Open Watcom Linux Port
GUI Software Requirements Specification

Created on 1/15/2004 2:01 PM

Table of Contents

GUI Software Requirements Specification

Executive Summary

This document describes a detailed approach to porting low-level Open Watcom GUI library to Linux platform using GTK toolkit for the X Window System.

1. Introduction

1.1 About this document

This paper represents a result of the Open Watcom low level GUI library research. It provides porting guidelines and identifies the effort required to port the GUI library to Linux platform using GTK windowing toolkit for the X Window System. The research was based on the Open Watcom version 1.1.7

This paper is intended as a base Software Requirements Specification for the Open Watcom GUI porting project.

1.2 Document structure

This document consists of several parts.

"GTK Overview" provides a short overview of the GIMP Toolkit

"General GUI Porting Approach" describes a porting approach.

"Issues" identifies a set of possible caveats that should be considered before commencing the porting effort.

"Target System Requirements" provides information about required software on a target system for the ported library to run.

"Detailed Porting Guidelines and Estimation" provides detailed guidelines on porting each library function.

2. GTK Overview

GTK (GIMP Toolkit) is a library for creating graphical user interfaces. It is licensed using the LGPL license, so open software, free software, or commercial non-free software can be developed using GTK without having to spend costs on licenses or royalties.

It's called the GIMP toolkit because it was originally written for developing the GNU Image Manipulation Program (GIMP), but GTK has now been used in a large number of software projects, including the GNU Network Object Model Environment (GNOME) project. GTK is built on top of GDK (GIMP Drawing Kit) which is basically a wrapper around the low-level functions for accessing the underlying windowing functions (Xlib in the case of the X windows system), and gdk-pixbuf, a library for client-side image manipulation.

GTK is essentially an object oriented application 'programmers interface (API). Although written completely in C, it is implemented using the idea of classes and callback functions (pointers to functions).

There is also a third component called GLib which contains a few replacements for some standard calls, as well as some additional functions for handling linked lists, etc. The replacement functions are used to increase GTK's portability, as some of the functions implemented here are not available or are non-standard on other Unixes (one example being g_strerror)(). Some also contain enhancements to the libc versions, such as g_malloc() that has enhanced debugging utilities.

In version 2.0, GLib has picked up the type system which forms the foundation for GTK's class hierarchy, the signal system which is used throughout GTK, a thread API which abstracts the different native thread APIs of the various platforms and a facility for loading modules.

As the last component, GTK uses the Pango library for internationalized text output.

Why use GTK?

It is:

- Stable,
- Free,
- Fast,
- Well documented,
- Broadly adopted,
- Themes support,
- Extensible.

3. General GUI Porting Approach

The Watcom GUI library depends on the Watcom Programming Interface (WPI) and Memory Tracker (TrMem) libraries. WPI is explicitly designed to port Windows functionality to OS/2 Presentation Manager (not the other way around). Both Windows and OS/2 APIs have similar and compatible designs to a large extent. GTK API is similar neither to Windows nor to OS/2 API, so it would be ineffective to extend the WPI library with GTK support. However taking into account that the Watcom GUI library actively uses WPI calls it seems reasonable to port most of the WPI functions called in the library.

Since the GUI library relies on OS/2 PM and Windows API, it uses predefined types that exist only in these environments, such as `HWND` and `HBRUSH`. It will be an additional work for the programmer to add appropriate type definitions in a special file. For instance, `HWND` should be defined as `GtkWidget*`, and `HBRUSH` should be defined as `GdkStyle*`.

As long as in Windows and OS/2 environments user interface controls are placed according to the specified coordinates we must replicate this in GTK using the following widget hierarchy:

Window – Vertical Box – Scrolled Window – View Port – Fixed

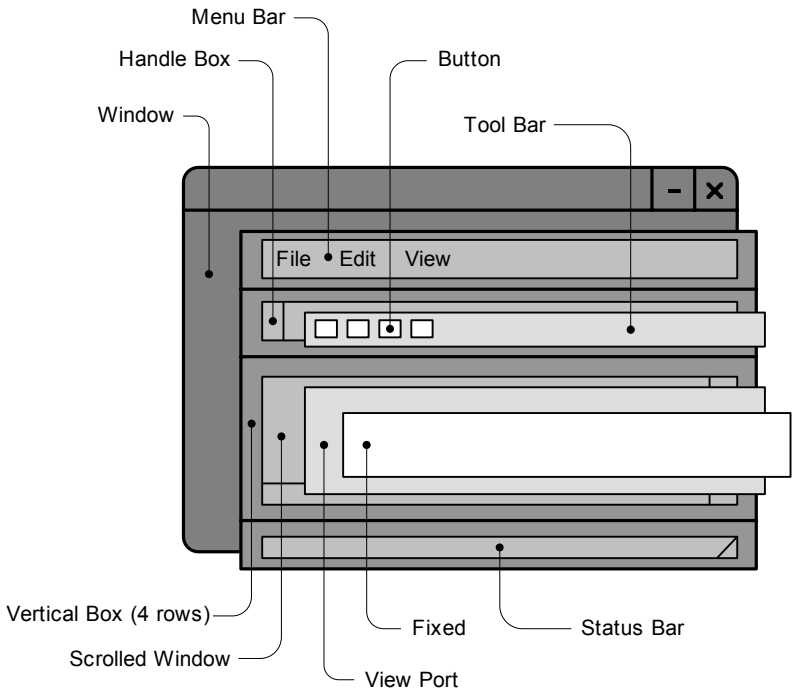

Fig.1 Widget hierarchy for a standard Open Watcom GUI window

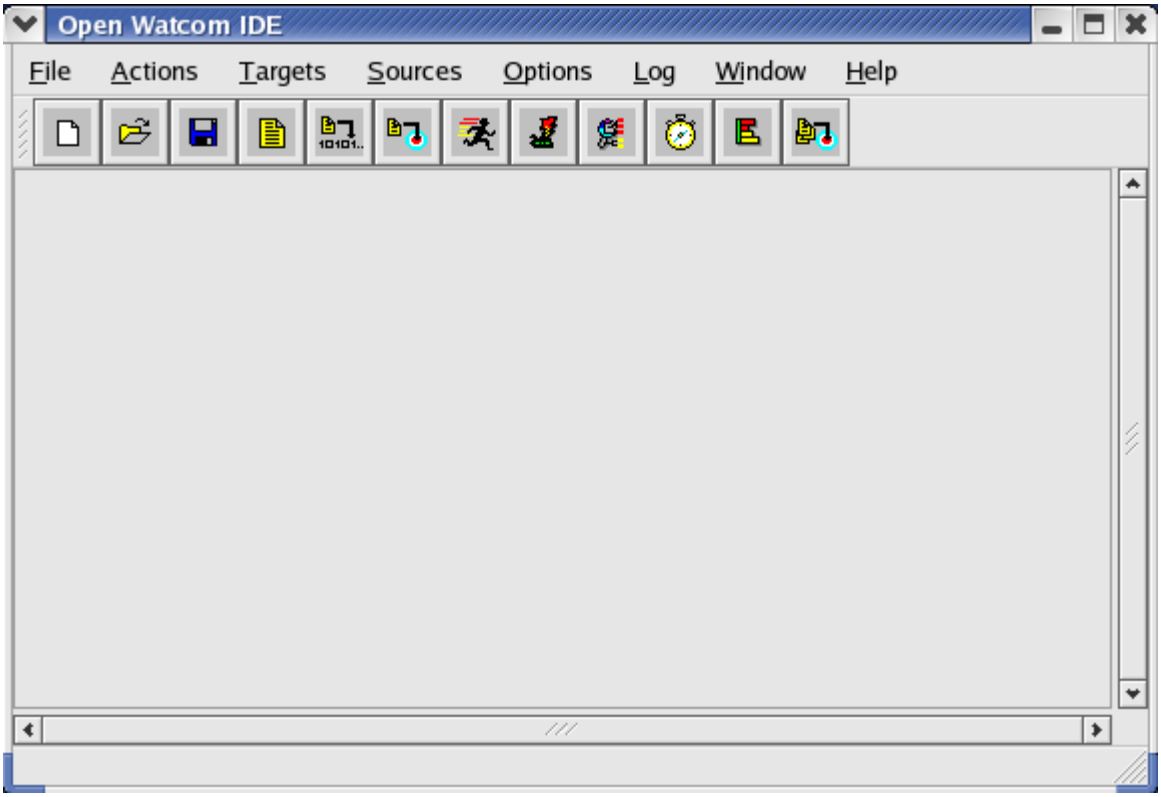

Fig.2 Open Watcom IDE GTK prototype.

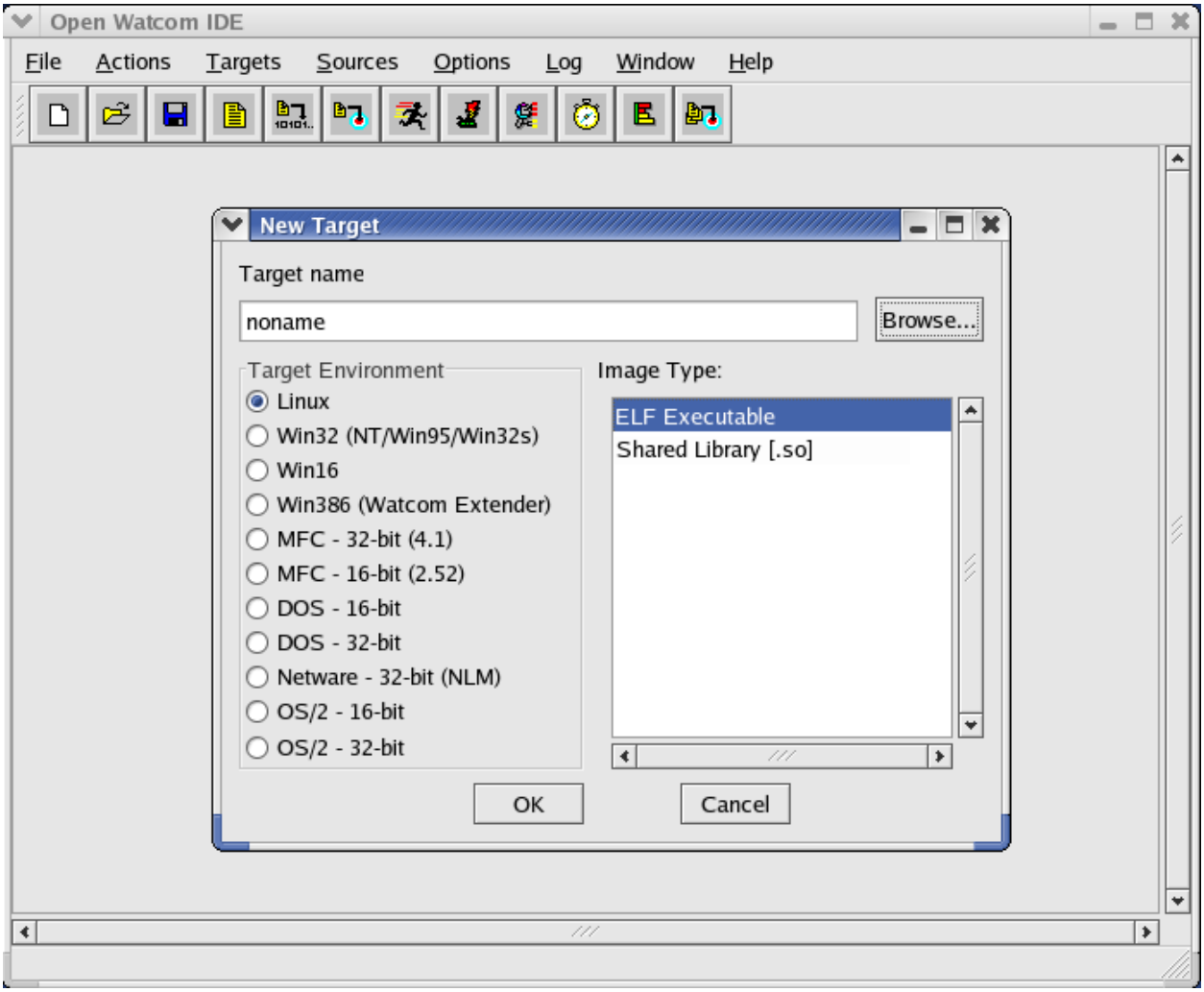

Fig 3. Open Watcom IDE GTK prototype

4. Issues

Due to the fact that Open Watcom itself is heavy oriented towards Windows, several issues should be considered and dealt with before commencing the porting effort.

- Resource files

There is no possibility in GTK to handle the Windows or OS/2 – style resource files. GTK uses its own resource files which are very much different from those in Windows or OS/2 and allows to specify only styles and key bindings of the widgets. There is a possibility to overcome this issue to some extent by using `libglade` library which is used in GTK to build user interface described in the XML file. A utility can be written to extract all the necessary information from the Windows – OS/2 resource files and present it in the XML format which can be handled by `libglade` to build a user interface. However this will provide only a partial solution as there is nothing like `String Table` in `libglade` as string resources are handled by `gettext` library in GTK. Also there are no numeric resource identifiers in GTK. Problematic functions are: `GUICreateResDialog()`, `GUICreateDialogFromRes()` and `GUILoadString()`.

- MDI "Windows in Window" model

GTK has no possibility to implement applications with MDI "Windows in Window" model. However, it is possible to assign a parent for windows, and they will be destroyed when the parent is closed.

- Help subsystem

GTK has no built in user help subsystem, as this level of user interaction is handled by desktop environments. Problematic functions here are: `GUIHelpInit()`, `GUIHelpFini()`, `GUIShowHelp()`, `GUIDisplayHelp()`, `GUIDisplayHelpWin4()` and `GUIDisplayHelpId()`.

Other issues to consider include:

- Numeric identifiers of widgets.
- Specific approach to window building includes Window Classes and Dialog Templates.
- The toolkit allows modifying window (system) menu, which is impossible in GTK.

5. Target System Requirements

GTK is broadly adopted among the major Linux distribution vendors including RedHat, SuSE, TurboLinux. So, in general case deployment of the ported library should not be a problem. In case the target distribution does not support GTK at all or uses incompatible or older versions of the toolkit the library can be linked statically, the drawback of this approach will be an increased size of the library.

In situations when the library would be shipped in source files and then compiled, target system will have to satisfy to a number of requirements. Namely it must have a C compiler and an X Window System including the following libraries:

- pkg-config (only for compilation)
- GNU make (only for compilation)
- JPEG, PNG and TIFF image libraries
- FreeType
- fontconfig
- GNU libiconv library
- GNU gettext
- GLib
- Pango
- ATK
- GTK+

Detailed Porting Guidelines and Estimation

Initial Stage

6. GUI Library Initialization.

GUI library initialization procedure starts from `GUIXMain()` function called from predefined in the library `main()`.

Below is the list of subsequent function calls originated from `GUIXMain()`.

```
GUIXMain()
 GUIStoreArgs()
  GUISetWindowClassName()
   GUIGetWindowClassName()
 _wpi_setwpiinst()
 GUIMemOpen()
 GUIFirstCrack()
 SetupClass()
 GUILoadStrInit()
  GUIGetExtName()
 GUIInitInternalStringTable()
 GUIInitGUIMenuHint()
 GUIGetFront()
 GUIWinMessageLoop()
 GUICleanup()
  GUIDeath()
  GUICleanupHotSpots()
  GUIFreeStatus()
  GUIFiniInternalStringTable()
  GUILoadStrFini()
  GUISysFini()
  GUIFiniDialog()
   GUIFree()
 GUIDead()
 GUIMemClose()
```

Majority of these functions does not need to be changed, `GUISetWindowClassName()` and `GUIGetWindowClassName()` is not needed for GTK port at all, since GTK has no window classes.

The following functions should be ported:

`GUIXMain()` – 0.5 hour(s)
`SetupClass()` – 0.5 hour(s)
`_wpi_postquitmessage()` – 0.5 hour(s)
`GUILoadStrInit()` – 0.5 hour(s)
`GUIWinMessageLoop()` – 1 hour(s)

GUIXMain

The only function called form the `main()`. It initializes program GUI, processes message queue and finishes the program.

Time needed for porting: 0.5 hours.

SetupClass

Registers a window class.

There are no window classes in GTK, so this function should be empty for the GTK port.

Time needed for porting: 0.5 hours.

_wpi_postquitmessage

Post `QUIT` message into the program.

This function should be empty in GTK port since there is no need to emit `QUIT` message if message loop is not started.

Time needed for porting: 0.5 hours.

GUILoadStrInit

Load external resource DLL under Windows, and set `GUIMsgInitFlag`.

Since there is no "external resource DLLs" in Linux, the GTK port should do the same that OS/2 port does, namely, just set `GUIMsgInitFlag`.

Time needed for porting: 0.5 hours.

GUIWinMessageLoop

Starts the main program loop.

In GTK program message loop strats with `gtk_main()`.

Time needed for porting: 1 hour.

Summary

3 hours are needed for porting and, additionally, approximately 3 more hours are needed for testing and tuning, totaling to 6 hours for this part.

At the completion of this step we will be able to execute the following code.

```
int

GUImain() {}
```

7. Display window initialization

In order to initialize the library, programmer should call `GUIWndInit()` function.

Below is the list of subsequent function calls originated from `GUIWndInit()`.

```
GUIWndInit()
 GUISysInit()
 _wpi_setdoubleclicktime()
 GUISetScreen()
 GUIInitDialog()
  GUIStrDup()
   GUIAlloc()
  DialogTemplate()
  DoneAddingControls()
  DynamicDialogBox()
  GUIInitDialogFunc()
   _wpi_enddialog()
```

The following functions should be ported:

`GUIWndInit()` – 1 hour(s)
`_wpi_setdouibleclicktime()` – 0.5 hour(s)
`GUIInitDialogFunc()` – 4 hour(s)
`_wpi_enddialog()` – 1 hour(s)

GUIWndInit

High level function for the library initialization.

It is needed to use `gdk_screen_get_width()` and `gdk_screen_get_height()` applied to the result of `gdk_screen_get_default()` in order to retrieve screen width and high, instead of `_wpi_get_systemmetric()`.

Time needed for porting: 1 hour(s).

_wpi_setdouibleclicktime

Sets the double click time.

Programmer should use `gdk_set_double_click_time()`

Time needed for porting: 0.5 hour(s).

GUIInitDialogFunc

Callback function the test dialog box used to get the dialog box font and client size information.

In GTK it is possible to get the dialog box font from the style returned by `gtk_widget_get_default_style()`.

Instead of `_wpi_getclientrect()`, `_wpi_getwidthrect()` and `_wpi_getheightrect()` functions the `gtk_widget_get_request_size()` should be called.

Time needed for porting: 4 hour(s).

_wpi_enddialog

This function is used to dismiss a dialog. Note, that this does not destroy the dialog by default but only hides the dialog.

The similar behaviour could be achieved in GTK by emitting "`delete_event`" to the given window. `g_signal_emit()` or `g_signal_emit_by_name()` should be used.

Time needed for porting: 1 hour(s).

Summary

6.5 hours are needed for porting activity and, additionally, approximately 3.5 hours are needed for testing and tuning. This step requires 10 hours. At the end of this step we will be able to execute the following code.

```
int
GUImain() {
    GUIWndInit(250);
}
```

8. Window creation

This step requires Resource Files handling implemented (see "Issues" section).

Below is the list of subsequent function calls originated from `GUICreateWindow()`.

```
GUICreateWindow()
 GUISetupWnd()
  GUIXSetupWnd()
 GUIXCreateWindow()
  GUISetupStruct()
   GUICalcLocation()
    GUIScaleToScreen()
     GUIConvertRect()
      GUIConvert()
       GUIMulDiv()
    GUIScaleToScreenR()
     GUIConvertRect()
      GUIConvert()
       GUIMulDiv()
    _wpi_getclientrect()
    _wpi_getheightrect()
    _wpi_cvtc_y_size_plus1()
     _wpi_cvtc_y()
      _wpi_getclientrect()
      _wpi_getheightrect()
   GUISetColours()
```

```
 GUIXSetColours()
  SetBKBrush()
   GUIGetRGB()
    FillInRGB()
     GetRValue()
     GetGValue()
     GetBValue()
     GUIRGB()
   _wpi_createsolidbrush()
 _wpi_loadmenu()
 GUICreateMenus()
  _wpi_createmenu()
  AppendMenus()
   GUICreateSubMenu()
   GUICreateMenuFlags()
   _wpi_appendmenu()
    AppendMenu()
    InsertMenu()
   InsertPopup()
    GetPopupHMENU()
     GetParentOffset()
     _wpi_getmenuitemcount()
     _wpi_getsubmenu()
    GUIGetHMENU()
     _wpi_get_systemmenu()
     _wpi_get_menu()
GUIInitHint()
 GetStructNum()
 CountMenus()
 InsertHint()
GUISetGUIHint()
_wpi_create_window_ex()
GUISetRedraw()
 _wpi_setredraw()
GUIMaximizeWindow()
```

```
 GUIMinimizeWindow()
 _wpi_showwindow()
 GUIShowWindowNA()
  GUIInvalidatePaintHandles()
   GUIFreeWndPaintHandles()
GUIInitMDI()
GUIIsValidWindow()
 GUIGetFront()
 GUIGetNextWindow()
  InList()
GUIFreeWindowMemory()
```

It is impossible to implement _wpi_loadmenu() and GUICreateMenus() on GTK because GTK doesn't support OS/2-like resource files and has no possibility to modify window (system) menu.

Refer to the “Issues” section.

The following functions should be ported:

`GUIXCreateWindow()` – 4 hour(s)

`GUICalcLocation()` – 2 hour(s)

`_wpi_createsolidbrush()` – 2 hour(s)

`_wpi_create_window_ex()` – 4 hour(s)

`_wpi_setredraw()` – 1 hour(s)

`GUIMaximizeWindow()` – 1 hour(s)

`GUIMinimizeWindow()` – 1 hour(s)

`_wpi_showwindow()` – 1 hour(s)

`GUIShowWindowNA()` – 1 hour(s)

`GUIFreeWindowMemory()` – 2 hour(s)

GUIXCreateWindow

This high level function creates a window widget and applies the passed parameters to it.

Despite the fact that all the low-level functionality is performed by sub-calls, some tweaking may be needed.

Time needed for porting: 4 hours.

GUICalcLocation

This function calculates coordinates of the new window. Note that X Window System window managers are free to ignore this; most window managers ignore requests for initial window placement (using a user-defined placement algorithm instead) and honors requests only after the window has been shown.

Instead of using `_wpi_getclientrect()` to get client area coordinates of the window, programmers should use `gtk_widget_get_request_size()`.

Time needed for porting: 2 hours.

_wpi_createsolidbrush

In Windows this returns a solid brush with color. In OS/2 PM this allocates space for the object structure and sets the foreground colour for the brush.

For GTK this should create `GtkStyle` structure with the given parameters.

Time needed for porting: 2 hours.

_wpi_create_window_ex

This function returns pointer to newly created window widget with the given parameters.

When ported, this function should make a call of `gtk_window_new()`, and apply parameters passed in `info`. After the window is created, it is necessary to put `GtkScrolledWindow` widget in it and then put `GtkFixed` widget in the `scrolled window` widget. See "General Porting Approach" section.

Time needed for porting: 4 hours.

_wpi_setredraw

This function enables/disables window updates.

Should be done via `gdk_window_thaw_updates()` and `gdk_window_freeze_updates()`

Time needed for porting: 1 hour.

GUIMaximizeWindow

This function maximizes the specified window.

In order to ask the window manager to maximize the window in GTK `gtk_window_maximize()` should be called.

Time needed for porting: 1 hour.

GUIMinimizeWindow

This function minimizes the specified window.

In order to ask window manager to minimize the window in GTK, `gtk_window_minimize()` should be called.

Time needed for porting: 1 hour.

_wpi_showwindow

Shows the window according to the given state. Windows predefined states are used.

In the GTK `gtk_widget_show()` and `gtk_window_hide()` should be used.

Time needed for porting: 1 hour.

GUIShowWindowNA

Shows the window.

`gdk_window_show()` should be used

Time needed for porting: 1 hour.

GUIFreeWindowMemory

Frees all resources related to the given window.

In GTK `gtk_object_destory()` should be called which automatically will destroy all children. Some additional tweaking may be needed.

Time needed for porting: 2 hours.

Summary

19 hours are needed for porting activity and, additionally, approximately 16 more hours are needed for testing and tuning. This step requires 35 hours. At the finish of this step we will be able to execute the following code.

```
#include "gui.h"
#include "guitypes.h"

static  gui_rect        Scale           = { 0, 0, 1000, 1000 };

bool MainEventWnd( gui_window *gui, gui_event gui_ev, void *param )
{
    gui = gui;
    gui_ev = gui_ev;
    param = param;
    return( TRUE );
}

static gui_create_info Parent = {
    "Sample Application",
    { 250, 250, 500, 500 },
    GUI_HSCROLL | GUI_VSCROLL,
    GUI_GADGETS | GUI_VISIBLE,
    NULL,
    0, NULL,
    0, NULL,
    &MainEventWnd,
    NULL,
    NULL,
    0
};

static gui_create_info Child = {
    "Child Window",
    { 300, 300, 200, 200 },
    GUI_SCROLL_BOTH,
    GUI_VISIBLE+GUI_CLOSEABLE+GUI_MAXIMIZE+GUI_RESIZEABLE+GUI_MINIMIZE,
    NULL,
    0,
    NULL,
    0, NULL,
    &MainEventWnd,
    NULL,
    NULL,
    0
};

void GUImain( void )
{
    GUIMemOpen();
    GUIWndInit( 300, GUI_GMOUSE );
    GUISetScale( &Scale );
    Child.parent = GUICreateWindow( &Parent );
```

```
    GUICreateWindow( &Child );
}
```

9. Standard sample

The next objective is to run a standard Watcom GUI sample program located in `gui/sample/samp2.c`.

Below is its listing.

```
#include <string.h>
#include "gui.h"

#define TRUE 1
#define FALSE 0

#define HEIGHT 5
#define WIDTH 8

static gui_ord Width = 0;

#define NUM_TEXT 5

static char Text[][NUM_TEXT] = { {"0%"}, {"25%"}, {"50%"}, {"75%"}, {"100%"}
};
static int Strlen[NUM_TEXT] = { 2, 3, 3, 3, 4 };

static GUICALLBACK GetNewFunction;
static GUICALLBACK StatusFunction;

static gui_create_info DialogWnd = {
    "Install Program: ",
    { 20, 20, 40, 40 },
    GUI_NOSCROLL,
    GUI_VISIBLE,
    NULL,
    0,
    NULL,
    0,
    NULL,
    &GetNewFunction,
    NULL,
    0
};

static gui_rect Rect;
static int Row;
static int NumEnters = 0;

static gui_colour_set StatusColours[GUI_NUM_ATTRS+1] = {
    /* Fore              Back        */
    GUI_BLUE,          GUI_WHITE,   /* GUI_MENU_PLAIN     */
    GUI_BLUE,          GUI_WHITE,   /* GUI_MENU_STANDOUT  */
    GUI_BLUE,          GUI_WHITE,   /* GUI_BACKGROUND     */
    GUI_BLUE,          GUI_WHITE,   /* GUI_TITLE_ACTIVE   */
    GUI_GREY,          GUI_WHITE,   /* GUI_TITLE_INACTIVE */
```

```
    GUI_BLUE,           GUI_WHITE,   /* GUI_FRAME_ACTIVE   */
    GUI_GREY,           GUI_WHITE,   /* GUI_FRAME_INACTIVE */
    GUI_BRIGHT_WHITE,   GUI_MAGENTA  /* GUI_FIRST_UNUSED   */
};

static gui_create_info StatusWnd = {
    "Percent of Installation Complete",
    { 19, 70, 42, 20 },
    GUI_NOSCROLL,
    GUI_VISIBLE | GUI_DIALOG_LOOK,
    NULL,
    0,
    NULL,
    GUI_NUM_ATTRS + 1,
    &StatusColours,
    &StatusFunction,
    NULL,
    0
};

enum {
   ctr_static,
   ctr_edit,
   ctr_cancelbutton,
   ctr_okbutton
};

static gui_control_info GetNew[] = {
    { GUI_STATIC,         "Please enter install path:",     { 4, 4, 30,
HEIGHT }, NULL, GUI_NOSCROLL, GUI_NONE, ctr_static },
    { GUI_EDIT,           NULL,     { 4, 15, 30, HEIGHT }, NULL,
GUI_NOSCROLL, GUI_NONE, ctr_edit },
    { GUI_PUSH_BUTTON,   "CANCEL",  { 6, 30, WIDTH, HEIGHT }, NULL,
GUI_NOSCROLL, GUI_NONE, ctr_cancelbutton },
    { GUI_DEFPUSH_BUTTON, "OK",     { 26, 30, WIDTH, HEIGHT }, NULL,
GUI_NOSCROLL, GUI_NONE, ctr_okbutton }
};

static gui_colour_set Colours[GUI_NUM_INIT_COLOURS] =
{
    { GUI_BRIGHT_WHITE, GUI_BLUE },
    { GUI_BRIGHT_WHITE, GUI_BLUE }
};

static char * text = NULL;
static gui_message_return ret_val = GUI_RET_CANCEL;

static gui_window * Status = NULL;

/*
 * GetNewFunction - call back routine for the GetNewVal dialog
 */

static bool GetNewFunction( gui_window * gui, gui_event gui_ev, void * param
)
{
```

```
    unsigned id;

    switch( gui_ev ) {
        case GUI_INIT_DIALOG :
            ret_val = GUI_RET_CANCEL;
            break;
        case GUI_DESTROY :
            if( Status != NULL ) {
                GUIDestroyWnd( Status );
            }
            break;
        case GUI_CLICKED :
            GUI_GETID( param, id );
            switch( id ) {
                case ctr_cancelbutton :
                    GUICloseDialog( gui );
                    ret_val = GUI_RET_CANCEL;
                    break;
                case ctr_okbutton :
                    text = GUIGetText( gui, ctr_edit );
                    if( Status == NULL ) {
                        Status = GUICreateWindow( &StatusWnd );
                    } else {
                        NumEnters ++;
                        Rect.width = ( NumEnters * Width ) / 4;
                        if( NumEnters > 4 ) {
                            GUICloseDialog( gui );
                        } else {
                            GUIWndDirty( Status );
                        }
                    }
                    ret_val = GUI_RET_OK;
                    break;
            }
        break;
    }
    return( TRUE );
}

/*
 * StatusFunction - call back routine for the status window
 */

static bool StatusFunction( gui_window * gui, gui_event gui_ev, void * param
)
{
    int              i;
    int              pos;
    gui_text_metrics metrics;

    param = param;

    switch( gui_ev ) {
        case GUI_INIT_WINDOW :
            Row = GUIGetNumRows( gui ) / 2;
            GUIGetTextMetrics( gui, &metrics );
```

```
            GUIGetClientRect( gui, &Rect );
            Rect.x = 1;
            Rect.y = 1;
            Width = Rect.width - 2 * Rect.x;
            Rect.width = 0;
            for( i = 0; i < NUM_TEXT; i++ ) {
                Strlen[i] *= metrics.max.x;
            }
            break;
        case GUI_DESTROY :
            break;
        case GUI_PAINT :
            GUIDrawRect( gui, &Rect, GUI_FIRST_UNUSED );
            for( i = 0; i < NUM_TEXT; i++ ) {
                pos = ( i * Width / 4 ) - Strlen[i] + Rect.x;
                if( pos < (int)Rect.x ) {
                    pos = Rect.x;
                }
                if( ( i > NumEnters ) ||
                    ( i == 0 ) && ( NumEnters == 0 ) ) {
                    GUIDrawText( gui, &Text[i], Strlen[i], Row, pos,
                                 GUI_TITLE_ACTIVE );
                } else {
                    GUIDrawText( gui, &Text[i], Strlen[i], Row, pos,
                                 GUI_FIRST_UNUSED );
                }
            }
            break;
    }
    return( TRUE );
}

void GUImain( void )
{

    GUIMemOpen();
    GUIWndInit( 250 );
    GUICreateDialog( &DialogWnd, NUM_CONTROLS, &GetNew );
}
```

Here is a list of functions not yet described:

`GUICreateDialog()`

`GUIDestroyWnd()`

`GUIGetText()`

`GUICloseDialog()`

`GUIWndDirty()`

`GUIGetNumRows()` (doesn't need to be ported)

`GUIGetTextMetrics()`

`GUIGetClientRect()`

`GUIDrawRect()`

GUIDrawText().

Below is the list of subsequent function calls.

```
GUICreateDialog()
 CreateDlg()
  GUISetupWnd()
  GUIXCreateDialog()
   GUISetupStruct()
   GUIDoCreateResDialog()
   AdjustForFrame()
   AdjustToDialogUnits()
    ToDialogUnits()
   GUIDlgCalcLocation()
   GUISetControlStyle()
   DialogTemplate()
   AddControl()
   GUIControlInsert()
   DoneAddingControls()
   DynamicDialogBox()
   GUIDialogFunc()
  GUIFreeWindowMemory()

GUIDestroyWnd()

GUIGetText()
 GUIGetControlClass()
  GUIGetControlByID()



GUICloseDialog()
 GUISendMessage()

GUIWndDirty()
 GUIGetFront()
 GUIGetNextWindow()
 GUIGetParentFrameHWND()
```

```
GUIGetTextMetrics()
 GUIGetMetrics()
  GUIGetTheDC()
  GUIReleaseTheDC()
 GUISetMetrics()
  GUIScreenToScaleR()
   GUIConvert()

GUIGetClientRect()
 _wpi_mapwindowpoints()
 _wpi_getwrectvalues()
 _wpi_getheightrect()
 GUIClientToScaleRect()
  GUIScreenToScaleRect()
   GUIConvertRect()

GUIDrawRect
 DrawRect()
  GUIGetScrollPos()
  ...

GUIDrawText()
 GUIGetTextMetrics()
 GUIXDrawText()
  GUIDrawTextBitmapAttr()
   GUIGetFore()
   GUIGetBack()
   GUIDrawTextBitmapRGB()
   ...
```

GUIXCreateDialog

Creates a dialog with specified controls.

Instead of calling `GlobalFree()` here the window should be destroyed with `gtk_widget_destroy()` Some tweaking may be needed.

Time needed for porting: 1 hour.

GUIDoCreateResDialog

Creates dialog defined in resource file.

Consider Issues section.

DialogTemplate

Creates a dialog template.

There are no dialog templates in GTK. However, Open Watcom GUI Library adds controls to dialog templates, so we will create a dialog in this function. `gtk_window_create()` is needed to create a dialog.

Time needed for porting: 2 hours.

AddControl

Adds control to the specified dialog template.

There is big amount of work here. This function should create all necessary controls, and apply specified attributes for every requested control.

Depending on type of the control and attributes various GTK functions should be used.

Time needed for porting: 8 hours.

DoneAddingControls

Called when there are no more controls.

Should be empty in GTK port.

Time needed for porting: 0.5 hour.

DynamicDialogBox

Creates a dynamic dialog box.

Should do nothing in GTK port, since `DialogTemplate()` function created the dialog.

Time needed for porting: 0.5 hour.

GUIDialogFunc

Callback function for all dynamically created dialogs.

Should be rewritten for GTK version. The callback should correctly process resizing "`resize-request`" and "`close`" events.

Time needed for porting: 2 hours.

GUIDestroyWnd

Destroys the given window or all windows if NULL.

In GTK port, the function should call `gtk_widget_destroy()` for the given window or for the top-level window if NULL.

Time needed for porting: 2 hours

GUIGetText

Returns a copy of the text.

Should be greatly modified in GTK port, but it is a straightforward task. Text of almost all widgets could be retrieved via `gtk_label_get_label()`, and `gtk_button_get_label()`. Note that returned string will be owned by a widget and must not be modified or freed. So, it is needed to duplicate them in this function.

Selection of `GtkTreeView` could be accessed via `gtk_tree_view_get_selection()` function.

Time needed for porting: 6 hours.

GUISendMessage

Sends a message to the specified window.

`g_signal_emit()` should be used for this purpose.

Time needed for porting: 1 hour.

GUIWndDirty

Tells the user interface that interface should be repainted.

`gdk_window_process_updates()` can repaint the window in GTK.

Time needed for porting: 1 hour.

GUIGetMetrics

Returns text metrics for font of the given window.

`GUIGetDC()` and `GUIReleaseDC` should not be called in GTK port. There is no need to work on so low level in GTK. `FontMetrics` of the given window could be retrieved in the following way:

```
PangoContext context = gtk_widget_get_pango_context(widget);
PangoFontMetrics metrics = pango_context_get_metrics(context,
widget->style->font_desk, pango_context_get_language(context));
g_memmove(&GUItm, metrics);
pango_font_metrics_unref(metrics);
```

Time needed for porting: 2 hours.

GUIGetClientRect

Returns scaled, depending upon current scaling settings, rectangle of the given window in absolute coordinates.

Note, that client area of the window is the area of the `GtkScrolledWindow` widget placed in the window.

Time needed for porting: 1 hour.

_wpi_mapwindowpoints

Translates coordinates relative to source widget's allocation to coordinates relative to destination widget's allocations.

The similar functionality in GTK should be achieved by `gtk_widget_translate_coordinates()`. Note that there is no definition similar to `HWND_DESKTOP` in GTK. To get the needed value (root window of a given widget) `gtk_widget_get_root_window()` should be called for a given widget.

Time needed for porting: 2 hours.

_wpi_getwrectvalues

Returns coordinates of the rectangle.

Should be rewritten using `GdkRectangle` type.

Time needed for porting: 0.5 hours.

_wpi_getheightrect

Returns height of the given rectangle.

Should be rewritten using `GdkRectangle` structure.

Time needed for porting: 0.5 hours.

DrawRect

Draws specified rectangle in the given window.

Should be rewritten. GTK paint functions applied to the `GtkFixed` widget placed in the window should be used for this purpose. This approach will allow avoiding all the sub-calls.

Time needed for porting: 4 hours.

GUIXDrawText

Draws specified text in the given window.

Should be rewritten. `pango_layout_set_text()` should be used for text rendering, and `gtk_paint_layout()` applied to `GtkFixed` widget placed in the window should be used for drawing. This approach will allow avoiding all the sub-calls.

Time needed for porting: 6 hours.

Summary

41 hours are needed for porting activity and approximately 24 more hours are required for testing and tuning. This step requires 65 hours.

Elaboration stage

10. Common control functions

There are a couple of common window control functions in the Open Watcom GUI Library. Below is the list of them with their sub-calls.

```
GUIAddControl()
GUIResizeControl()
GUIDeleteControl()
 _wpi_destroywindow()
 GUIControlDelete()
  GUIDeleteCtrlWnd()
GUIEnableControl()
 _wpi_enablewindow()
GUIIsControlEnabled()
```

`_wpi_iswindowenabled()`

`GUIGetControlRect()`

`GUIHideControl()`

`ShowControl()`

`_wpi_showwindow()`

`GUIShowControl()`

`GUIIsControlVisible()`

`_wpi_iswindowwisible()`

GUIAddControl

Adds a control to a window.

The function should create an appropriate control with appropriate styles and other attributes in the requested position. Already described `AddControl` should be used here.

Time needed for porting: 5 hours.

GUIResizeControl

Set size and location of a control (relative parent).

Instead of using `_wpi_getdlgitem()` it is needed to use `GUIGetControlByID()` because the last is portable.

Instead of using `_wpi_movewindow()` it is needed to use `gtk_fixed_move()` to move the specified control in the fixed widget of window where the control is located.

Time needed for porting: 1 hour.

_wpi_destroywindow

Destroys the given widget.

`gtk_widget_destroy()` should be called in GTK port.

Time needed for porting: 0.5 hour.

GUIEnableControl

Sets control enabled or not enabled.

Instead of using `_wpi_movewindow()` it is need to use `gtk_fixed_move()` to move the specified control in the fixed widget of window where the control is located.

Time needed for porting: 1 hour.

_wpi_enablewindow

Sets specified widget enabled or disabled.

`gtk_widget_set_sensitive()` should be called within this function.

Time needed for porting: 0.5 hour.

GUIIsControlEnabled

Checks if control is enabled.

Instead of using `_wpi_movewindow()` it is needed to use `gtk_fixed_move()` to move the specified control in the fixed widget of window where the control is located.

Time needed for porting: 1 hour.

_wpi_iswindowenabled

Checks if the given widget is enabled.

`gtk_widget_get_sensitive()` should be called within this function.

Time needed for porting: 0.5 hour.

GUIGetControlRect

Gets location of a control relative to the parent.

Instead of using `_wpi_movewindow()` it is needed to use `gtk_fixed_move()` to move the specified control in the fixed widget of window where the control is located.

Instead of using `GUIGetRelRect()` it is needed to use `g_object_get_property()` to read the coordinates of the control, and use `gtk_widget_get_size_request()` to get its size.

Time needed for porting: 2 hours.

_wpi_showwindow

Shows or hides widget depending upon the flag.

Depending upon the flag `gtk_widget_show_all()` or `gtk_widget_hide_all()` function should be called.

Time needed for porting: 1 hour.

_wpi_iswindowwisible

Checks if the widget is visible.

Property "`visible`" should be used to test if the widget is visible.

Time needed for porting: 1 hour.

Summary

13.5 hours are needed for porting activity and, additionally, approximately 13.5 hours are required for testing and tuning. This step requires about 27 hours.

11. Common text functions

Below is the list of the common text related functions.

`GUISetText()`

`GUIClearText()`

`GUISelectAll()`

`GUISetEditSelect()`

`GUIGetEditSelect()`

`GUIDlgBuffGetText()`

GUISetText

Sets the text of the given widget to the given text.

Depending on the class of the given widget `gtk_entry_set_text()` or `gtk_label_set_label()` should be used.

Time needed for porting: 4 hours.

GUIClearText

Clears the text.

`GUISetText()` should be called to perform the operation.

Time needed for porting: 0.5 hour.

GUISelectAll

Selects the text within the controls `GUI_EDIT` or `GUI_EDIT_COMBOBOX`.

Should call `gtk_editable_select()` applied to a valid `GtkEntry` widget to perform the operation.

Time needed for porting: 4 hours.

GUISetEditSelect

Selects the portion of text within controls `GUI_EDIT` or `GUI_EDIT_COMBOBOX`.

Should call `gtk_editable_select()` applied to valid `GtkEntry` widget to perform the operation.

Time needed for porting: 2 hours.

GUIGetEditSelect

Finds out the portion of the text selected within `GUI_EDIT` or `GUI_EDIT_COMBOBOX`.

Should call `gtk_editable_get_selection_bounds()` and `gtk_editable_get_chars()` applied to valid `GtkEntry` widget to perform the operation.

Time needed for porting: 4 hours.

GUIDlgBuffGetText

Gets text from the control into a buffer.

Additional investigation is needed in order to find out, whether the current memory allocation algorithm in Open Watcom GUI Library can be reused in the GTK port.

Time needed for porting: 1 hour.

Summary

15.5 hours are needed for porting activity and, additionally, approximately 8.5 hours are needed for testing and tuning. This step requires about 24 hours.

12. Special dialogs functions

Below is the list of functions related to special dialogs with their sub-calls.

```
GUIDisplayMessage
GUIGetNewVal()
 GUIDlgOpen()
  DlgOpen()
   GUIGetDlgTextMetrics()
   GUITruncToPixel()
   DlgSetSize()
    GUIGetSystemMetrics()
   DlgSetCtlSizes()
   GUICreateSysModalDialog()
   GUICreateDialog()

GUIDlgPickWithRtn()
 GUIDlgOpen()

GUIDlgPick()
 GUIDlgPickWithRtn()

GUIGetFileName()
```

GUIDisplayMessage

Displays a message and gets the response. Identical functionality could be achieved by using GTK's message dialog via `gtk_message_new()`.

Time needed for porting: 4 hours.

DlgOpen

Calls `GUICreateDilog()`, formatting locations to look good on every OS.

Some effort to make the controls look similar in Linux to all other systems will be needed.

Time needed for porting: 4 hours

GUIGetDlgTextMetrics

Gets the metrics of the dialog font.

FontMetrics of the given window could be retrieved in the following way:

```
PangoContext context = gtk_widget_get_pango_context(widget);
PangoFontMetrics metrics = pango_context_get_metrics(context,
widget->style->font_desk, pango_context_get_language(context));
g_memmove(&GUItm, metrics);
pango_font_metrics_unref(metrics);
```

Time needed for porting: 2 hours.

GUIGetSystemMetrics

Selects the text within `GUI_EDIT` or `GUI_EDIT_COMBOBOX`.

Instead of calling `_wpi_getsystemmetrics()` the function should call various GTK functions to get different values.

Time needed for porting: 10 hours.

DlgSetCtlSizes

Sets sizes of controls in `GUIDlgOpen`.

Some effort to make the controls look similar in all systems will be needed.

Time needed for porting: 2 hours.

GUIGetEditSelect

Finds out the portion of text selected within `GUI_EDIT` or `GUI_EDIT_COMBOBOX`.

Should call `gtk_editable_get_selection_bounds()` and `gtk_editable_get_chars()` applied to valid `GtkEntry` widget to perform the operation.

Time needed for porting: 4 hours.

GUIDlgBuffGetText

Gets text from the control into a buffer.

Additional research should be performed in order to find out whether Open Watcom GUI Library memory allocation algorithm can be reused in the GTK port.

Time needed for porting: 1 hour.

Summary

27 hours are needed for porting activity and approximately 44 more hours are needed for testing and tuning. This step requires about 71 hour.

13. Scrolling functions

`GUIInitHScrollCol()`

`GUIInitVScrollRow()`

`GUISetHScrollCol()`

`GUISetVScrollRow()`

`GUIGetHScrollCol()`

`GUIGetVScrollRow()`

`GUISetHScrollRangeCols()`

`GUISetVScrollRangeRows()`

`GUIGetHScrollRangeCols()`

`GUIGetVScrollRangeRows()`

`GUIDoHScroll()`

`GUIDoVScroll()`

`GUIDoHScrollClip()`

`GUIDoVScrollClip()`

`GUIInitHScroll()`

`GUIInitVScroll()`

`GUISetHScroll()`

`GUISetVScroll()`

`GUIGetHScroll()`

`GUIGetVScroll()`

`GUISetHScrollRange()`

`GUISetVScrollRange()`

`GUIGetHScrollRange()`

`GUIGetVScrollRange()`

`GUISetHScrollThumb()`

`GUISetVScrollThumb()`

Common guidelines

WGL windows of the GTK port mush have `GTKScroledWindow` in them. This should contain `GTKViewPort` that contains `GtkFixed`.

Values of the scroll bars could be set and get via `gtk_scrolled_window_set_hajustment()`, `gtk_scrolled_window_set_vajustment()`, `gtk_scrolled_window_get_hajustment()` and `gtk_scrolled_window_get_vajustment()` functions.

Since the scroll bars behavior is automatic, there is no possibility to set/get scroll bar range values. Required values could be achieved by getting width and height of the `GtkFixed` widget. Setting range of a scroll means resizing of the `GtkFixed` widget.

Page size could be obtained from size of the `GtkViewPort` widget.

Summary

16 hours are needed for the porting activity and 16 more hours are required for for testing and tuning. This step requires about 32 hours.

14. Status window functions

```
GUICreateStatusWindow()
GUICloseStatusWindow()
GUIHasStatus()
GUIDrawStatusText()
GUIClearStatusText()
GUIResizeStatusWindow()
```

Common guidelines

WGL windows of the GTK port mush have `GTKStatus` widget in them. All these functions will interact with it.

GUICreateStatusWindow

Creates a status window.

Should make the `GtkStatus` visible using `gtk_widget_show()`.

Time needed for porting: 1 hour.

GUICloseStatusWindow

Closes the status window.

Should make the `GtkStatus` invisible using `Gtk_widget_hide()`

Time needed for porting: 1 hour.

GUIDrawStatusText

Draws the text to the status window.

Should use `gtk_status_push()`.

Time needed for porting: 1 hour.

GUIResizeStatusWindow

Resizes the status window.

It is impossible to change `X` position of the `GtkStatus` in our layout; however it is never changed through all Open Watcom sources. In addition, it isn't necessary to change vertical size of the bar because it is done automatically. So, this function can be empty in the GTK port.

Time needed for porting: 1 hour.

Summary

4 hours are required for the porting activity and, additionally, 4 hours are required for testing and tuning. This step requires about 8 hours.

15. Toolbar functions

```
GUICreateFloatToolBar()
GUICreateToolBar()
GUICloseToolBar()
GUIHasToolBar()
GUIChangeToolBar()
GUIToolBarFixed()
```

Common guidelines

WGL windows of the GTK port mush have `GtkHandleBox` widget which contains `GtkToolbar` widget in itself. All these functions will interact with it.

GUICreateFloatToolBar

Does the same as `GUICreateToolbar()` which is described below.

GUICreateToolBar

Creates a tool bar.

Should make the `GtkHandleBox` visible using `gtk_widget_show()`

Time needed for porting: 1 hour.

GUICloseToolBar

Closes the status window.

Should make the `GtkHandleBox` invisible using `gtk_widget_hide()`

Time needed for porting: 1 hour.

GUIChangeToolBar

Changes the tool bar. Makes it fixed or floating.

To make the tool bar fixed it is needed to make `GtkHandleBox` invisible and set parent of the `GtkToobar` to parent of `GtkHandleBox`. To reverse the operation it is needed to make the `GtkHandleBox` visible and set parent of the `GtkToobar` to the `GtkHandleBox` widget.

Time needed for porting: 2 hours.

Summary

4 hours are needed for porting activity and, additionally, approximately 6 more hours are needed for testing and tuning. This step requires about 10 hours.

16. Menu functions

`GUIAppendMenu()`

`GUIAppendMenuByOffset()`

`GUIInsertMenu()`

`GUIEnableMenuItem()`

`GUICheckMenuItem()`

`GUISetMenuText()`

`GUISetHintText()`

`GUIDeleteMenuItem()`

`GUIResetMenus()`

`GUIEnableMDIMenus()`

`GUICreateFloatingPopup()`

`GUITrackFloatingPopup()`

`GUIGetMenuPopupCount()`

`GUIAppendMenuToPopup()`

`GUIInsertMenuToPopup()`

Common guidelines

WGL windows of the GTK port must have `GtkMenuBar` widget in them. All these functions will interact with it.

GUIAppendMenu

Appends a menu to the menu bar.

`gtk_menu_shell_append()` should be used to add the menu to `GtkMenuBar` widget. Floating status could be adjusted via “tear-off” state of the `GtkMenu`.

Additionally, the menu and all items should be added in global controls table via `GUIControlInsert()`, since GTK doesn't allow assigning ID to the menu items.

Time needed for porting: 2 hours.

GUIAppendMenuByOffset

Appends sub menu to the menu.

`gtk_menu_shell_append()` should be used to insert the menu to `GtkMenu` widget.

Additionally, the menu and all items should be added in global controls table via `GUIControlInsert`, since GTK doesn't allow assigning ID for the menu items.

Time needed for porting: 2 hours.

GUIInsertMenu

Inserts a menu into the menu bar.

`gtk_menu_shell_insert()` should be used to add the menu to `GtkMenuBar` widget. Floating status could be adjusted via tear-off state.

Additionally, the menu and all items should be added to the global controls table via `GUIControlInsert()`, since GTK doesn't allow to assign ID for the menu items.

Time needed for porting: 2 hours.

GUIEnableMenuItem

Enables/disables menu item.

`gtk_widget_set_sensitive()` applied to the appropriate `GtkMenuItem` should be used in this function. The `GtkMenuItem` widget can be obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUIEnableMenuItem

Checks/unchecks menu item.

`gtk_check_menu_item_set_active()` applied to the appropriate `GtkMenuItem` should be used in this function. The `GtkMenuItem` widget can be obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUISetMenuText

Change the text of a menu item.

`gtk_label_set_label()` applied to `GtkLabel` contained in appropriate `GtkMenuItem` should be used in this function. The `GtkMenuItem` widget can be obtained via `GUIGetControlByID()`. The `GtkLabel` widget could be obtained via `gtk_bin_get_child()`.

Time needed for porting: 2 hours.

GUISetHintText

Sets the hint for the menu item.

`gtk_tooltips_set_tip()` applied to `GtkLabel` contained in appropriate `GtkMenuItem` should be used in this function. The `GtkMenuItem` widget can be obtained via `GUIGetControlByID()`. The `GtkLabel` widget can be obtained via `gtk_bin_get_child()`.

Time needed for porting: 2 hours.

GUIDeleteMenuItem

Delete a menu item for a pull down menu.

`gtk_widget_destroy()` applied to the appropriate `GtkMenuItem` should be used in this function. The `GtkMenuItem` widget can be obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUIResetMenus

Resets the entire menu structure for a window.

The function should destroy and create new `GtkMenuBar`. And, then, build and add given menu structure.

Time needed for porting: 6 hours.

GUIEnableMDIMenus

Enables/disables the MDI menus.

Since the "Window in Window" MDI model is not possible in GTK. This function should do nothing for the GTK port.

Time needed for porting: 0.5 hour.

GUICreateFloatingPopup

Creates a floating popup menu.

Should create a `GtkMenu` widget, and call `GUITrackFloatingPopup`.

Time needed for porting: 2 hours.

GUITrackFloatingPopup

Tracks a floating popup menu.

`gtk_menu_popup()` applied to the appropriate `GtkMenu` should be used in this function. The `GtkMenu` widget can be obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUIGetMenuPopupCount

Returns number of items in the given floating popup menu.

`gtk_container_get_children()` applied to the appropriate `GtkMenu` should be used in this function to obtain a list of the items. Then `g_list_length()` should be applied to the list to obtain a number of elements in the list. The `GtkMenu` widget can be obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUIAppendMenuToPopup

Appends sub menu to the given popup menu.

`GUIAppendMenu()` should be called in this function to perform the operation.

Time needed for porting: 2 hours.

GUIInsertMenuToPopup

Inserts sub menu to the given popup menu.

`GUIAppendMenuByOffset()` should be called in this function to perform the operation.

Time needed for porting: 2 hours.

Summary

32.5 hours are needed for the porting activity and, additionally, approximately 32.5 hours are required for testing and tuning. This step requires about 65 hours.

17. Text Handling Functions

`GUISetWindowText()`

`GUIGetWindowTextLength()`

`GUIGetWindowText()`

`GUIGetRow()`

`GUIGetCol()`

`GUIGetStringPos()`

`GUIGetExtentX()`

`GUIGetExtentY()`

`GUIGetControlExtentX()`

`GUIGetControlExtentY()`

`GUIGetTextMetrics()`

`GUIGetDlgTextMetrics()`

`GUIGetPoint()`

`GUIGetRow()`, `GUIGetCol()`, `GUIGetTextMetrics()`, `GUIGetDlgTextMetrics()` and `GUIGetPoint()` does not need to be ported.

GUISetWindowText

Set window caption.

`gtk_set_window_title()` should be used here.

Time needed for porting: 2 hours

GUIGetWindowText

Get window caption.

`gtk_get_window_title()` should be used here.

Time needed for porting: 2 hours

GUIGetWindowTextLength

Get caption text length.

In GTK the length of a char, pointer to which is returned by `gtk_get_window_title()` can be measured.

Time needed for porting: 2 hours

GUIGetExtentX

Finds X extent of the given string in current font.

Width of the given string in specified window could be obtained in the following way.

```
PangoLayout pango_layout = gtk_widget_create_layout(fixed, NULL);
pango_layout_set_text(pango_layout, text, text_len);
pango_layout_get_pixel_size(pango_layout, &width, NULL);
```

Where `fixed` is the `GtkFixed` widget located in the given window, `text` is the given text in the UTF-8 ecoding, `text_len` is the length if the text, and `width` is variable for desired value.

Time needed for porting: 2 hours.

GUIGetExtentY

Finds Y extent of the given string in current font.

Height of the given string in specified window could be obtained in the following way.

```
PangoLayout pango_layout = gtk_widget_create_layout(fixed, NULL);
pango_layout_set_text(pango_layout, text, text_len);
pango_layout_get_pixel_size(pango_layout, NULL, &height);
```

Where `fixed` is the `GtkFixed` widget located in the given window, `text` is the given text in the UTF-8 ecoding, `text_len` is the length if the text, and `height` is variable for desired value.

Time needed for porting: 2 hours.

GUIGetControlExtentX

Finds X extent of the given string in font of the given control.

Width of the given string in specified window could be obtained in the following way.

```
PangoLayout pango_layout = gtk_widget_create_layout(widget,
NULL);
pango_layout_set_text(pango_layout, text, text_len);
pango_layout_get_pixel_size(pango_layout, &width, NULL);
```

Where `widget` is the given widget returned by `GUIGetControlByID` located in the given window, `text` is the given text in the UTF-8 ecoding, `text_len` is the length if the text, and `width` is variable for desired value.

Time needed for porting: 2 hours.

GUIGetControlExtentY

Finds Y extent of the given string in font of the given control.

Height of the given string in specified window could be obtained in the following way.

```
PangoLayout pango_layout = gtk_widget_create_layout(widget,
NULL);
pango_layout_set_text(pango_layout, text, text_len);
pango_layout_get_pixel_size(pango_layout, NULL, &height);
```

Where `widget` is the given widget returned by `GUIGetControlByID` located in the given window, `text` is the given text in the UTF-8 encoding, `text_len` is the length of the text, and `height` is variable for desired value.

Time needed for porting: 2 hours.

GUIGetStringPos

Returns offset (in characters) of the given point if string is offset from left by given amount.

Calling of `GUIGetTheDC()` and `DoReturn` should be eliminated in this function as GTK operates at a higher level.

Time needed for porting: 1 hour.

Summary

15 hours are needed for porting activity and, additionally, approximately 15 hours are needed for testing and tuning. This step requires about 30 hours.

18. Drawing functions

`GUIFillRect()`

`GUIDrawRect()`

`GUIDrawLine()`

`GUIFillRectRGB()`

`GUIDrawRectRGB()`

`GUIDrawLineRGB()`

`GUIDrawText()`

`GUIDrawTextPos()`

`GUIDrawTextExtent()`

`GUIDrawTextExtentPos()`

`GUIDrawTextRGB()`

`GUIDrawTextPosRGB()`

`GUIDrawTextExtentRGB()`

`GUIDrawTextExtentPosRGB()`

`GUIDrawBar()`

`GUIDrawBarGroup()`

Since a couple of functions use other already ported functions, just a few of these functions needs modifications to make them work under GTK.

GUIDrawLine and GUIDrawLineRGB

Draw a line given a `gui_attr` or RGB and style information.

These functions use `DrawLine()` function which should be rewritten using `gtk_paint_hline()`, `gtk_paint_vline()` or `gtk_paint_polygon()`.

Time needed for porting: 4 hour.

GUIDrawBar

Draws the outline of a rectangle given a `gui_attr`.

The function should use `gtk_paint_bar()` to perform the operation.

Time needed for porting: 4 hour.

Summary

8 hours are needed for porting activity and, additionally, approximately 16 hours are needed for testing and tuning. This step requires about 24 hours.

19. Font handling functions

`GUIFontsSupported()`

`GUIChangeFont()`

`GUIGetFontInfo()`

`GUISetFontInfo()`

`GUISetSystemFont()`

`GUIGetFontFromUser()`

GUIFontsSupported

Returns `TRUE` if these font functions are supported.

All these functions except `GUISetSystemFont()` can be implemented. So, `GUIFontsSupported()` should always return `TRUE`.

Time needed for porting: 0.5 hours.

GUIChangeFont

Gets font choice from user and changes the font of the given window.

The function could be ported implemented with `GUIGetFontFromUser()` and `GUISetFontInfo()`.

Time needed for porting: 2 hours.

GUIGetFontInfo

Gets the font information for a window.

`pango_font_description_to_string()` applied to a style attribute of the given window should be used to perform the operation.

Time needed for porting: 2 hours.

GUISetFontInfo

Sets the font information for a window.

The function should create `PangoFontDescription` via `pango_font_description_from_string()` function, and modify style of the given window with `gtk_widget_modify_style()`.

Time needed for porting: 3 hours.

GUISetSystemFont

Sets font to the system font (fixed or proportional).

It is impossible to implement this function in GTK. So, this function should do nothing in the GTK port.

Time needed for porting: 0.5 hours.

GUIGetFontFromUser

Gets the font information from the user using a dialog.

This function should be implemented using standard `GtkFontSelectionDialog()`.

Time needed for porting: 4 hours.

Summary

12 hours are needed for porting activity and, additionally, approximately 12 hours are needed for testing and tuning. This step requires 24 hours.

20. Cursor functions

`GUISetMouseCursor()`

`GUIResetMouseCursor()`

GUISetMouseCursor

Sets the type of mouse cursor.

This function should create an apropriate cursor with `gdk_cursor_new()`. And use `gdk_window_set_cursor()` applied to the result of `gdk_get_default_root_window()` to change the cursor.

Time needed for porting: 2 hours.

GUIResetMouseCursor

Resets the type of mouse cursor.

This function should use `gdk_window_set_cursor()` applied to the result of `gdk_get_default_root_window()` and pass `NULL` as cursor type to change the cursor to default.

Time needed for porting: 2 hours.

Summary

4 hours are needed for porting activity and 2 more hours are required for testing and tuning. This step requires about 6 hours.

21. Window functions

`GUIGetWindowColours()`

`GUIControlDirty()`

`GUIWndDirtyRow()`

`GUIWndDirtyRect()`

`GUIRefresh()`

`GUIActivateNC()`

`GUIBringToFront()`

`GUIGetRootWindow()`

`GUISetFocus()`

`GUIGetFocus()`

`GUIResizeWindow()`

`GUIIsMinimized()`

`GUIIsMaximized()`

`GUIRestoreWindow()`

`GUIHideWindow()`

`GUIShowWindow()`

`GUIIsWindowVisible()`

`GUISetRestoredSize()`

`GUIGetRestoredSize()`

`GUISetIcon()`

`GUICascadeWindows()`

GUIControlDirty

Causes refresh of the given control.

`gdk_window_process_updates()` can repaint the control obtained via `GUIGetControlByID()`.

Time needed for porting: 2 hours.

GUIWndDirtyRow

Causes refresh of the given row.

`gdk_widget_queue_draw_area()` can repaint the window region. Note that calculation of the row coordinates is needed.

Time needed for porting: 3 hours.

GUIWndDirtyRect

Causes refresh of the given rect.

`gdk_widget_queue_draw_area()` can repaint the window region.

Time needed for porting: 3 hours.

GUIRefresh

Causes refresh of the screen.

`gdk_window_process_all_updates()` can repaint all windows.

Time needed for porting: 2 hours.

GUIBringToFront

Bring the window to the top of all others.

`Gdk_window_show()` should be used.

Time needed for porting: 2 hours.

GUISetFocus

Sets input focus to a control in a dialog box or in a window.

`gtk_window_set_focus()` applied to the control obtained via `GUIGetControlByID()` should be used here.

Time needed for porting: 2 hours.

GUIGetFocus

Finds out which main window has the focus.

This function makes no sense in GTK as it is impossible to implement an MDI in GTK. When ported – it will always return supplied parameter without modification.

Time needed for porting: 0.5 hour.

GUIResizeWindow

Gives the window a new size and location.

`gtk_window_move()` should be called to move the window, and `gtk_widget_set_size_request()` called to set a new size.

Note, that window managers are free to ignore the moving request; most window managers ignore request for initial window positions (instead using a user-defined placement algorithm) and honor requests after the window has already been shown.

Time needed for porting: 1 hour.

GUIIsMinimized and GUIIsMaximized

Returns true if window is minimized/maximized.

`gdk_window_get_state()` should be used to perform the operation.

Time needed for porting: 2 hours.

GUIRestoreWindow

Restore window to pre-minimize or maximize size.

`gdk_window_unmaximize()` should be used to perform the operation.

Time needed for porting: 1 hour.

GUIHideWindow

Hides the given window.

`gdk_window_show()` should be used to perform the operation.

Time needed for porting: 1 hour.

GUIIsWindowVisible

Checks if the given window is visible.

`gdk_is_window_visible()` should be used to perform the operation.

Time needed for porting: 1 hour.

GUISetRestoredSize

Sets a size of the given window to a restored state.

`gtk_window_set_size()` should be used to perform the operation.

Time needed for porting: 1 hour.

GUIGetRestoredSize

Gets a size of the given window in a restored state.

`gtk_window_get_size()` should be used to perform the operation.

Time needed for porting: 1 hour.

GUISetIcon

Sets an icon for the given window.

`gtk_window_set_icon()` should be used to perform the operation.

Time needed for porting: 2 hours.

GUICascadeWindows

Arrange all child windows in MDI in cascade.

It is impossible to implement in GTK, so this function should just return TRUE.

Time needed for porting: 0.5 hour.

Summary

25 hours are needed for the porting activity and, additionally, approximately 30 hours are required for testing and tuning. This step requires about 55 hours.

22. Hot spot functions

`GUIInitHotSpots()`

`GUIGetNumHoSpots()`

`GUIGetHotSpotSize()`

`GUIDrawHotSpot()`

GUIInitHotSpots

Sets the bitmaps associated with user defined hot spots.

The function should create an array of `GtkImage` objects. Some code to convert bitmaps passed in function to some acceptable format will be needed.

Time needed for porting: 8 hours.

GUIDrawHotSpot

Draws a hot spot at a given location.

The function should place the requested hot spot in the appropriate position with `gtk_fixed_put()` function.

Time needed for porting: 2 hours.

Summary

10 hours are needed for porting activity and 6 more hours are needed for testing and tuning. This step requires about 16 hours.

23. Information functions

`GUIGetKeyState()`

`GUISetExtra()`

`GUIGetExtra()`

`GUIGetClientRect()`

`GUIGetPaintRect()`

`GUIGetAbsRect()`

`GUIGetRect()`

`GUIGetScrollStyle()`

`GUIGetCreateStyle()`

`GUIGetMousePosn()`

`GUIGetSystemMetrics()`

`GUIGetMinSize()`

`GUIEnumChildWindows()`

`GUIEnumControls()`

`GUIGetArgs()`

GUIGetKeyState

Gets the current Shift-key state.

Since the function doesn't allow specifying which input device should be tested, it isn't clear.Shift key on which input device should be checked. However as this function is called only in event callbacks, it is possible to get the current Shift state of the input device which initiated the event via `gdk_keymap_translate_keyboard_state()` function.

GUIGetPaintRect

The function uses `_wpi_getpaintrect()` which should be reimplemented for the GTK port to support `GdkRectangle` structure.

Time needed for porting: 2 hours.

GUIGetAbsRect

Get a window size and location in absolute user defined units.

Windows size can be obtained via `gtk_window_get_size()` function and position can be obtained via `gtk_window_get_position` function.

Time needed for porting: 2 hours.

GUIGetAbsRect

Get a window size and location relative to its parent.

The function should call `GUIGetAbsRect()`.

Time needed for porting: 0.5 hours.

GUIGetMousePosn

Gets the mouse position (in user coordinates).

`gtk_widget_get_pointer()` function should be used to obtain the location of the mouse pointer in widget coordinates. Then the coordinates should be transformed relative to a WGL scaling factor.

Time needed for porting: 2 hours.

GUIGetMinSize

Returns the smallest valid size for the window.

The function returns result based in `MIN_WIDTH` and `MIN_HEIGHT` definitions that are defined as 0. But in GTK windows may not be resized smaller that 1 by 1 pixels. So these definitions should be changed in the GTK port.

Time needed for porting: 0.5 hours.

GUIEnumChildWindows

Enumerates child windows by calling given function with the `gui_window` for each.

The function should call `gdk_window_get_children()`, get the child's `gui_window` via `GUIFindWindowFromHWND`, and call necessary callback.

Time needed for porting: 4 hours.

Summary

11 hours are required for the porting activity and, additionally, 10 hours are required for testing and tuning. This step requires about 21 hours.

24. Color functions

`GUISetWindowColours()`

`GUISetWndColour()`

`GUIGetRGBFromUser()`

`GUIGetColourFromUser()`

GUIGetRGBFromUser

Creates dialog to ask user for color, returns its RGB.

`GtkColorSelection()` should used for this purpose.

Time needed for porting: 4 hours.

Summary

4 hours are needed for porting activity and 2 more hours are needed for testing and tuning. This step requires about 6 hours.

25. Combo list/box functions

`GUIAddText()`

`GUISetListItemData()`

`GUIGetListItemData()`

`GUIAddTextList()`

`GUIInsertText()`

`GUIClearList()`

`GUIDeleteItem()`

`GUIGetListSize()`

`GUIGetCurrSelect()`

`GUISetCurrSelect()`

`GUIGetListItem()`

`GUILimitEditText()`

`GUIInsertMenuByID`

GUIAddText

Adds a text item to the list.

`gtk_list_append_items()` function should be used to perform the operation.

Time needed for porting: 2 hours.

GUISetListItemData

Associates data with a list box item.

`gtk_object_set_user_data()` function should be used to perform the operation.

Time needed for porting: 1 hour.

GUIGetListItemData

Gets the data associated with a list box item.

`gtk_object_get_user_data()` function should be used to perform the operation.

Time needed for porting: 4 hours.

GUIAddTextList

Adds a text list item to the list using a callback.

`gtk_list_append_items()` function should be used to perform the operation.

Time needed for porting: 4 hours.

GUIInsertText

Inserts a text item to a given location in list.

`gtk_list_insert_items()` function should be used to perform the operation.

Time needed for porting: 2 hours.

GUIClearList

Removes all items from the list.

`gtk_list_clear_items()` function should be used to perform the operation.

Time needed for porting: 2 hours.

GUIDeleteItem

Deletes an item from the list.

`gtk_list_clear_items()` function should be used to perform the operation.

Time needed for porting: 2 hours.

GUIGetListSize

Returns the number of items in the list.

The number of items in the list can be obtained via `g_list_length()` applied to the result of `gtk_container_get_children()` function.

Time needed for porting: 2 hours.

GUIGetCurrSelect

Gets the position of the current selection.

To get the list of the selected items `selection` field of the `GtkList` should be used. Then, it is needed to compare all the elements in the list with the selected item, and find out number of the selected item.

Time needed for porting: 3 hours.

GUISetCurrSelect

Sets current selection by position.

`gtk_list_select_item()` function should be used to perform the operation.

Time needed for porting: 2 hours.

GUIGetListItem

Gets the text of a list item by position.

`g_list_nth()` applied to the result of `gtk_container_get_children()` function should be used to perform the operation.

Time needed for porting: 3 hours.

GUILimitEditText

Sets the maximum allowed length of the contents of the combo box.

`gtk_entry_set_max_length()` applied to `entry` field of the `GtkCombo` should be used to perform the operation.

Time needed for porting: 3 hours.

GUIInsertMenuByID

Inserts a sub menu into the menu before the item with the given ID.

The function should insert an entry in the `GtkList` widget using `g_list_insert_before()` function.

Time needed for porting: 3 hours.

Summary

33 hours is needed for porting activity and, additionally, it is needed approximately 33 hours for testing and tuning. This step requires about 66 hours.

26. Radio button and check box functions

`GUIIsChecked()`

`GUISetChecked()`

GUIIsChecked

Finds out if button is checked or not.

`gtk_toggle_button_get_active()` should be used to perform the operation.

Time needed for porting: 2 hours.

GUISetChecked

Sets button as checked or not.

`gtk_toggle_button_set_active()` should be used to perform the operation.

Time needed for porting: 2 hours.

Summary

4 hours are needed for porting activity and, additionally, 6 hours are required for testing and tuning. This step requires about 10 hours.

27. F1 key hook functions

`GUIHookF1()`

`GUIUnHookF1()`

GUIHookF1

Sets the hook for F1 key pressing event.

Required functionality can be achieved by using `gtk_key_snooper_install()` function.

Time needed for porting: 4 hours.

GUIUnHookF1

Removes the hook for F1 key pressing event.

Required functionality can be achieved by using `gtk_key_snooper_install()` function.

Time needed for porting: 1 hour.

Summary

5 hours are needed for porting activity and, additionally, 4 hours are needed for testing and tuning. This step requires about 9 hours.

28. Other functions.

`GUISpawnStart()`

`GUISpawnEnd()`

`GUICharLen()`

First two does nothing in the GUI version. The third function returns length of a specified character in a char places. In this case, this function should always return 1.

29. Type definitions

Here is the basic list of definitions that need to be defined in the GTK port. The definitions of this list was determined on investigation stage, however programmer that will be porting the library will be able determine all necessary type definitions more sharply. Time required – 16 hours.

```
WPI_TEXTMETRIC – PangoFontMetrics

HBRUSH -- GtkRcStyle*

HWND -- GtkWindow*

WM_CLOSE -- "close"

WM_RESIZE -- "size-request"

WPI_COLOUR -- GdkColor*
```

30. Functions that are never used in the current code

GUIDropDown

Drops down or raise dropped down list box.

GUIScrollCaret

Scrolls the caret.

GUISetTopIndex

Sets the index of item at the top of list.

GUIGetTopIndex

Gets the index of item at the top of list.

GUISetHorizontalExtent

Sets the width of the widest list box item.

GUIActivateNC

Activates non client MDI window.