const

OPRelCoord
OPGrailFont::ReturnDescender () const
```

Returns the size properties of the font. The ascender is the maximum distance between the base line and the top of a character.

The descender is the maximum distance between the base line and the bottom of a character. It is a *negative* value.

These are constants value for a given font.

```
OPCoord
OPGrailFont::ReturnWidth (const LatinChar* text) const

OPCoord
OPGrailFont::ReturnWidth (OPStringIn text) const

OPCoord
OPGrailFont::ReturnWidth (OPStringIn text,
                          Count start, Count length) const
```

Return the width of the given string written in the current font.

2.2.5 Mouse Cursor

`sd_grailcur.cxx`

```
OPGrailCursor::OPGrailCursor ()
```

Set the attributes to null values.

```
OPGrailCursor::~OPGrailCursor ()
```

Free the resources allocated for the cursor.

```
void
OPGrailCursor::Initialise (OPCoord hotx, OPCoord hoty,
                           OPCoord dx, OPCoord dy,
                           OPGrailPixmapIn fg,
                           OPCoord xf, OPCoord yf,
                           OPGrailPixmapIn bg,
                           OPCoord xb, OPCoord yb)
```

Initialise a cursor from the specified pixmaps. The background will be opaque wherever it is set to black. The foreground will be painted wherever a black background has been drawn.

The foreground and background pixmaps are extracted respectively from the given pixmaps `fg` and `bg`. They both are `dx` by `dy` sized and are fetched at (`xf`, `yf`) for the foreground and at (`xb`, `yb`) for the background.

(`hotx`, `hoty`) is the hotspot of the cursor.

```
void
OPGrailCursor::Activate () const
```

Make the cursor visible on the screen. This is not window dependent; the cursor shape is changed for all windows.

```
void
OPGrailCursor::ResetToNormal () const
```

Switch back to default cursor.

```
void
OPGrailCursor::Hide () const
```

Remove any cursor from the screen.

2.3 Event Handling

`sd_event.cxx`

The event handling is implemented through the `WaitForEvent` method of the `OPEvent` class. This class is instanciated once only to create the current event object. `WaitForEvent` fills in this object as soon as a system event occurs.

The attributes of the event to fill are:

```
Card32        type;       //  event type
Card32        id;         //  event unique ID

OPWindowRef   window;     //  window in which event occurred

OPRelCoord    x;          //  pointer: horizontal position (wdo relative)
OPRelCoord    y;          //  pointer: vertical position (wdo relative)
Card32        buttons;    //  pointer: set of current buttons
```

```
Card32        change;     //  specific: change (button, key, etc)
OPObjectRef   data;       //  specific: data (user data, etc)
OPAtomRef     name;       //  specific: name (key name, etc)
Card32        code;       //  specific: code (OEM code, etc)

OPKey         key;        //  keyboard: shift, control & meta keys
```

The following paragraphs describe how to fill the event structure for each type of event listed below:

```
OP_EVENT_NONE         no event - used internally

OP_EVENT_PRESS        pointing device: button pressed
OP_EVENT_RELEASE      pointing device: button released
OP_EVENT_MOTION       pointing device: moved

OP_EVENT_KEY_DOWN     keyboard: key pressed
OP_EVENT_KEY_REP      keyboard: automatic key repetition
OP_EVENT_KEY_UP       keyboard: key released
OP_EVENT_KEY_CHANGE   keyboard: one shift/modifier changed
```

The `key` attribute is a structure composed by the following attributes:

```
Card8       type;       //  key type (either alphanumeric or special)
Card8       code;       //  special key code (ASCII upper case)
Card16      value;      //  Unicode key value or other special value
Card32      modifiers;  //  associated extra modifiers
```

The `key.modifiers` attribute is a bitwise-ored combination of the following values:

```
OP_EV_SHIFT
OP_EV_CONTROL
OP_EV_CAPS_LOCK
OP_EV_META_1
OP_EV_META_2
OP_EV_META_3
OP_EV_META_4
OP_EV_META_5
```

The class and constants definitions are in `OPaC/event.h` and `OPaC/key.h`

2.3.1 Mouse

Each mouse action is reported by an event.

`OP_EVENT_PRESS` A mouse button is pressed. The following attributes must be set: `window`, `x`, `y`, `buttons`, `change`, `key`.

`buttons` is the state of the mouse buttons represented by a bitwise-ored combination of the following values:

```
OP_EV_BUTTON_LEFT
OP_EV_BUTTON_MIDDLE
OP_EV_BUTTON_RIGHT
```

`change` is a bit field showing which button has been changed. It is one of the above values.

`key` must be updated. Only the `modifiers` attribute has to be set. The others have no sense for a mouse event.

`OP_EVENT_RELEASE` A mouse button is released.

The attributes to set are the same as for `OP_EVENT_PRESS`.

`OP_EVENT_MOTION` The mouse is moved.

The following attributes must be set: `window`, `x`, `y`, `buttons`, `key`. They all have the same meaning as for the other mouse events.

2.3.2 Keyboard

`OP_EVENT_KEY_DOWN` A key is pressed. The following attributes must be set: `window`, `x`, `y`, `buttons`, `change`, `key`.

`change` is a bit field showing which modifiers have been changed since last event. This is a bitwise-ored combination of the following values:

```
OP_EV_SHIFT
OP_EV_CONTROL
OP_EV_CAPS_LOCK
OP_EV_META_1
OP_EV_META_2
OP_EV_META_3
OP_EV_META_4
OP_EV_META_5
```

`key.modifiers` is the current modifiers state.

`key.type` is one of the following values, according to the kind of key pressed:

```
OP_KEY_NONE         no key :-)
OP_KEY_SPECIAL      special key (PRINT SCRN, MACRO, UNDO...)
OP_KEY_ALPHANUM     any character
OP_KEY_FUNCTION     function keys F0..Fn
OP_KEY_RETURN       return key (that's the "<-+" key")
OP_KEY_ENTER        extended enter key (numeric key pad)
OP_KEY_TAB          tabulation key
OP_KEY_BACKSPACE    backspace (that's the "<--" key)
OP_KEY_DELETE       delete forward character
OP_KEY_CURSOR       cursor (up/down/left/right/page/home...
```

For any key to which corresponds a Latin-1 character, use the `OP_KEY_ALPHANUM` type. `key.value` must be set to this character. `key.code` must be set to the upper-case version of this character, if any. `key.name` is left empty.

For the common keys, i.e. *return, enter, tab, backspace* and *delete*, use the corresponding type and set the `key.name` attribute to the type string without `OP_KEY_` before. `key.code` and `key.value` are left empty.

For the cursor keys, use the `OP_KEY_CURSOR` type. Fill in the `key.code` attribute with the corresponding value listed below:

```
OP_CURSOR_SPECIAL     machine specific cursor key
OP_CURSOR_UP          move one line up
OP_CURSOR_DOWN        move one line down
OP_CURSOR_LEFT        move one character left
OP_CURSOR_RIGHT       move one character right
OP_CURSOR_PAGE_UP     move one page up
OP_CURSOR_PAGE_DOWN   move one page down
OP_CURSOR_HOME        move to line beginning
OP_CURSOR_END         move to line end
OP_CURSOR_TOP         move to document top
OP_CURSOR_BOTTOM      move to document bottom
```

`key.name` must be set to the type string without `OP_CURSOR_` before and replace underscore characters by spaces. `key.value` is left empty.

Treat the numeric keypad like the other character keys. OPaC need not to see the difference.

For the function keys, use the `OP_KEY_FUNCTION` type. Set the `key.name` attribute to `"Fxx"` where `xx` is the value written on the function key. The `key.code` and `key.value` are left empty.

Any other key that should be handled is considered as a special key and has `OP_KEY_SPECIAL` type. Set the `key.name` attribute to the following values for the usual keys:

```
"CLEAR"
"ESCAPE"
"EXECUTE"
"HELP"
"INSERT"
"NUM LOCK"
"PAUSE"
"PRINT"
"RETURN"
"SELECT"
"SEPARATOR"
```

The `key.code` and `key.value` are left empty.

2.3.3 Other events

Some other events have to be treated:

Timer In order to provide some computing while waiting user events, a timer event must be given periodically. Its type must be set to `OP_TIMER_TICK`. The period should be about 50 milliseconds. The other attributes remain empty.

Exposure Some systems give the responsability to the program to repaint the window after being hidden by aother window. They usually provide an event to notify that the window should be entirely or partially repainted.

No OPaC event exists to perform this operation. When an event like that is received, the `Redraw` method of the window must be called.

Size and position When the user or the window manager changes the geometry of the window, the event handler must call the `UserChangedGeometry` method of the modified window. A window iconification has to be notified through a call to its `SysChangedReduced` method.

Focus When a window gets or looses the focus, it should be notified through a call to its `SysChangedActive` method.

Close When the user or the window manager tries to kill a window, the event handler should call the `PostQuitEvent` method of the current event object. This leads to the end of the application.

2.3.4 Methods

```
void
OPEvent::HandOverToWindow (OPWindowIn window)
```

Hand the event capture from the current window to the specified one. This may be useful if some dragging op creates a new window and needs to give it the focus.

```
OPEvent::OPEvent ()
```

Initialise an event record. As only one `OPEvent` object exists, this method is called once only.

```
OPEvent::~OPEvent ()
```

Destroy the event. Free resources allocated in `OPEvent::Initialise`.

```
Bool
OPEvent::WaitForEvent ()
```

This is the main method. It waits for system events and fills in the attributes as described above. This is usually a big switch statement.

```
void
OPEvent::DispatchEvent()
```

Just call the system independent `DispatchEvent` method of the window concerned by the event.

```
void
OPEvent::PostQuitEvent ()
```

Indicate that the application has to quit. This has to lead to set the `is_dead` attribute of the event object to `TRUE`.

```
void
OPEvent::Initialise ()
```

Allocate resource for the event object. As only one `OPEvent` object exists, this method is called once only.

```
void
OPEvent::Kill ()
```

Clean up before exiting the application.

2.4 Color Manager

`sd_colman.cxx`

The color manager should detect the type of display to choose a color model and, if needed, use a colormap. This system dependent part interfaces the OPaC color manager to the underlaying color system of the specific platform.

The class definition is in `OPaC/grail.h`

```
OPColorMan::OPColorMan ()
```

Initialise a color manager. Determine if it will have to use an indexed or a true color model.

This method must call `ComputeNearestColor` to set up a table for gray levels. `ComputeNearestColor` is a system independent method.

```
OPColorMan::~OPColorMan ()
```

Clean up the color manager.

```
OPColorPixel
OPColorMan::MakeTrueColor (Card8 r, Card8 g, Card8 b) const
```

Return the pixel value for the RGB components. This will be used to get a true color pixel representation.

```
Bool
OPColorMan::FindColor (OPColorPixel p,
                       Card8Out r, Card8Out g, Card8Out b) const
```

Return the color components from a pixel value.

2.5 Font Manager

`sd_fontman.cxx`

The font manager registers the system fonts. A database is build by the constructor `OPFontMan::OPFontMan`.

Since the properties and the name of fonts are system specific, this section is not extensively documented. In the future, OPaC should provide or use an universal font system, such as True Type or a custom system.

To port OPaC without implementing the font manager, leave `sd_fontman.cxx` empty and implement `OPGrailFont::RealizeFont` (p. 18) to always realize the default font.

2.6 Memory

`sd_memory.cxx` The memory manager allocates system memory. This is easily implemented on systems that provide entire virtual space memory allocation. Some systems need special handling for allocation, this special handling has to be treated in this file.

```
Bool
SysMemOpenZone (const LatinChar* name,
                Card32Out zone_id)
```

Create a system zone if the system uses allocation zones.

```
Bool
SysMemCloseZone (Card32In zone_id)
```

Close a memory zone.

```
Bool
SysMemAlloc (Card32In zone_id, Card32In size,
             BytePtrOut ptr,
             Card32Out allocated,
             Card32Out mem_id)
```

Allocate memory in the specified zone.

```
Bool
SysMemFree (Card32In zone_id,
            BytePtrIn ptr,
            Card32In allocated,
            Card32In mem_id)
```

The memory free function will have to free the memory which has been allocated previously by SysMemAlloc.

2.7 File Input/Output

`sd_file_io.cxx` These file functions handle access to files. They should not be difficult to implement on a common system, since sequential file access is popular.

```
OPaC_FileDesc
OPaC_OpenFileRead (OPStringIn name)
```

Open a file for reading in it.

```
OPaC_FileDesc
OPaC_OpenFileWrite (OPStringIn name)
```

Open a file for writing in it.

```
OPaC_FileDesc
OPaC_OpenFileAppend (OPStringIn name)
```

Open a file for appending data to it.

```
Size
OPaC_ReadFile (OPaC_FileDesc file, void* void_buffer, Size size)
```

Read a chunk of data from an open file.

```
Size
OPaC_WriteFile (OPaC_FileDesc file,
                const void* void_buffer, Size size)
```

Write a chunk of data to a file.

```
Bool
OPaC_SeekFileAbs (OPaC_FileDesc file, Int32 offset)
```

Move the reading/writing head `offset` bytes from the beginning of the file.

```
Bool
OPaC_SeekFileRel (OPaC_FileDesc file, Int32 offset)
```

Move the reading/writing head `offset` bytes from the current position. `offset` can be negative.

```
Size
OPaC_TellFilePos (OPaC_FileDesc file)
```

Returns the position of the reading/writing head from the beginning of the file.

```
Size
OPaC_TellFileSize (OPaC_FileDesc file)
```

Returns the file size.

```
Bool
OPaC_FlushFile (OPaC_FileDesc file)
```

Synchronize the in-memory file and the stored version. Flush all read/write buffers.

```
Bool
OPaC_TruncFile (OPaC_FileDesc file)
```

Truncate the file at he current reading/writing head position.

```
Bool
OPaC_ResizeFile (OPaC_FileDesc file, Size size)
```

Truncate the file at most `size` bytes.

```
Bool
OPaC_CloseFile (OPaC_FileDesc file)
```

Close the open file.

2.8 Startup

`sd_main.cxx`

This file contains the `main()` function. This function should contain all the initialisations that have to be made before all and which are system dependent. `OPaC_MainInit` must be called to create the application.

Chapter 3

Report on Port of OPaC to Unix/X11

This chapter explains how the port was done to Unix/X11. The development has been done under Linux with gcc. The Xlib programming has been realized with the help of O'Reilly & Associates X window guides [1, 2].

A "spiral" approach has been chosen, it means that the features have been iteratively implemented through several passes. The goal of the first pass was to get the Builder application (provided with OPaC Class Library) to work. The methods have been implemented the simpliest way as possible. Each next pass improved the quality of the implementation in matters of specification matching and performance optimisation. The product, at the time this document has been written is valid for a wide range of applications to run under Linux. Further development should be done to get a full Unix portable version. See chapter 4 for the limitations and bugs of the current version.

Each feature of the portability guide is described in this chapter explaining which Xlib primitives has been used and, the way some tricky parts have been implemented.

This should help the programmer who want to port OPaC to a new platform and is a basis for further development on the port to Unix.

The reader shall refer to the source for implementation details.

3.1 Interface

`sd_interface.h`

Since OPaC design has been inspired by some X concepts, the structures of the library and those of Xlib are quite similar.

Some `#define` symbols are common to OPaC and Xlib. They have been redefined in the interface file.

3.2 Graphic Interface Layer

3.2.1 Drawing

`sd_grail.cxx`

Global functions

```
void
OPaC_GrailPort_Initialise (OPaC_GrPort*& info,
                           OPColorPixel f,
                           OPColorPixel b)
```

Xlib primitive used:

```
XCreateGC
```

```
void
OPaC_GrailPort_Kill (OPaC_GrPort*& info)
```

Xlib primitive used:

```
XFreeGC
```

```
void
OPaC_GrailPort_BeginDraw (OPaC_GrPort* info,
                          int x1, int y1, int x2, int y2,
                          OPaC_ColorMan* cman)
```

No special implementation.

```
void
OPaC_GrailPort_EndDraw (OPaC_GrPort* info)
```

No special implementation.

```
void
OPaC_GrailPort_Flush (OPaC_GrPort* info)
```

Xlib primitive used:

```
XFlush
```

```
void
OPaC_GrailPort_SetClipRect (OPaC_GrPort* info,
                            int x1, int y1, int x2, int y2)
```

Xlib primitive used:

```
XSetClipRectangles
```

```
void
OPaC_GrailPort_SetForeColor (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPColorManIn cman)
```

Xlib primitive used:

```
XSetForeground
```

```
void
OPaC_GrailPort_SetBackColor (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPColorManIn cman)
```

Xlib primitive used:

```
XSetBackground
```

```
void
OPaC_GrailPort_SetLineStyle (OPaC_GrPort* info,
                             OPColorPixel pixel,
                             OPLineStyle style,
                             OPColorManIn cman)
```

Xlib primitives used:

```
XSetDashes
XSetForeground
XSetLineAttributes
```

```
void
OPaC_GrailPort_SetFont (OPaC_GrPort* info,
                        OPaC_GrFont* font_info)
```

Xlib primitive used:

```
XLoadFont
XSetFont
```

Target specific methods

```
void
OPGrailPort::DrawDot (OPCoord x, OPCoord y)

void
OPGrailPort::DrawDot (OPCoord x, OPCoord y, OPColorPixel color)
```

Xlib primitive used:

```
XDrawPoint
```

```
void
OPGrailPort::DrawLineDX (OPCoord x, OPCoord y, OPCoord dx)

void
OPGrailPort::DrawLineDY (OPCoord x, OPCoord y, OPCoord dy)
```

Xlib primitive used:

```
XDrawLine
```

```
void
OPGrailPort::DrawLineToX (OPCoord x1, OPCoord y1, OPCoord x2)

void
OPGrailPort::DrawLineToY (OPCoord x1, OPCoord y1, OPCoord y2)
```

Xlib primitive used:

```
XDrawLine
```

```
void
OPGrailPort::DrawRect (OPRectIn box, OPRelCoord margin)

void
OPGrailPort::DrawRect (OPCoord x1, OPCoord y1,
                                   OPCoord x2, OPCoord y2)
```

Xlib primitive used:

```
XDrawRectangle
```

```
void
OPGrailPort::FillRect (OPRectIn box, OPRelCoord margin)

void
OPGrailPort::FillRect (OPCoord x1, OPCoord y1,
                       OPCoord x2, OPCoord y2)
```

Xlib primitive used:

`XFillRectangle`

```
void
OPGrailPort::DrawHairLine (OPCoord x, OPCoord y,
                           OPCoord dx, OPCoord dy)

void
OPGrailPort::DrawHairLineTo (OPCoord x1, OPCoord y1,
                             OPCoord x2, OPCoord y2)
```

Xlib primitive used:

`XDrawLine`

```
void
OPGrailPort::DrawAALine (OPCoord x, OPCoord y,
                         OPCoord dx,OPCoord dy)

void
OPGrailPort::DrawAALineTo (OPCoord x1, OPCoord y1,
                           OPCoord x2, OPCoord y2)
```

Not implemented. Use `XDrawLine`.

```
void
OPGrailPort::ShowText (OPCoord x, OPCoord y, OPStringIn text,
                       Count start, Count length,
                       Card8 pos,
                       OPCoord dx)

void
OPGrailPort::ShowText (OPRectIn box, OPStringIn text,
                       Count start, Count length,
                       Card8 pos, OPRelCoord margin)

void
```

```
OPGrailPort::ShowText (OPCoord x1, OPCoord y1,
                       OPCoord x2, OPCoord y2,
                       OPStringIn text,
                       Count start, Count length,
                       Card8 pos)
```

Xlib primitives used:

```
XQueryTextExtents
XDrawString
```

First queries the X server for the geometry of the string to display in the current font to align it according to the style wanted.
The first method do the actual system calls, the others call the first.

```
void
OPGrailPort::CopyRect (OPGrailPortIn source,
                       OPCoord source_x, OPCoord source_y,
                       OPCoord dest_x1, OPCoord dest_y1,
                       OPCoord dest_x2, OPCoord dest_y2)

OPGrailPort::CopyRect (OPGrailPortIn source,
                       OPCoord source_x, OPCoord source_y,
                       OPRectIn dest_box, OPRelCoord dest_margin)
```

Xlib primitives used:

```
XFillRectangle
XCopyArea
XCopyPlane
XCreateGC
XCreatePixmap
XFreeGC
XFreePixmap
```

`XCopyArea` performs a copy without transparency. For transparency support, the following process has been used:

- Create a colormask pixmap that is initially filled with the transparent color
- Merge the source to the colormask in order to an image where transparent pixels are white and others are not white
- Create a clipmask (pixmap of only one bitplane) from the colormask by merging the planes. The result is a bitmap with white pixels for the transparent color and black for the others
- Invert the clipmask
- Initialize a temporary pixmap identical to the destination
- Copy the source to a temporary pixmap using the clipmask

- Eventually copy the temporary pixmap to the destination

The use of the temporary pixmap is needed to hold the clipmask of the destination to its original shape.

```
void
OPGrailPort::SetColorPixels (OPCoord x, OPCoord y, OPCoord dx,
                             const OPColorPixel* pixels)

void
OPGrailPort::SetGrayPixels (OPCoord x, OPCoord y, OPCoord dx,
                            const Card8* intensity)

void
OPGrailPort::SetRGBPixels (OPCoord x, OPCoord y, OPCoord dx,
                           const Card8* red,
                           const Card8* green,
                           const Card8* blue)
```

Xlib primitives used for `SetColorPixels`:

```
XSetForeground
XDrawPoint
```

Xlib primitives used for `SetRGBPixels`:

```
XAllocColor
XPutPixel
XPutImage
XDestroyImage
```

`SetGrayPixels` calls `SetRGPPixels`
`SetRGBPixels` writes into an `XImage` structure that is local to the client. It only allocates colors once by storing them in an array. This is efficient for pixel lines that use a few colors. When the image is drawn, it is sent in one request to the server.

3.2.2 Pixmap

`sd_grailpxm.cxx`

```
OPGrailPixmap::OPGrailPixmap ()
```

No special implementation.

```
OPGrailPixmap::~OPGrailPixmap ()
```

Xlib primitive used:

```
XFreePixmap
```

```
void
OPGrailPixmap::BeginDraw (OPRectIn box, OPRelCoord margin)

void
OPGrailPixmap::BeginDraw (OPCoord x1, OPCoord y1,
                          OPCoord x2, OPCoord y2)
```

No special implementation.

```
void
OPGrailPixmap::EndDraw ()
```

No special implementation.

```
void
OPGrailPixmap::Initialise (OPGrailPixmapIn window,
                           OPCoord dx,
                           OPCoord dy)

void
OPGrailPixmap::Initialise (Count depth, OPCoord dx, OPCoord dy)
```

Xlib primitives used:

```
XCopyGC
XCreatePixmap
XDefaultDepth
```

```
Bool
OPGrailPixmap::SystemGetRGBPixels (OPCoord x, OPCoord y,
                                   OPCoord dx,
                                   Card8* r, Card8* g, Card8* b)
```

Xlib primitives used:

```
XGetImage
XDestroyImage
XQueryColors
```

Fetch the whole line in one request by using an `XImage` structure, get the color values in one call, then traverse the color array to fill the given arrays.

3.2.3 Window

sd_grailwdo.cxx

```
OPGrailWindow::OPGrailWindow ()
```

```
OPGrailWindow::~OPGrailWindow ()
```

Xlib primitives used:

```
XDestroyWindow
XWindowEvent
```

Destroy the window, then a loop wait for all the pending events associated with the window up to the notification insuring that the window has been destroyed. This work is done to avoid receiving events associated with the window after it was destroyed.

```
void
OPGrailWindow::Initialise (OPWindowIn view,
                           OPWindowStyle style,
                           OPStringIn name,
                           OPRectIn box, OPRelCoord margin)

void
OPGrailWindow::Initialise (OPWindowIn view,
                           OPWindowStyle style, OPStringIn name,
                           OPCoord x1, OPCoord y1,
                           OPCoord x2, OPCoord y2)
```

Xlib primitives used:

```
XCreateSimpleWindow
XSetWMSizeHints
XSetWMProtocols
XChangeWindowAttributes
XStoreName
XChangeProperty
XSelectInput
```

The window is created and its size set according to the given window style. The name is stored as well as the id and a numeric value that is the adress of the `OPGrailWindow` structure. It is stored as a string to support architectures that have different adress spaces. These informations are stored as window properties, in X jargon. They are accessible via atoms. Later, the presence of the properties can be checked to determine if a window is an OPaC window.

An event mask is set for the new window, it specifies what kind of events are sent for this window.

```
void
OPGrailWindow::ChangeName (OPStringIn name)
```

Xlib primitive used:

```
XStoreName
```

```
void
OPGrailWindow::Open ()
```

Xlib primitive used:

```
XMapWindow
```

```
void
OPGrailWindow::Reduce ()
```

Xlib primitive used:

```
IconifyWindow
```

```
void
OPGrailWindow::Close ()
```

Xlib primitive used:

```
XUnmapWindow
```

```
void
OPGrailWindow::ActivateAsFrontWindow (Bool front)
```

Xlib primitive used:

```
XSetInputFocus
```

```
void
OPGrailWindow::SysChangedActive (Bool yes)

void
OPGrailWindow::SysChangedOpen (Bool yes)

void
OPGrailWindow::SysChangedVisible (Bool yes)

void
OPGrailWindow::SysChangedReduced (Bool yes)
```

No special implementation.

```
Bool
OPGrailWindow::IsActive () const
```

No special implementation.

```
void
OPGrailWindow::SetOrigin (OPCoord x, OPCoord y)
```

Xlib primitive used:

```
XMoveWindow
```

```
void
OPGrailWindow::GetOrigin (OPCoordOut x, OPCoordOut y) const
```

No special implementation.

```
void
OPGrailWindow::SetSize (OPCoord dx, OPCoord dy)
```

Xlib primitive used:

```
XResizeWindow
```

```
void
OPGrailWindow::GetSize (OPCoordOut dx, OPCoordOut dy) const
```

No special implementation.

```
void
OPGrailWindow::UserChangedGeometry ()
```

Xlib primitives used:

```
XGetGeometry
XTranslateCoordinates
```

`XTranslateCoordinates` is used to get the absolute coordinates of the window.

```
void
OPGrailWindow::GetFrameSize (OPCoordOut xl, OPCoordOut yt,
                             OPCoordOut xr, OPCoordOut yb)


void
OPGrailWindow::GetFrameSize (OPWindowStyle style,
                             OPCoordOut xl, OPCoordOut yt,
                             OPCoordOut xr, OPCoordOut yb)
```

Not implemented. X window can't give the decoration size of the window manager without specifying a window. The static method can't be implemented. The rest of the implementation is done in such a way that this it's not a constraint.

```
void
OPGrailWindow::BeginDraw (OPRectIn box, OPRelCoord margin)

void
OPGrailWindow::BeginDraw (OPCoord x1, OPCoord y1,
                          OPCoord x2, OPCoord y2)
```

Stores the rectangle coordinates in order to optimize the update done by `EndDraw`.

```
void
OPGrailWindow::EndDraw ()
```

Just call `Flush`.

```
void
OPGrailWindow::Flush ()
```

Xlib primitives used:

```
XCopyArea
XFlush
```

Copy the rectangle modified between `BeginDraw` and `EndDraw`.

```
void
OPGrailWindow::Redraw ()
```

Xlib primitives used:

```
XCopyGC
XSetClipRectangles
XCopyArea
```

Set the clip mask to the entire window and copy the whole pixmap. A temporary graphic context is created to hold the original clipmask.

```
Bool
OPGrailWindow::FindOwnWindow (OPCoord x, OPCoord y,
                              OPGrailWindowOut wdo,
                              Count index)
```

Xlib primitives used:

```
XTranslateCoordinates
XGetWindowProperty
XQueryTree
XGetWindowAttributes
XGetGeometry
```

Establish a list of all children of the root window, then traverse this list to find which childs contain the given coordinate. Find the nth child and descent to the most nested window, check if it is an OPaC window.

To improve performance, the topmost window is checked before the list traversing. If it is the root window or a non-OPaC window, the method exits immediately.

3.2.4 Font

`sd_grailfnt.cxx`

```
OPGrailFont::OPGrailFont ()
```

No special implementation.

```
OPGrailFont::~OPGrailFont ()
```

No special implementation.

```
void
OPGrailFont::StoreInstance (OPStorageIn storage)
```

No special implementation.

```
void
OPGrailFont::RestoreInstance (OPStorageIn storage)
```

No special implementation.

```
Bool
OPGrailFont::Initialise (OPStringIn name,
                         OPStringIn style, OPCoord size)
```

No special implementation.

```
Bool
OPGrailFont::RealizeFont ()
```

Xlib primitive used:

```
XLoadFont
```

Since the font manager is not implemented, this method always realize the `fixed` font.

```
void
OPGrailFont::GetFontName (OPStringOut name) const

void
OPGrailFont::GetFontStyle (OPStringOut name) const

OPCoord
OPGrailFont::ReturnFontSize () const
```

No special implementation.

```
OPRelCoord
OPGrailFont::ReturnAscender () const

OPRelCoord
OPGrailFont::ReturnDescender () const
```

Xlib primitive used:

```
XQueryTextExtents
```

```
OPCoord
OPGrailFont::ReturnWidth (const LatinChar* text) const

OPCoord
OPGrailFont::ReturnWidth (OPStringIn text) const

OPCoord
OPGrailFont::ReturnWidth (OPStringIn text,
                          Count start, Count length) const
```

Xlib primitive used:

```
XQueryTextExtents
```

3.2.5 Mouse Cursor

`sd_grailcur.cxx`

```
OPGrailCursor::OPGrailCursor ()
```

No special implementation.

```
OPGrailCursor::~OPGrailCursor ()
```

No special implementation.

```
void
OPGrailCursor::Initialise (OPCoord hotx, OPCoord hoty,
                           OPCoord dx, OPCoord dy,
                           OPGrailPixmapIn fg,
                           OPCoord xf, OPCoord yf,
                           OPGrailPixmapIn bg,
                           OPCoord xb, OPCoord yb)
```

Xlib primitives used:

```
XCreatePixmap
XCreateGC
XCopyPlane
XCreatePixmapCursor
XFreeGC
XFreePixmap
```

Two bitmaps (pixmaps of depth 1) are created and passed to `XCreatePixmapCursor`.

```
void
OPGrailCursor::Activate () const
```

Xlib primitive used:

```
XDefineCursor
```

```
void
OPGrailCursor::ResetToNormal () const
```

Xlib primitive used:

```
XUndefineCursor
```

```
void
OPGrailCursor::Hide () const
```

Xlib primitive used:

```
XDefineCursor
```

3.3 Event Handling

`sd_event.cxx`

3.3.1 Global functions

Some global functions have been defined in this file to help event handling. They are called from `WaitForEvent`:

```
static OPGrailWindow*
ReturnOPwinFromXwin(Window xwindow)
```

Given an X window id, return a pointer to the associated `OPGrailWindow` structure. This is done using the window property storing the pointer (see page 38).

```
void
SetOPEventState(Card32& buttons,
                Card32& modifiers,
                unsigned int state)
```

Fills in the `buttons` and `modifiers` fields according to state field of X event.

```
void
HandleKeyEvent( OPKeyOut key, OPAtomRef name, XKeyEvent& xkey)
```

Fills in the OPKey structure and the name of key used according to the X key event.
This is realized with a large `switch` statement inspecting the type of the X key.

3.3.2 Methods

```
void
OPEvent::HandOverToWindow (OPWindowIn window)
```

Xlib primitive used:

```
XGrabPointer
```

```
OPEvent::OPEvent ()
```

Initialise the file descriptor and the flags for the `select` function implementing the timer (see the `WaitForEvent` method.

```
OPEvent::~OPEvent ()
```

No special implementation.

```
Bool
OPEvent::WaitForEvent ()
```

Xlib primitives used:

```
XPending
XNextEvent
```

This method uses the system function `select` with the file descriptor of the input event stream. This allows to wait for an event for a given time. This implements a timer; if the functions returns because of a timeout, a timer tick is sent to the application, otherwise an user event occured and has to be treated. This gives the priority to user events but leave enough time for the timer ticks.

The user events are handled in a `switch` statement, the associated window structure is fetch via `ReturnOPwinFromXwin`.

Exposure event Call `Redraw` for the concerned window. Flushes the eventual other exposure event from the queue.

Mouse events Use the `SetOPEventState` function to fill the `button` and `modifiers` fields.

When a motion event is received, all the pending motion event are flushed from the queue. This improves performance.

Keyboard events Use the `SetOPEventState` function to fill the `button` and `modifiers` fields. Call `HandleKeyEvent`.

Configure notify This event occurs when the user or the window manager changes the geometry of the window. `UserChangedGeometry` is called for the window.

Focus events Call the `SysChangedActive` method of the window when the focus is got or lost.

Client message When a window is destroyed by the window manager or the user, such an event occurs. If the `data` attribute of the event structure corresponds to a certain atom, the `PostQuitEvent` method is called. All other client messages are ignored.

```
void
OPEvent::DispatchEvent()
```

No special implementation.

```
void
OPEvent::PostQuitEvent ()
```

Set to `TRUE` a global variable that is treated in the `WaitForEvent` method to exit the event loop.

```
void
OPEvent::Initialise ()
```

No special implementation.

```
void
OPEvent::Kill ()
```

No special implementation.

3.4 Color Manager

`sd_colman.cxx`

```
OPColorMan::OPColorMan ()
```

No special implementation.

```
OPColorMan::~OPColorMan ()
```

No special implementation.

```
OPColorPixel
OPColorMan::MakeTrueColor (Card8 r, Card8 g, Card8 b) const
```

Xlib primitive used:

```
XAllocColor
```

```
Bool
OPColorMan::FindColor (OPColorPixel p,
                       Card8Out r, Card8Out g, Card8Out b) const
```

Xlib primitive used:

```
XQueryColor
```

3.5 Font Manager

`sd_fontman.cxx`

3.6 Memory

```
Bool
SysMemOpenZone (const LatinChar* name,
                Card32Out zone_id)
```

No special implementation.

```
Bool
SysMemCloseZone (Card32In zone_id)
```

No special implementation.

```
Bool
SysMemAlloc (Card32In zone_id, Card32In size,
             BytePtrOut ptr,
             Card32Out allocated,
             Card32Out mem_id)
```

Call to `malloc`.

```
Bool
SysMemFree (Card32In zone_id,
            BytePtrIn ptr,
            Card32In allocated,
            Card32In mem_id)
```

Call to `free`.

3.7 File Input/Output

`sd_file_io.cxx`

```
OPaC_FileDesc
OPaC_OpenFileRead (OPStringIn name)
```

Call to system function `open`.

```
OPaC_FileDesc
OPaC_OpenFileWrite (OPStringIn name)
```

Call to system function `open`.

```
OPaC_FileDesc
OPaC_OpenFileAppend (OPStringIn name)
```

Call to system function `open`.

```
Size
OPaC_ReadFile (OPaC_FileDesc file, void* void_buffer, Size size)
```

Call to system function `read`.

```
Size
OPaC_WriteFile (OPaC_FileDesc file,
                const void* void_buffer, Size size)
```

Call to system function `write`.

```
Bool
OPaC_SeekFileAbs (OPaC_FileDesc file, Int32 offset)
```

Call to system function `lseek`.

```
Bool
OPaC_SeekFileRel (OPaC_FileDesc file, Int32 offset)
```

Call to system function `lseek`.

```
Size
OPaC_TellFilePos (OPaC_FileDesc file)
```

Call to system function `lseek`.

```
Size
OPaC_TellFileSize (OPaC_FileDesc file)
```

Call to system function `llseek`.

```
Bool
OPaC_FlushFile (OPaC_FileDesc file)
```

Call to system function `fsync`.

```
Bool
OPaC_TruncFile (OPaC_FileDesc file)
```

Call to system functions `lseek` and `ftruncate`.

```
Bool
OPaC_ResizeFile (OPaC_FileDesc file, Size size)
```

Call to system function `ftruncate`.

```
Bool
OPaC_CloseFile (OPaC_FileDesc file)
```

Call to system function `close`.

3.8 Startup

`sd_main.cxx`

The pointer to the display structure is declared in this file. The `main` function calls `XOpenDisplay` an `XCloseDisplay`.

X window allows to open windows on different screens and hosts. This has not been used in the implementation, since this doesn't matches OPaC portability philosophy.

Chapter 4

Concluding with limitations

The port of the library showed that the time of software development is mostly spent in tests and debugging. Since the features were not too complex, the coding was not a long work. The asynchronous nature of X window leads to some of tricky debugging situations. Hopefully, every problem was found a solution but sometimes requiring a long investigation. The "spiral" approach discussed in the introduction allowed to get a satisfying result. Though, some missing features need to be implemented and some others have to be modified for better performance. Some bugs and limitations are yet obvious at this stage of work:

- The bitplane operations used in `OPGrailCursor::Initialise` and in `OPGrailPort::CopyRect` don't give the expected results on some X servers.
- The startup takes too much time. Some initialisation stuff could be optimized to reduce the number of queries to the server.
- The indexed color displays are not supported. As they tend to disappear, they may never need to be supported...
- The font management is not implemented. For a fully portable system, OPaC should use a system independent font package.

Inspite of these misgivings, OPaC is beginning to conquer open Unix world. With some time spent in improving and supporting the library, the graphical interface standards will have a new fellow, a dynamic one.

Laurent Bovet

February 9, 1998
Lausanne, Switzerland

Bibliography

[1] A. NYE, *Xlib Programming Manual*, O'Reilly & Associates Inc., USA, 1992.

[2] A. NYE, *Xlib Reference Manual*, Third Edition, O'Reilly & Associates Inc., USA, 1992.

Index