UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Scienze della Terra

ALLEGATI ALLA TESI DI LAUREA

CODICI

Relatore: Prof. Alberto Luigi Marcellini
Correlatore: Prof. Roberto Sabadini

Tesi di Laurea di:
Valerio Poggi
Matr. n. 535266

Anno Accademico 2005/2006

Indice

SEZIONE 1

GSesame 1.0
Codice Sorgente

1.1 - Gsesame.c

```
/*****************************************************************************/
/*  G-SESAME Version 1.0:                                                    */
/*  a graphical front-end for SESAME modules written with GTK+               */
/*****************************************************************************/
/*  Copyright (c) 2005 Poggi Valerio                                         */
/*  Email: valerio.poggi@idpa.cnr.it                                         */
/*****************************************************************************/
/*  This program is free software; you can redistribute it and/or modify     */
/*  it under the terms of the GNU General Public License as published by     */
/*  the Free Software Foundation; either version 2 of the License, or        */
/*  (at your option) any later version.                                      */
/*                                                                           */
/*  This program is distributed in the hope that it will be useful,          */
/*  but WITHOUT ANY WARRANTY; without even the implied warranty of           */
/*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            */
/*  GNU General Public License for more details.                             */
/*                                                                           */
/*  You should have received a copy of the GNU General Public License        */
/*  along with this program; if not, write to the Free Software              */
/*  Foundation, Inc., 59 Temple Place, Suite 330, Boston,                    */
/*  MA  02111-1307  USA                                                      */
/*****************************************************************************/

# include <gtk/gtk.h>
# include <string.h>
# include <time.h>
# include <locale.h>

# include "globals.h"
# include "gsesame.h"
# include "platdep.h"
# include "toolbar.h"
# include "menu.h"
# include "list.h"
# include "winselcfg.h"
# include "hvproccfg.h"
# include "gsescfg.h"
# include "mainproc.h"
# include "about.h"
# include "thanks.h"
# include "warning.h"

/*****************************************************************************/
/* MAIN FUNCTION                                                             */
/*****************************************************************************/

gint main (gint argc, gchar *argv[])
{
    GtkWidget *h_separator;
    GtkWidget *status_bar;

    /* Disabling Gtk localization */
    gtk_disable_setlocale();
    setlocale (LC_ALL, "");
    setlocale (LC_NUMERIC, "C");

    /* Gtk initialization */
    gtk_init (&argc, &argv);

    /* Set absolute and relative path */
```

```
set_path ();

/* Configuration windows */
create_ws_par_window ();
create_hv_par_window ();
create_gs_par_window ();

/* Processing console */
create_console_window ();

/* About & Thanks windows */
create_about_window ();
create_thanks_window ();

/* Main window */
{
    main_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_widget_set_size_request (GTK_WIDGET(main_window), 550, 300);
    gtk_window_move (GTK_WINDOW (main_window), 50, 50);
    gtk_widget_show (main_window);

    /* Project's default parameters setup*/
    {
        set_default_project_name ();
        set_project_file ();
        set_project_title ();
    }

    /* Quit event signal connection */
    {
        g_signal_connect (G_OBJECT(main_window), "delete_event",
                          G_CALLBACK(warning_unsaved_work),NULL);
        g_signal_connect (G_OBJECT(main_window), "destroy",
                          G_CALLBACK(warning_unsaved_work),NULL);
    }
}

/* Main vertical box */
{
    vbox_main = gtk_vbox_new (FALSE,0);
    gtk_container_add (GTK_CONTAINER(main_window), vbox_main);
    gtk_widget_show (vbox_main);
}

/* Menu bar */
create_menu ();

/* Button toolbar */
create_toolbar ();

/* File list widget */
/* NOTE: the first item of the saf structure MUST be initilized to NULL */
{
    first_saf = NULL;
    create_gtk_list (vbox_main, argc, argv);
}

/* Status bar (for future use) */
{
    h_separator = gtk_hseparator_new ();
    gtk_box_pack_start (GTK_BOX (vbox_main), h_separator, FALSE, TRUE, 0);
    gtk_widget_show (h_separator);
```

```
        status_bar = gtk_statusbar_new ();
        gtk_box_pack_start (GTK_BOX (vbox_main), status_bar, FALSE, TRUE, 0);
        gtk_widget_show (status_bar);
    }

    /* Main window should be the next window showed and actived (optional) */
    gtk_widget_show_all (main_window);

    /* GTK main loop */
    gtk_main ();

    /* End of main */
    return 0;
}

/*****************************************************************************/
/* SET_DEFAULT_PROJECT_NAME:                                                 */
/* This function automatically generates a default project name using the    */
/* actual creation date ("Project_day_month_year").                          */
/*****************************************************************************/

void set_default_project_name (void)
{
    time_t now;
    struct tm *time_ptr;

    time (&now);
    time_ptr = localtime (&now);
    strftime (project.name, 50, "Project_%d-%m-%Y", time_ptr);
}

/*****************************************************************************/
/* SET_PROJECT_FILE:                                                         */
/* This function only generates a default project file name from the         */
/* current project name, but doesn't save that file.                         */
/*****************************************************************************/

void set_project_file (void)
{
    sprintf (project.file, "%s.prj", project.name);
}

/*****************************************************************************/
/* SET_PROJECT_TITLE:                                                        */
/* This function sets the main window's title including in it the current    */
/* project name.                                                             */
/*****************************************************************************/

void set_project_title (void)
{
    sprintf (project.title_bar, "GSesame Version 1.0 - %s", project.name);
    gtk_window_set_title (GTK_WINDOW(main_window), project.title_bar);
}

/*****************************************************************************/
/* RESET_PROJECT_FILE:                                                       */
/* This is a callback function that simply modify the project file name.     */
/* It's called when the user want to save the project with a different name  */
/* or in a different place (see the "SAVE_PROJECT_AS" function).             */
/*****************************************************************************/

void reset_project_file (GtkWidget *widget, gpointer abstrptr)
{
```

```
    const gchar *string;

    GtkWidget *file_selector = GTK_WIDGET (abstrptr);
    string = gtk_file_selection_get_filename
             (GTK_FILE_SELECTION(file_selector));
    strcpy (project.file, string);
}
```

1.2 - Gsesame.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef GSESAME_H
#define GSESAME_H

    void    set_default_project_name    (void);

    void    set_project_file            (void);

    void    set_project_title           (void);

    void    reset_project_file          (GtkWidget *widget,
                                         gpointer abstrptr);

# endif
```

1.3 - Globals.h

```
/*****************************************************************************/
/* GLOBAL VARIABLES DECLARATION:                                             */
/*****************************************************************************/

#ifndef GLOBALS_H
#define GLOBALS_H

    struct saf_file
    {
        gchar *file;
        gchar sta_code   [100];
        gchar start_time [100];
        gchar samp_freq  [100];
        gchar ndat       [100];
        gchar ch0_id     [100];
        gchar ch1_id     [100];
        gchar ch2_id     [100];
        gchar units      [100];
        struct saf_file *next_saf;
    };

    struct prj_struct
    {
       gchar name[50];
       gchar file[100];
       gchar title_bar[100];
    } project;

    struct confpar_struct
```

```
{
    gdouble stalen_def;
    gdouble ltalen_def;
    gdouble minstalta_def;
    gdouble maxstalta_def;
    gdouble winlen_def;
    gdouble overlap_def;
    gdouble tolerance_def;
    gboolean saturation_def;
    gboolean noisywin_def;

    gdouble stalen;
    gdouble ltalen;
    gdouble minstalta;
    gdouble maxstalta;
    gdouble winlen;
    gdouble overlap;
    gdouble tolerance;
    gboolean saturation;
    gboolean noisywin;

    gint  freqsp_type_def;
    gdouble freqsp_min_def;
    gdouble freqsp_max_def;
    gdouble freqsp_npnt_def;
    gint   offrem_type_def;
    gdouble offrem_freq_def;
    gint   tape_type_def;
    gdouble tape_perc_def;
    gint   smth_type_def;
    gdouble smth_band_def;
    gint   smth_wgh_def;
    gint   merging_def;
    gboolean outsinwin_def;

    gint   freqsp_type;
    gdouble freqsp_min;
    gdouble freqsp_max;
    gdouble freqsp_npnt;
    gint   offrem_type;
    gdouble offrem_freq;
    gint   tape_type;
    gdouble tape_perc;
    gint   smth_type;
    gdouble smth_band;
    gint   smth_wgh;
    gint   merging;
    gboolean outsinwin;
} confpar;

typedef struct saf_file SAF_LIST;
typedef SAF_LIST *SAF_LIST_PTR;

SAF_LIST_PTR first_saf;

GtkWidget *main_window;
GtkWidget *vbox_main;
GtkWidget *view;

gboolean plotfo;
gboolean plotsdhv;
gboolean plotsdfo;
gboolean imgfmt;
```

```
    gboolean plotfo_def;
    gboolean plotsdhv_def;
    gboolean plotsdfo_def;
    gboolean imgfmt_def;

    gchar imgext[50];

#endif
```

1.4 - Menu.c

```
# include <gdk/gdkkeysyms.h>
# include <gtk/gtk.h>

# include "globals.h"
# include "menu.h"
# include "list.h"
# include "project.h"
# include "saf.h"
# include "winselcfg.h"
# include "hvproccfg.h"
# include "gsescfg.h"
# include "plotch.h"
# include "plothv.h"
# include "mainproc.h"
# include "about.h"
# include "warning.h"
# include "icons/ch16x16.xpm"
# include "icons/hv16x16.xpm"

/*****************************************************************************/
/* CREATE_MENU:                                                              */
/* This function create the the main menu bar with submenu. It's done in the  */
/* "hard way" (more code but more control).                                  */
/*****************************************************************************/

void create_menu(void)
{
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle  *style;

    GtkWidget *menu_bar;

    GtkWidget *file_item;
    GtkWidget *file_menu;

        GtkWidget *new_prj_item;
        GtkWidget *new_prj_img;
        GtkWidget *open_prj_item;
        GtkWidget *open_prj_img;
        GtkWidget *save_prj_item;
        GtkWidget *save_prj_img;
        GtkWidget *save_prj_as_item;
        GtkWidget *save_prj_as_img;
        GtkWidget *file_sep1;
        GtkWidget *import_saf_item;
        GtkWidget *import_saf_img;
        GtkWidget *delete_saf_item;
        GtkWidget *delete_saf_img;
```

```
        GtkWidget *clear_list_item;
        GtkWidget *clear_list_img;
        GtkWidget *file_sep2;
        GtkWidget *exit_item;
        GtkWidget *exit_img;

    GtkWidget *preferences_item;
    GtkWidget *preferences_menu;

        GtkWidget *ws_par_item;
        GtkWidget *ws_par_img;
        GtkWidget *hv_par_item;
        GtkWidget *hv_par_img;
        GtkWidget *preferences_sep1;
        GtkWidget *gen_opt_item;
        GtkWidget *gen_opt_img;

    GtkWidget *ch_plot_item;
    GtkWidget *ch_plot_menu;

        GtkWidget *ch_0_item;
        GtkWidget *ch_0_img;
        GtkWidget *ch_1_item;
        GtkWidget *ch_1_img;
        GtkWidget *ch_2_item;
        GtkWidget *ch_2_img;

    GtkWidget *processing_item;
    GtkWidget *processing_menu;

        GtkWidget *hv_sp_item;
        GtkWidget *hv_sp_img;
        GtkWidget *hv_sp_menu;
        GtkWidget *selected_hv_item;
        GtkWidget *all_hv_item;
        GtkWidget *average_hv_item;

    GtkWidget *as_plot_item;
    GtkWidget *as_plot_menu;

        GtkWidget *as_0_plot_item;
        GtkWidget *as_0_plot_img;
        GtkWidget *as_1_plot_item;
        GtkWidget *as_1_plot_img;
        GtkWidget *as_2_plot_item;
        GtkWidget *as_2_plot_img;

    GtkWidget *sp_plot_item;
    GtkWidget *sp_plot_menu;

        GtkWidget *av_sp_plot_item;
        GtkWidget *av_sp_plot_img;
        GtkWidget *c1_sp_plot_item;
        GtkWidget *c1_sp_plot_img;
        GtkWidget *c2_sp_plot_item;
        GtkWidget *c2_sp_plot_img;

    GtkWidget *help_item;
    GtkWidget *help_menu;

        GtkWidget *gs_man_item;
        GtkWidget *gs_man_img;
        GtkWidget *help_sep1;
```

```
    GtkWidget *about_item;
    GtkWidget *about_img;

GtkAccelGroup *accel_group;


/* Make an accelerator group (shortcut keys) */
accel_group = gtk_accel_group_new ();

/* Creating Menu bar */
menu_bar = gtk_menu_bar_new ();
gtk_box_pack_start (GTK_BOX (vbox_main), menu_bar, FALSE, TRUE, 0);

/* Widget style for creating pixmap */
style = gtk_widget_get_style(menu_bar);

/*************************************************************************/
/* File Menu                                                             */
/*************************************************************************/

file_item = gtk_menu_item_new_with_mnemonic ("_File");
gtk_widget_show (file_item);
gtk_container_add (GTK_CONTAINER (menu_bar), file_item);

file_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (file_item), file_menu);

    /* New Project */

    new_prj_item = gtk_image_menu_item_new_with_mnemonic
                   ("_New Project");
    gtk_widget_show (new_prj_item);
    gtk_container_add (GTK_CONTAINER (file_menu), new_prj_item);
    g_signal_connect (GTK_WIDGET(new_prj_item), "activate",
                      G_CALLBACK (new_project), NULL);
    gtk_widget_add_accelerator (new_prj_item, "activate", accel_group,
                                GDK_N, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    new_prj_img = gtk_image_new_from_stock ("gtk-new",
                                            GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (new_prj_item),
                                   new_prj_img);
    gtk_widget_show (new_prj_img);

    /* Open Project */

    open_prj_item = gtk_image_menu_item_new_with_mnemonic
                    ("_Open Project");
    gtk_widget_show (open_prj_item);
    gtk_container_add (GTK_CONTAINER (file_menu), open_prj_item);
    g_signal_connect (GTK_WIDGET(open_prj_item), "activate",
                      G_CALLBACK (open_project), NULL);
    gtk_widget_add_accelerator (open_prj_item, "activate", accel_group,
                                GDK_O, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    open_prj_img = gtk_image_new_from_stock ("gtk-open",
                                             GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (open_prj_item),
                                   open_prj_img);
    gtk_widget_show (open_prj_img);
```

```
/* Save Project */

save_prj_item = gtk_image_menu_item_new_with_mnemonic
                ("_Save Project");
gtk_widget_show (save_prj_item);
gtk_container_add (GTK_CONTAINER (file_menu), save_prj_item);
g_signal_connect (GTK_WIDGET(save_prj_item), "activate",
                  G_CALLBACK (save_project), NULL);
gtk_widget_add_accelerator (save_prj_item, "activate", accel_group,
                            GDK_S, GDK_CONTROL_MASK,
                            GTK_ACCEL_VISIBLE);

save_prj_img = gtk_image_new_from_stock ("gtk-save",
                                         GTK_ICON_SIZE_MENU);
gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (save_prj_item),
                               save_prj_img);
gtk_widget_show (save_prj_img);

/* Save Project As */

save_prj_as_item = gtk_image_menu_item_new_with_mnemonic
                   ("Save Project _As");
gtk_widget_show (save_prj_as_item);
gtk_container_add (GTK_CONTAINER (file_menu), save_prj_as_item);
g_signal_connect (GTK_WIDGET(save_prj_as_item), "activate",
                  G_CALLBACK (save_project_as), NULL);
gtk_widget_add_accelerator (save_prj_as_item, "activate", accel_group,
                            GDK_A, GDK_CONTROL_MASK,
                            GTK_ACCEL_VISIBLE);

save_prj_as_img = gtk_image_new_from_stock ("gtk-save-as",
                                            GTK_ICON_SIZE_MENU);
gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (save_prj_as_item),
                               save_prj_as_img);
gtk_widget_show (save_prj_as_img);

/* Separator */

file_sep1 = gtk_separator_menu_item_new ();
gtk_widget_show (file_sep1);
gtk_container_add (GTK_CONTAINER (file_menu), file_sep1);
gtk_widget_set_sensitive (file_sep1, FALSE);

/* Import Saf */

import_saf_item = gtk_image_menu_item_new_with_mnemonic
                  ("Import Saf _File");
gtk_widget_show (import_saf_item);
gtk_container_add (GTK_CONTAINER (file_menu), import_saf_item);
g_signal_connect (GTK_WIDGET(import_saf_item), "activate",
                  G_CALLBACK (saf_file_selection), NULL);
gtk_widget_add_accelerator (import_saf_item, "activate", accel_group,
                            GDK_F, GDK_CONTROL_MASK,
                            GTK_ACCEL_VISIBLE);

import_saf_img = gtk_image_new_from_stock ("gtk-add",
                                           GTK_ICON_SIZE_MENU);
gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (import_saf_item),
                               import_saf_img);
gtk_widget_show (import_saf_img);

/* Delete Saf */
```

```
    delete_saf_item = gtk_image_menu_item_new_with_mnemonic
                      ("_Delete Item");
    gtk_widget_show (delete_saf_item);
    gtk_container_add (GTK_CONTAINER (file_menu), delete_saf_item);
    g_signal_connect (GTK_WIDGET(delete_saf_item), "activate",
                      G_CALLBACK (del_from_list), NULL);
    gtk_widget_add_accelerator (delete_saf_item, "activate", accel_group,
                                GDK_D, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    delete_saf_img = gtk_image_new_from_stock ("gtk-delete",
                                               GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (delete_saf_item),
                                   delete_saf_img);
    gtk_widget_show (delete_saf_img);

    /* Clear List */

    clear_list_item = gtk_image_menu_item_new_with_mnemonic
                      ("_Clear List");
    gtk_widget_show (clear_list_item);
    gtk_container_add (GTK_CONTAINER (file_menu), clear_list_item);
    g_signal_connect (GTK_WIDGET(clear_list_item), "activate",
                      G_CALLBACK (clear_list), NULL);
    gtk_widget_add_accelerator (clear_list_item, "activate", accel_group,
                                GDK_C, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    clear_list_img = gtk_image_new_from_stock ("gtk-clear",
                                               GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (clear_list_item),
                                   clear_list_img);
    gtk_widget_show (clear_list_img);

    /* Separator */

    file_sep2 = gtk_separator_menu_item_new ();
    gtk_widget_show (file_sep2);
    gtk_container_add (GTK_CONTAINER (file_menu), file_sep2);
    gtk_widget_set_sensitive (file_sep2, FALSE);

    /* Exit */

    exit_item = gtk_image_menu_item_new_with_mnemonic
                ("_Exit");
    gtk_widget_show (exit_item);
    gtk_container_add (GTK_CONTAINER (file_menu), exit_item);
    g_signal_connect (GTK_WIDGET(exit_item), "activate",
                      G_CALLBACK (warning_unsaved_work), NULL);
    gtk_widget_add_accelerator (exit_item, "activate", accel_group,
                                GDK_Q, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    exit_img = gtk_image_new_from_stock ("gtk-quit",
                                         GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (exit_item),
                                   exit_img);
    gtk_widget_show (exit_img);

/*************************************************************************/
/* Preferences Menu                                                      */
/*************************************************************************/
```

```
preferences_item = gtk_menu_item_new_with_mnemonic ("_Preferences");
gtk_widget_show (preferences_item);
gtk_container_add (GTK_CONTAINER (menu_bar), preferences_item);

preferences_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (preferences_item),
                           preferences_menu);

    /* Window Selection Preferences */

    ws_par_item = gtk_image_menu_item_new_with_mnemonic
                  ("_Window Selection Parameters");
    gtk_widget_show (ws_par_item);
    gtk_container_add (GTK_CONTAINER (preferences_menu), ws_par_item);
    g_signal_connect (GTK_WIDGET(ws_par_item), "activate",
                      G_CALLBACK (ws_window_showhide), NULL);
    gtk_widget_add_accelerator (ws_par_item, "activate", accel_group,
                                GDK_W, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    ws_par_img = gtk_image_new_from_stock ("gtk-preferences",
                                           GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (ws_par_item),
                                   ws_par_img);
    gtk_widget_show (ws_par_img);

    /* HV Processing Preferences */

    hv_par_item = gtk_image_menu_item_new_with_mnemonic
                  ("_H/V Processing Parameters");
    gtk_widget_show (hv_par_item);
    gtk_container_add (GTK_CONTAINER (preferences_menu), hv_par_item);
    g_signal_connect (GTK_WIDGET(hv_par_item), "activate",
                      G_CALLBACK (hv_window_showhide), NULL);
    gtk_widget_add_accelerator (hv_par_item, "activate", accel_group,
                                GDK_H, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    hv_par_img = gtk_image_new_from_stock ("gtk-preferences",
                                           GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (hv_par_item),
                                   hv_par_img);
    gtk_widget_show (hv_par_img);

    /* Separator */

    preferences_sep1 = gtk_separator_menu_item_new ();
    gtk_widget_show (preferences_sep1);
    gtk_container_add (GTK_CONTAINER (preferences_menu), preferences_sep1);
    gtk_widget_set_sensitive (preferences_sep1, FALSE);

    /* GSesame General Options */

    gen_opt_item = gtk_image_menu_item_new_with_mnemonic
                   ("_GSesame General Options");
    gtk_widget_show (gen_opt_item);
    gtk_container_add (GTK_CONTAINER (preferences_menu), gen_opt_item);
    g_signal_connect (GTK_WIDGET(gen_opt_item), "activate",
                      G_CALLBACK (gs_window_showhide), NULL);
    gtk_widget_add_accelerator (gen_opt_item, "activate", accel_group,
                                GDK_G, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);
```

```
    gen_opt_img = gtk_image_new_from_stock ("gtk-preferences",
                                            GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (gen_opt_item),
                                   gen_opt_img);
    gtk_widget_show (gen_opt_img);

/*************************************************************************/
/* Trace Plot Menu                                                       */
/*************************************************************************/

ch_plot_item = gtk_menu_item_new_with_mnemonic ("_Trace Plot");
gtk_widget_show (ch_plot_item);
gtk_container_add (GTK_CONTAINER (menu_bar), ch_plot_item);

ch_plot_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (ch_plot_item), ch_plot_menu);

pixmap = gdk_pixmap_create_from_xpm_d (menu_bar->window, &mask,
         &style->bg[GTK_STATE_NORMAL],(gchar **) ch16x16_xpm);

    /* Channel 0 Plot */

    ch_0_item = gtk_image_menu_item_new_with_mnemonic
                ("Channel _0");
    gtk_widget_show (ch_0_item);
    gtk_container_add (GTK_CONTAINER (ch_plot_menu), ch_0_item);
    g_signal_connect (GTK_WIDGET(ch_0_item), "activate",
                      G_CALLBACK (plot_channel), "Channel 0");
    gtk_widget_add_accelerator (ch_0_item, "activate", accel_group,
                                GDK_0, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    ch_0_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (ch_0_item),
                                   ch_0_img);
    gtk_widget_show (ch_0_img);

    /* Channel 1 Plot */

    ch_1_item = gtk_image_menu_item_new_with_mnemonic
                ("Channel _1");
    gtk_widget_show (ch_1_item);
    gtk_container_add (GTK_CONTAINER (ch_plot_menu), ch_1_item);
    g_signal_connect (GTK_WIDGET(ch_1_item), "activate",
                      G_CALLBACK (plot_channel), "Channel 1");
    gtk_widget_add_accelerator (ch_1_item, "activate", accel_group,
                                GDK_1, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    ch_1_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (ch_1_item),
                                   ch_1_img);
    gtk_widget_show (ch_1_img);

    /* Channel 2 Plot */

    ch_2_item = gtk_image_menu_item_new_with_mnemonic
                ("Channel _2");
    gtk_widget_show (ch_2_item);
    gtk_container_add (GTK_CONTAINER (ch_plot_menu), ch_2_item);
    g_signal_connect (GTK_WIDGET(ch_2_item), "activate",
                      G_CALLBACK (plot_channel), "Channel 2");
    gtk_widget_add_accelerator (ch_2_item, "activate", accel_group,
```

```
                                GDK_2, GDK_CONTROL_MASK,
                                GTK_ACCEL_VISIBLE);

    ch_2_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (ch_2_item),
                                   ch_2_img);
    gtk_widget_show (ch_2_img);

/*************************************************************************/
/* Processing Menu                                                       */
/*************************************************************************/

processing_item = gtk_menu_item_new_with_mnemonic ("_Processing");
gtk_widget_show (processing_item);
gtk_container_add (GTK_CONTAINER (menu_bar), processing_item);

processing_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (processing_item),
                           processing_menu);

    /* HV Spectral Ratio Menu */

    hv_sp_item = gtk_image_menu_item_new_with_mnemonic
                ("_H/V Spectral Ratio");
    gtk_widget_show (hv_sp_item);
    gtk_container_add (GTK_CONTAINER (processing_menu), hv_sp_item);

    hv_sp_img = gtk_image_new_from_stock ("gtk-execute",
                                          GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (hv_sp_item),
                                   hv_sp_img);
    gtk_widget_show (hv_sp_img);

    hv_sp_menu = gtk_menu_new ();
    gtk_menu_item_set_submenu (GTK_MENU_ITEM (hv_sp_item), hv_sp_menu);

        /* Processing Selected item */

        selected_hv_item = gtk_menu_item_new_with_mnemonic
                           ("_Selected Item");
        g_signal_connect (GTK_WIDGET(selected_hv_item), "activate",
                          G_CALLBACK (console_window_show), "single");
        gtk_widget_show (selected_hv_item);
        gtk_container_add (GTK_CONTAINER (hv_sp_menu), selected_hv_item);

        /* Processing All items */

        all_hv_item = gtk_menu_item_new_with_mnemonic
                      ("_All Items");
        g_signal_connect (GTK_WIDGET(all_hv_item), "activate",
                          G_CALLBACK (console_window_show), "all");
        gtk_widget_show (all_hv_item);
        gtk_container_add (GTK_CONTAINER (hv_sp_menu), all_hv_item);

        /* Processing Average */

        average_hv_item = gtk_menu_item_new_with_mnemonic
                          ("_List Average");
        g_signal_connect (GTK_WIDGET(average_hv_item), "activate",
                          G_CALLBACK (console_window_show), "average");
        gtk_widget_show (average_hv_item);
        gtk_container_add (GTK_CONTAINER (hv_sp_menu), average_hv_item);
```

```
/************************************************************************/
/* Absolute Spectra Plot Menu                                           */
/************************************************************************/

as_plot_item = gtk_menu_item_new_with_mnemonic ("_Spectrum Plot");
gtk_widget_show (as_plot_item);
gtk_container_add (GTK_CONTAINER (menu_bar), as_plot_item);

as_plot_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (as_plot_item), as_plot_menu);

pixmap = gdk_pixmap_create_from_xpm_d (menu_bar->window, &mask,
         &style->bg[GTK_STATE_NORMAL],(gchar **) hv16x16_xpm);

    /* Channel 0 Absolute Spectrum Plot */

    as_0_plot_item = gtk_image_menu_item_new_with_mnemonic
                     ("Channel 0");
    gtk_widget_show (as_0_plot_item);
    gtk_container_add (GTK_CONTAINER (as_plot_menu), as_0_plot_item);
    g_signal_connect (GTK_WIDGET(as_0_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "Ch 0 Spectrum");
    gtk_widget_add_accelerator (as_0_plot_item, "activate", accel_group,
                                GDK_0, GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    as_0_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (as_0_plot_item),
                                   as_0_plot_img);
    gtk_widget_show (as_0_plot_img);

    /* Channel 1 Absolute Spectrum Plot */

    as_1_plot_item = gtk_image_menu_item_new_with_mnemonic
                     ("Channel 1");
    gtk_widget_show (as_1_plot_item);
    gtk_container_add (GTK_CONTAINER (as_plot_menu), as_1_plot_item);
    g_signal_connect (GTK_WIDGET(as_1_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "Ch 1 Spectrum");
    gtk_widget_add_accelerator (as_1_plot_item, "activate", accel_group,
                                GDK_1, GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    as_1_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (as_1_plot_item),
                                   as_1_plot_img);
    gtk_widget_show (as_1_plot_img);

    /* Channel 2 Absolute Spectrum Plot */

    as_2_plot_item = gtk_image_menu_item_new_with_mnemonic
                     ("Channel 2");
    gtk_widget_show (as_2_plot_item);
    gtk_container_add (GTK_CONTAINER (as_plot_menu), as_2_plot_item);
    g_signal_connect (GTK_WIDGET(as_2_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "Ch 2 Spectrum");
    gtk_widget_add_accelerator (as_2_plot_item, "activate", accel_group,
                                GDK_2, GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    as_2_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (as_2_plot_item),
                                   as_2_plot_img);
```

```
    gtk_widget_show (as_2_plot_img);

/*************************************************************************/
/* Spectral Ratio Plot Menu                                              */
/*************************************************************************/

sp_plot_item = gtk_menu_item_new_with_mnemonic ("_Spectral Ratio");
gtk_widget_show (sp_plot_item);
gtk_container_add (GTK_CONTAINER (menu_bar), sp_plot_item);

sp_plot_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (sp_plot_item), sp_plot_menu);

pixmap = gdk_pixmap_create_from_xpm_d (menu_bar->window, &mask,
         &style->bg[GTK_STATE_NORMAL],(gchar **) hv16x16_xpm);

    /* Average Spectral Ratio Plot */

    av_sp_plot_item = gtk_image_menu_item_new_with_mnemonic
                      ("_Average");
    gtk_widget_show (av_sp_plot_item);
    gtk_container_add (GTK_CONTAINER (sp_plot_menu), av_sp_plot_item);
    g_signal_connect (GTK_WIDGET(av_sp_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "Average H/V");
    gtk_widget_add_accelerator (av_sp_plot_item, "activate", accel_group,
                                GDK_0, GDK_CONTROL_MASK|GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    av_sp_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (av_sp_plot_item),
                                   av_sp_plot_img);
    gtk_widget_show (av_sp_plot_img);

    /* Channel 1 Spectral Ratio Plot */

    c1_sp_plot_item = gtk_image_menu_item_new_with_mnemonic
                      ("H(_1)/V");
    gtk_widget_show (c1_sp_plot_item);
    gtk_container_add (GTK_CONTAINER (sp_plot_menu), c1_sp_plot_item);
    g_signal_connect (GTK_WIDGET(c1_sp_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "H(1)/V");
    gtk_widget_add_accelerator (c1_sp_plot_item, "activate", accel_group,
                                GDK_1, GDK_CONTROL_MASK|GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    c1_sp_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (c1_sp_plot_item),
                                   c1_sp_plot_img);
    gtk_widget_show (c1_sp_plot_img);

    /* Channel 2 Spectral Ratio Plot */

    c2_sp_plot_item = gtk_image_menu_item_new_with_mnemonic
                      ("H(_2)/V");
    gtk_widget_show (c2_sp_plot_item);
    gtk_container_add (GTK_CONTAINER (sp_plot_menu), c2_sp_plot_item);
    g_signal_connect (GTK_WIDGET(c2_sp_plot_item), "activate",
                      G_CALLBACK (plot_hvsp), "H(2)/V");
    gtk_widget_add_accelerator (c2_sp_plot_item, "activate", accel_group,
                                GDK_2, GDK_CONTROL_MASK|GDK_MOD1_MASK,
                                GTK_ACCEL_VISIBLE);

    c2_sp_plot_img = gtk_image_new_from_pixmap (pixmap, mask);
```

```
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (c2_sp_plot_item),
                                   c2_sp_plot_img);
    gtk_widget_show (c2_sp_plot_img);

/*************************************************************************/
/* Help Menu                                                             */
/*************************************************************************/

help_item = gtk_menu_item_new_with_mnemonic ("_Help");
gtk_widget_show (help_item);
gtk_container_add (GTK_CONTAINER (menu_bar), help_item);

help_menu = gtk_menu_new ();
gtk_menu_item_set_submenu (GTK_MENU_ITEM (help_item), help_menu);

    /* GSesame Manual */

    gs_man_item = gtk_image_menu_item_new_with_mnemonic
                  ("GSesame _Manual");
    gtk_widget_show (gs_man_item);
    gtk_container_add (GTK_CONTAINER (help_menu), gs_man_item);
    gtk_widget_add_accelerator (gs_man_item, "activate", accel_group,
                                GDK_F1, 0,
                                GTK_ACCEL_VISIBLE);

    gs_man_img = gtk_image_new_from_stock ("gtk-help",
                                           GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (gs_man_item),
                                   gs_man_img);
    gtk_widget_show (gs_man_img);

    /* Separator */

    help_sep1 = gtk_separator_menu_item_new ();
    gtk_widget_show (help_sep1);
    gtk_container_add (GTK_CONTAINER (help_menu), help_sep1);
    gtk_widget_set_sensitive (help_sep1, FALSE);

    /* GSesame Manual */

    about_item = gtk_image_menu_item_new_with_mnemonic ("_About");
    gtk_widget_show (about_item);
    gtk_container_add (GTK_CONTAINER (help_menu), about_item);
    g_signal_connect (GTK_WIDGET(about_item), "activate",
                      G_CALLBACK (about_window_showhide), NULL);
    gtk_widget_add_accelerator (about_item, "activate", accel_group,
                                GDK_F2, 0,
                                GTK_ACCEL_VISIBLE);

    about_img = gtk_image_new_from_stock ("gtk-about",
                                          GTK_ICON_SIZE_MENU);
    gtk_image_menu_item_set_image (GTK_IMAGE_MENU_ITEM (about_item),
                                   about_img);
    gtk_widget_show (about_img);

/*************************************************************************/
/* End Menu                                                              */
/*************************************************************************/

/* Attach the new accelerator group to the window. */
gtk_window_add_accel_group (GTK_WINDOW (main_window), accel_group);

gtk_widget_show_all (menu_bar);
```

```
}
```

1.5 - Menu.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef MENU_H
#define MENU_H

    void    create_menu    (void);

# endif
```

1.6 - Toolbar.c

```
# include <gtk/gtk.h>

# include "globals.h"
# include "toolbar.h"
# include "project.h"
# include "list.h"
# include "saf.h"
# include "mainproc.h"
# include "plothv.h"
# include "warning.h"
# include "icons/hv24x24.xpm"

/*****************************************************************************/
/* CREATE_TOOLBAR:                                                           */
/* This function create a button toolbar for a fast functions recall.        */
/*****************************************************************************/

void create_toolbar (void)
{
    GtkWidget  *toolbar_main;
    GtkWidget  *toolbutton1;
    GtkWidget  *toolbutton2;
    GtkWidget  *toolbutton3;
    GtkWidget  *toolbutton4;
    GtkWidget  *toolbutton5;
    GtkWidget  *toolbutton6;
    GtkWidget  *toolbutton7;
    GtkWidget  *toolbutton8;
    GtkWidget  *toolbutton9;
    GtkWidget  *separatortoolitem1;
    GtkWidget  *separatortoolitem2;
    GtkWidget  *separatortoolitem3;
    GdkPixmap  *pixmap;
    GdkBitmap  *mask;
    GtkStyle   *style;
    GtkWidget  *tmp_image;
    GtkIconSize tmp_toolbar_icon_size;

    /*************************************************************************/
    /* Main Toolbar                                                          */
    /*************************************************************************/
```

```
toolbar_main = gtk_toolbar_new ();
gtk_widget_show (toolbar_main);
gtk_box_pack_start (GTK_BOX (vbox_main), toolbar_main, FALSE, TRUE, 0);
gtk_toolbar_set_style (GTK_TOOLBAR (toolbar_main), GTK_TOOLBAR_BOTH);
tmp_toolbar_icon_size = gtk_toolbar_get_icon_size
                        (GTK_TOOLBAR (toolbar_main));

/*************************************************************************/
/* New Project                                                           */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-new", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton1 = (GtkWidget*) gtk_tool_button_new (tmp_image, "New");
g_signal_connect (GTK_WIDGET(toolbutton1), "clicked",
                  G_CALLBACK (new_project), NULL);
gtk_widget_show (toolbutton1);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton1);

/*************************************************************************/
/* Open Project                                                          */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-open", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton2 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Open");
g_signal_connect (GTK_WIDGET(toolbutton2), "clicked",
                  G_CALLBACK (open_project), NULL);
gtk_widget_show (toolbutton2);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton2);

/*************************************************************************/
/* Save Project                                                          */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-save", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton3 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Save");
g_signal_connect (GTK_WIDGET(toolbutton3), "clicked",
                  G_CALLBACK (save_project), NULL);
gtk_widget_show (toolbutton3);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton3);

/*************************************************************************/
/* Separator                                                             */
/*************************************************************************/

separatortoolitem1 = (GtkWidget*) gtk_separator_tool_item_new ();
gtk_widget_show (separatortoolitem1);
gtk_container_add (GTK_CONTAINER (toolbar_main), separatortoolitem1);

/*************************************************************************/
/* Import SAF                                                            */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-add", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton4 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Add");
g_signal_connect (GTK_WIDGET(toolbutton4), "clicked",
                  G_CALLBACK (saf_file_selection), NULL);
gtk_widget_show (toolbutton4);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton4);
```

```
/*************************************************************************/
/* Delete SAF                                                            */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-delete", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton5 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Delete");
g_signal_connect (GTK_WIDGET(toolbutton5), "clicked",
                  G_CALLBACK (del_from_list), NULL);
gtk_widget_show (toolbutton5);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton5);

/*************************************************************************/
/* Clear list                                                            */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-clear", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton6 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Clear");
g_signal_connect (GTK_WIDGET(toolbutton6), "clicked",
                  G_CALLBACK (clear_list), NULL);
gtk_widget_show (toolbutton6);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton6);

/*************************************************************************/
/* Separator                                                             */
/*************************************************************************/

separatortoolitem2 = (GtkWidget*) gtk_separator_tool_item_new ();
gtk_widget_show (separatortoolitem2);
gtk_container_add (GTK_CONTAINER (toolbar_main), separatortoolitem2);

/*************************************************************************/
/* Item process                                                          */
/*************************************************************************/

tmp_image = gtk_image_new_from_stock ("gtk-execute", tmp_toolbar_icon_size);
gtk_widget_show (tmp_image);
toolbutton7 = (GtkWidget*) gtk_tool_button_new (tmp_image, "H/V");
g_signal_connect (GTK_WIDGET(toolbutton7), "clicked",
                  G_CALLBACK (console_window_show), "single");
gtk_widget_show (toolbutton7);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton7);

/*************************************************************************/
/* Spectral ratio plot                                                   */
/*************************************************************************/

style = gtk_widget_get_style(toolbar_main);
pixmap = gdk_pixmap_create_from_xpm_d (toolbar_main->window, &mask,
                                       &style->bg[GTK_STATE_NORMAL],
                                       (gchar **) hv24x24_xpm);
tmp_image = gtk_image_new_from_pixmap (pixmap, mask);
gtk_widget_show (tmp_image);
toolbutton8 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Plot");
g_signal_connect (GTK_WIDGET(toolbutton8), "clicked",
                  G_CALLBACK (plot_hvsp), "Average H/V");
gtk_widget_show (toolbutton8);
gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton8);

/*************************************************************************/
/* Separator                                                             */
/*************************************************************************/
```

```
    separatortoolitem3 = (GtkWidget*) gtk_separator_tool_item_new ();
    gtk_widget_show (separatortoolitem3);
    gtk_container_add (GTK_CONTAINER (toolbar_main), separatortoolitem3);

    /*****************************************************************************/
    /* Main quit                                                                 */
    /*****************************************************************************/

    tmp_image = gtk_image_new_from_stock ("gtk-quit", tmp_toolbar_icon_size);
    gtk_widget_show (tmp_image);
    toolbutton9 = (GtkWidget*) gtk_tool_button_new (tmp_image, "Exit");
    g_signal_connect (GTK_WIDGET(toolbutton9), "clicked",
                      G_CALLBACK (warning_unsaved_work), NULL);
    gtk_widget_show (toolbutton9);
    gtk_container_add (GTK_CONTAINER (toolbar_main), toolbutton9);


    gtk_widget_show_all (toolbar_main);
}
```

1.7 - Toolbar.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef TOOLBAR_H
#define TOOLBAR_H

    void    create_toolbar    (void);

# endif
```

1.8 - List.c

```
# include <gtk/gtk.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

# include "globals.h"
# include "list.h"
# include "platdep.h"

    /* SAF FIle Header's parameters enumeration:                  */
    /*    This kind of association between strings and progressive */
    /*    numbers make easy the list's elements referentiation.    */
    /*    The "SAF_POINTER" element refers to an invisible field   */
    /*    and is only used to connect the GTK list's element with  */
    /*    the associated saf file.                                 */
    enum
    {
        COLUMN_FILE_NAME,
        COLUMN_STA_CODE,
        COLUMN_START_TIME,
        COLUMN_SAMP_FREQ,
        COLUMN_NDAT,
        COLUMN_CH0_ID,
```

```
        COLUMN_CH1_ID,
        COLUMN_CH2_ID,
        COLUMN_UNITS,
        SAF_POINTER,
        COLUMNS
    };

/*****************************************************************************/
/* CREATE_LIST:                                                              */
/* This function create a clean GTK list structure and make it visible.      */
/* (No elements are still included)                                          */
/*****************************************************************************/

    void create_gtk_list (GtkWidget *vbox_main,gint argc, gchar *argv[])
{
    GtkWidget *scrolledwindow;
    GtkListStore *model;
    GtkTreeSelection *selection;
    GtkCellRenderer *renderer;

    /* Scrolled window container initialization */
    {
        scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
        gtk_box_pack_start (GTK_BOX (vbox_main), scrolledwindow, TRUE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (scrolledwindow), 5);
        gtk_scrolled_window_set_shadow_type(GTK_SCROLLED_WINDOW(scrolledwindow),
                                            GTK_SHADOW_IN);
        gtk_scrolled_window_set_policy (GTK_SCROLLED_WINDOW (scrolledwindow),
                                        GTK_POLICY_AUTOMATIC,
                                        GTK_POLICY_AUTOMATIC);
        gtk_widget_show (scrolledwindow);
    }

    /* List model initialization */
    model = gtk_list_store_new (COLUMNS,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_STRING,
                                G_TYPE_POINTER);

    /* Tree view initialization from previous model*/
    {
        view = gtk_tree_view_new_with_model (GTK_TREE_MODEL(model));
        selection = gtk_tree_view_get_selection (GTK_TREE_VIEW(view));

        gtk_tree_selection_set_mode (selection, GTK_SELECTION_SINGLE);
        gtk_tree_view_set_rules_hint (GTK_TREE_VIEW(view), TRUE);
    }

    /* Tree's visible columns definition */
    {
        renderer = gtk_cell_renderer_text_new ();
        gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                     "File",
                                                     renderer,
                                                     "text",
                                                     COLUMN_FILE_NAME,
```

```
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Station Code",
                                                renderer,
                                                "text",
                                                COLUMN_STA_CODE,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Start Time",
                                                renderer,
                                                "text",
                                                COLUMN_START_TIME,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Sampling Freqency",
                                                renderer,
                                                "text",
                                                COLUMN_SAMP_FREQ,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Samples Number",
                                                renderer,
                                                "text",
                                                COLUMN_NDAT,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Channel 0",
                                                renderer,
                                                "text",
                                                COLUMN_CH0_ID,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Channel 1",
                                                renderer,
                                                "text",
                                                COLUMN_CH1_ID,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Channel 2",
                                                renderer,
                                                "text",
                                                COLUMN_CH2_ID,
                                                NULL);

    renderer = gtk_cell_renderer_text_new ();
    gtk_tree_view_insert_column_with_attributes (GTK_TREE_VIEW(view),-1,
                                                "Units",
                                                renderer,
                                                "text",
```

```
                                                    COLUMN_UNITS,
                                                    NULL);
    }

    gtk_widget_show (view);
    g_object_unref (model);

    gtk_container_add (GTK_CONTAINER (scrolledwindow), view);

    /* Optional: function for alternate rows evidentiation */
    gtk_tree_view_set_rules_hint (GTK_TREE_VIEW (view), TRUE);
}

/*****************************************************************************/
/* ADD_TO_LIST:                                                              */
/* This function add an item to the GTK list, getting its values from the    */
/* "saf_file" structure.                                                     */
/*****************************************************************************/

void add_to_gtk_list (SAF_LIST_PTR new_saf)
{
    GtkTreeModel *model;
    GtkTreeIter iter;

    model = gtk_tree_view_get_model(GTK_TREE_VIEW(view));

    gtk_list_store_append(GTK_LIST_STORE(model), &iter);

    gtk_list_store_set (GTK_LIST_STORE(model), &iter,
            COLUMN_FILE_NAME,     saf_filename_extract(new_saf->file),
            COLUMN_STA_CODE,      new_saf->sta_code,
            COLUMN_START_TIME,    new_saf->start_time,
            COLUMN_SAMP_FREQ,     new_saf->samp_freq,
            COLUMN_NDAT,          new_saf->ndat,
            COLUMN_CH0_ID,        new_saf->ch0_id,
            COLUMN_CH1_ID,        new_saf->ch1_id,
            COLUMN_CH2_ID,        new_saf->ch2_id,
            COLUMN_UNITS,         new_saf->units,
            SAF_POINTER,          new_saf,
            -1);
}

/*****************************************************************************/
/* DEL_FROM_LIST:                                                            */
/* This function delete an item from the GTK list.                           */
/*****************************************************************************/

void del_from_list (void)
{
    GtkTreeModel *model;
    GtkTreeIter iter;
    GtkTreeSelection *selection;
    SAF_LIST_PTR ptr_del, ptr_list, ptr_tmp;

    model = gtk_tree_view_get_model(GTK_TREE_VIEW(view));
    selection = gtk_tree_view_get_selection(GTK_TREE_VIEW(view));

    if (gtk_tree_selection_get_selected(selection, NULL, &iter))
    {
        gtk_tree_model_get(model, &iter,
                           SAF_POINTER, &ptr_del,
                           -1);
```

```
        ptr_list = first_saf;

        if(ptr_list == ptr_del)
        {
            first_saf = first_saf->next_saf;
            list_item_mem_free(ptr_list);
        }
        else
        {
            while(ptr_list != ptr_del)
            {
                ptr_tmp = ptr_list;
                ptr_list = ptr_list->next_saf;
            }

            ptr_tmp->next_saf = ptr_list->next_saf;
            list_item_mem_free(ptr_list);
        }

        gtk_list_store_remove(GTK_LIST_STORE(model), &iter);
    }
}

/*****************************************************************************/
/* LIST_ITEM_MEM_FREE:                                                       */
/* This function frees memory of the headre's variables used in a list item; */
/* NOTE: it doesn't frees data sample's memory.                              */
/*****************************************************************************/

void list_item_mem_free (SAF_LIST_PTR ptr_list)
{
    if (ptr_list->file != NULL)
    {
        free(ptr_list->file);
    }

    if (ptr_list != NULL)
    {
        free(ptr_list);
    }
}

/*****************************************************************************/
/* CLEAR_LIST:                                                               */
/* This function clear each list's element freeing memory.                   */
/*****************************************************************************/

void clear_list (void)
{
    GtkTreeModel *model;
    SAF_LIST_PTR ptr_list, ptr_tmp;

    model = gtk_tree_view_get_model(GTK_TREE_VIEW(view));
    gtk_list_store_clear(GTK_LIST_STORE(model));

    {
        ptr_list = first_saf;

        while(ptr_list != NULL)
        {
            ptr_tmp = ptr_list->next_saf;
            list_item_mem_free(ptr_list);
            ptr_list = ptr_tmp;
```

```
        }
        //if(ptr_list != NULL)
        {
            //list_item_mem_free(ptr_tmp);    CONTROLLARE BENE!!!
            first_saf = NULL;
        }
    }
}
```

1.9 - List.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef LIST_H
#define LIST_H

    void create_gtk_list        (GtkWidget *vbox_main,
                                 int argc,
                                 char *argv[]);

    void add_to_gtk_list        (SAF_LIST_PTR new_saf);

    void del_from_list          (void);

    void list_item_mem_free     (SAF_LIST_PTR ptr_list);

    void clear_list             (void);

# endif
```

1.10 - Project.c

```
# include <gtk/gtk.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

# include "globals.h"
# include "gsesame.h"
# include "list.h"
# include "project.h"
# include "saf.h"
# include "winselcfg.h"
# include "hvproccfg.h"
# include "platdep.h"

/*****************************************************************************/
/* NEW_PROJECT:                                                              */
/* This function generates a dialog window that lets users to define/change  */
/* the project name. Note that any possible change is unsaved, so you have   */
/* to save it manually after.                                                */
/*****************************************************************************/

void new_project (void)
{
    GtkWidget *prj_new_dlg;
    GtkWidget *hbox;
```

```
    GtkWidget *stock_image;
    GtkWidget *label;
    GtkWidget *entry;
    const gchar *string;
    gint response;

    /* Dialog window */
    prj_new_dlg = gtk_dialog_new_with_buttons ("New Project",
                                               NULL,
                                               GTK_DIALOG_MODAL |
                                               GTK_DIALOG_DESTROY_WITH_PARENT,
                                               GTK_STOCK_APPLY,
                                               GTK_RESPONSE_ACCEPT,
                                               GTK_STOCK_CANCEL,
                                               GTK_RESPONSE_REJECT,
                                               NULL);

    /* Horizontal box */
    {
        hbox = gtk_hbox_new (FALSE,10);
        gtk_container_set_border_width (GTK_CONTAINER(hbox),10);
        gtk_box_pack_start (GTK_BOX(GTK_DIALOG(prj_new_dlg)->vbox),
                            hbox,
                            FALSE,
                            FALSE,
                            0);
    }

    /* Dialog question stock image and label */
    {
        stock_image = gtk_image_new_from_stock (GTK_STOCK_DIALOG_QUESTION,
                                                GTK_ICON_SIZE_DIALOG);
        gtk_box_pack_start (GTK_BOX(hbox),stock_image,FALSE,FALSE,0);

        label = gtk_label_new_with_mnemonic ("Enter Project Name:");
        gtk_box_pack_start (GTK_BOX(hbox),label,FALSE,FALSE,0);
    }

    /* Entry widget */
    {
        entry = gtk_entry_new ();
        gtk_entry_set_text (GTK_ENTRY(entry), project.name);
        gtk_box_pack_start (GTK_BOX(hbox),entry,FALSE,FALSE,0);
    }

    gtk_widget_show_all (prj_new_dlg);

    /* Action evaluation control:                                    */
    /* if the OK button is pressed, the project name is redifined. */
    {
        response = gtk_dialog_run (GTK_DIALOG(prj_new_dlg));

        if (response == GTK_RESPONSE_ACCEPT)
        {
            string = gtk_entry_get_text (GTK_ENTRY(entry));
            strcpy (project.name,string);
            set_project_title ();
            set_project_file ();
        }
    }

    gtk_widget_destroy (prj_new_dlg);
}
```

```
/*****************************************************************************/
/* SAVE_PROJECT:                                                             */
/* This function save the project by writing a file in which are stored all  */
/* the project parameters.                                                   */
/* It uses the "project file name" global variable as target:                */
/* if the file exists it is overwritten, otherwise is created by calling the */
/* SAVE_PROJECT_AS function.                                                 */
/*****************************************************************************/

void save_project (void)
{
    FILE *fp;

    if((fp = fopen (project.file, "r")) != NULL)
    {
        fclose (fp);
        write_project_file ();
    }
    else
    {
        save_project_as ();
    }
}

/*****************************************************************************/
/* SAVE_PROJECT_AS:                                                          */
/* This function save the project by writing a file in which are stored all  */
/* the project parameters.                                                   */
/* It uses the "project file name" global variable, but allows user to change */
/* the target with a simple dialog window; if the file exists than the window */
/* ask for owerwrite.                                                        */
/*****************************************************************************/

void save_project_as (void)
{
    GtkWidget *prj_selection;
    gchar fileseldef[200];

    /* File selection dialog window */
    prj_selection = gtk_file_selection_new ("Save Project");

    /* Set the default filename */
    {
        sprintf(fileseldef, "%s%s", prj_path, project.file);

        gtk_file_selection_set_filename (GTK_FILE_SELECTION(prj_selection),
                                         fileseldef);
    }

    /* OK button signal connection */
    {
        g_signal_connect (GTK_FILE_SELECTION(prj_selection)->ok_button,
                          "clicked",
                          G_CALLBACK(reset_project_file),
                          prj_selection);

        g_signal_connect (GTK_FILE_SELECTION(prj_selection)->ok_button,
                          "clicked",
                          G_CALLBACK(ask_for_overwrite_project),
                          prj_selection);
    }
```

```
    /* Cancel button signal connection */
    g_signal_connect_swapped (GTK_FILE_SELECTION(prj_selection)->cancel_button,
                              "clicked",
                              G_CALLBACK(gtk_widget_destroy),
                              prj_selection);

    gtk_widget_show (prj_selection);
}

/*****************************************************************************/
/* ASK_FOR_OVERWRITE_PROJECT:                                                */
/* This function open a small dialog window that ask user for overwrite      */
/* (or not) the current project file, if exists.                             */
/* If not, write the project in the selected file.                           */
/*****************************************************************************/

void ask_for_overwrite_project(GtkWidget *widget, gpointer abstrptr)
{
    GtkWidget *file_selector = GTK_WIDGET(abstrptr);
    GtkWidget *overwrite;
    GtkWidget *hbox;
    GtkWidget *stock_image;
    GtkWidget *label;
    gint response;
    FILE *fp;

    /* File exists */
    if ((fp = fopen(project.file, "r")) != NULL)
    {
        /* Dialog window */
        overwrite = gtk_dialog_new_with_buttons ("Warning!",
                                                 NULL,
                                                 GTK_DIALOG_MODAL |
                                                 GTK_DIALOG_DESTROY_WITH_PARENT,
                                                 GTK_STOCK_APPLY,
                                                 GTK_RESPONSE_ACCEPT,
                                                 GTK_STOCK_CANCEL,
                                                 GTK_RESPONSE_REJECT,
                                                 NULL);

        /* Horizontal box */
        {
            hbox = gtk_hbox_new (FALSE,10);
            gtk_container_set_border_width (GTK_CONTAINER(hbox),10);
            gtk_box_pack_start (GTK_BOX(GTK_DIALOG(overwrite)->vbox),
                                hbox,FALSE,FALSE,0);
        }

        /* Dialog question stock image and label */
        {
            stock_image = gtk_image_new_from_stock (GTK_STOCK_DIALOG_QUESTION,
                                                    GTK_ICON_SIZE_DIALOG);
            gtk_box_pack_start (GTK_BOX(hbox),stock_image,FALSE,FALSE,0);

            label = gtk_label_new_with_mnemonic ("File exists: overwrite?");
            gtk_box_pack_start (GTK_BOX(hbox),label,FALSE,FALSE,0);
        }

        gtk_widget_show_all (overwrite);

        /* Response evaluation */
        {
        response = gtk_dialog_run (GTK_DIALOG(overwrite));
```

```
            if (response == GTK_RESPONSE_ACCEPT)
            {
                write_project_file ();
                gtk_widget_destroy (GTK_WIDGET(file_selector));
            }
        }

        gtk_widget_destroy (overwrite);
        fclose (fp);
    }

    /* File doesn't exist */
    else
    {
        write_project_file ();
        gtk_widget_destroy (GTK_WIDGET(file_selector));
    }
}

/*****************************************************************************/
/* OPEN_PROJECT:                                                             */
/* This function open a fileselection dialog window to open an existing      */
/* project file.                                                             */
/*****************************************************************************/

void open_project (void)
{
    GtkWidget *prj_selection;

    /* File selection dialog window */
    prj_selection = gtk_file_selection_new ( "Open Project");

    /* Set the default target */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(prj_selection),
                                     prj_path);

    /* Signal connection */
    {
        g_signal_connect (GTK_FILE_SELECTION (prj_selection)->ok_button,
                          "clicked",
                          G_CALLBACK(read_project_file),
                          prj_selection);

        g_signal_connect_swapped
                         (GTK_FILE_SELECTION(prj_selection)->cancel_button,
                          "clicked",
                          G_CALLBACK(gtk_widget_destroy),
                          prj_selection);
    }

    gtk_widget_show (prj_selection);
}

/*****************************************************************************/
/* WRITE_PROJECT_FILE:                                                       */
/* This function save the project informations (name and saf file list) in a */
/* ASCII project file.                                                       */
/*****************************************************************************/

void write_project_file (void)
{
    FILE *fp;
```

```
SAF_LIST_PTR ptr_list;
gchar *string[300];
gint count = 0;
gint i;

/* Open project file */
if ((fp = fopen(project.file, "w+")) != NULL);
{
    /* File header */
    fprintf(fp,"GSESAME PROJECT FILE\n");

    /* Project name */
    fprintf(fp,"NAME = ");
    fprintf(fp,"%s\n",project.name);

    /* Winselection parameters */
    fprintf(fp,"WSPAR = ");
    fprintf(fp,"%.2f ",confpar.stalen);
    fprintf(fp,"%.2f ",confpar.ltalen);
    fprintf(fp,"%.2f ",confpar.minstalta);
    fprintf(fp,"%.2f ",confpar.maxstalta);
    fprintf(fp,"%.2f ",confpar.winlen);
    fprintf(fp,"%.2f ",confpar.overlap);
    fprintf(fp,"%.2f ",confpar.tolerance);
    fprintf(fp,"%d ",confpar.saturation);
    fprintf(fp,"%d\n",confpar.noisywin);

    /* HVProcessing parameters */
    fprintf(fp,"HVPAR = ");
    fprintf(fp,"%d ",confpar.freqsp_type);
    fprintf(fp,"%.2f ",confpar.freqsp_min);
    fprintf(fp,"%.2f ",confpar.freqsp_max);
    fprintf(fp,"%.2f ",confpar.freqsp_npnt);
    fprintf(fp,"%d ",confpar.offrem_type);
    fprintf(fp,"%.2f ",confpar.offrem_freq);
    fprintf(fp,"%d ",confpar.tape_type);
    fprintf(fp,"%.2f ",confpar.tape_perc);
    fprintf(fp,"%d ",confpar.smth_type);
    fprintf(fp,"%.2f ",confpar.smth_band);
    fprintf(fp,"%d ",confpar.smth_wgh);
    fprintf(fp,"%d ",confpar.merging);
    fprintf(fp,"%d\n",confpar.outsinwin);

    /* SAF file list:                        */
    /* the list structure is reorganized to */
    /* be saved in the correct order.       */
    if (first_saf != NULL)
    {
        ptr_list = first_saf;

        while (ptr_list != NULL)
        {
            string[count] = ptr_list->file;
            ptr_list = ptr_list->next_saf;
            count++;
        }
        for (i=1;i<=count;i++)
        {
            fprintf(fp,"SAF = ");
            fprintf(fp,"%s\n",string[count - i]);
        }
    }
```

```
        /* Close file */
        fclose(fp);
    }
}

/*****************************************************************************/
/* READ_PROJECT_FILE:                                                        */
/* this function read an existing project file and import its informations.  */
/*****************************************************************************/

void read_project_file (GtkWidget *widget, gpointer abstrptr)
{
    gchar *header = "SESAME ASCII data format (saf) v. 1";
    GtkWidget *file_selector = GTK_WIDGET(abstrptr);
    const gchar *selected_filename;
    gchar *extract = NULL;
    gchar saf_string[50];
    gchar string[200];
    FILE *fp, *saf_fp;


    /* Clear each previous list item */
    clear_list();

    /* Open project file */
    selected_filename = gtk_file_selection_get_filename
                        (GTK_FILE_SELECTION(file_selector));

    if((fp = fopen(selected_filename, "r")) != NULL)
    {
        /* Check if the file is a project file; if so */
        /* close the file selection.                  */
        fgets(string,200,fp);

        if(strstr(string,"GSESAME PROJECT FILE") == string)
        {
            gtk_widget_destroy(GTK_WIDGET(file_selector));

            /* Redefine the project.name variable */
            strcpy(project.file,selected_filename);

            while (!feof(fp))
            {
                fgets(string,200,fp);

                extract = string_extract(fp, string);

                /* Redefine the project.name variable */
                /* and the project title              */
                if (strstr(string,"NAME") == string)
                {
                    strcpy(project.name,extract);
                    set_project_title();
                }

                /* Winselection parameters */
                if (strstr(string,"WSPAR") == string)
                {
                    sscanf(extract, "%lf%lf%lf%lf%lf%lf%lf%d%d",
                    &confpar.stalen,
                    &confpar.ltalen,
                    &confpar.minstalta,
                    &confpar.maxstalta,
```

```
                    &confpar.winlen,
                    &confpar.overlap,
                    &confpar.tolerance,
                    &confpar.saturation,
                    &confpar.noisywin);

                    ws_undo_value (NULL, NULL);
                }

                /* HVProcessing parameters */
                if (strstr(string,"HVPAR") == string)
                {
                    sscanf(string,"%d%lf%lf%lf%d%lf%d%lf%d%lf%d%d%d",
                    &confpar.freqsp_type,
                    &confpar.freqsp_min,
                    &confpar.freqsp_max,
                    &confpar.freqsp_npnt,
                    &confpar.offrem_type,
                    &confpar.offrem_freq,
                    &confpar.tape_type,
                    &confpar.tape_perc,
                    &confpar.smth_type,
                    &confpar.smth_band,
                    &confpar.smth_wgh,
                    &confpar.merging,
                    &confpar.outsinwin);

                    hv_undo_value (NULL, NULL);
                }

                /* Get the saf file item */
                if (strstr(string,"SAF") == string)
                {
                    /* Check if the file exist. */
                    if((saf_fp = fopen(extract, "r")) != NULL)
                    {
                        fgets(saf_string,50,saf_fp);
                        fclose(saf_fp);

                        /* Check if the file is a SAF file. */
                        if
                        (strstr(saf_string,header) == saf_string)
                        {
                            add_to_c_list ((const gchar *)extract);
                            add_to_gtk_list (first_saf);
                        }
                    }
                }

                /* Clear the string for the next cicle */
                strcpy(string,"*****");
                extract = string;
            }
        }
        fclose(fp);
    }
}

gchar *string_extract (FILE *fp, gchar *string)
{
    gchar *newline_ptr;
    gchar *extract = NULL;
```

```
    /* Control for a line terminator character: */
    /* if present, it's substituted whit a      */
    /* string terminator character.             */

    newline_ptr = strchr (string,'\n');
    if (newline_ptr != NULL)
    {
        newline_ptr[0] = '\0';
    }

    newline_ptr = strchr (string,'\r');
    if (newline_ptr != NULL)
    {
        newline_ptr[0] = '\0';
    }

    /* Getting string information whitout identifier:     */
    /* any eventual white space at the begin is omitted   */

    extract = strchr(string, '=');

    if (extract != NULL)
    {
        extract = extract + sizeof(char);

        while ((extract[0] == ' ') ||
               (extract[0] == '\0'))
        {
            extract = extract + sizeof(char);
        }
    }
    else
    {
        extract = string;
    }

    return extract;
}
```

1.11 - Project.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef PROJECT_H
#define PROJECT_H

    void    new_project                (void);

    void    save_project               (void);

    void    save_project_as            (void);

    void    ask_for_overwrite_project  (GtkWidget *widget,
                                        gpointer abstrptr);

    void    open_project               (void);

    void    write_project_file         (void);
```

```
    void    read_project_file          (GtkWidget *widget,
                                        gpointer abstrptr);

    gchar  *string_extract             (FILE *fp,
                                        gchar *string);

# endif
```

1.12 - Saf.c

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <gtk/gtk.h>

# include "globals.h"
# include "project.h"
# include "list.h"
# include "saf.h"
# include "platdep.h"

/*****************************************************************************/
/* SAF_FILE_SELECTION:                                                       */
/* This function open a fileselection dialog window to open a saf file       */
/* through the SAF_FILE_OPEN callback.                                       */
/*****************************************************************************/

void saf_file_selection (void)
{
    GtkWidget *saf_selection;

    /* File selection dialog window */
    saf_selection = gtk_file_selection_new ("Open SAF file");

    /* Set the default target */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(saf_selection),
                                     saf_path);

    /* Signal connection */
    {
        g_signal_connect (GTK_FILE_SELECTION(saf_selection)->ok_button,
                          "clicked",
                          G_CALLBACK(saf_file_open),
                          saf_selection);

        g_signal_connect_swapped
                         (GTK_FILE_SELECTION(saf_selection)->cancel_button,
                          "clicked",
                          G_CALLBACK(gtk_widget_destroy),
                          saf_selection);
    }

    gtk_widget_show (saf_selection);
}

/*****************************************************************************/
/* SAF_FILE_OPEN:                                                            */
/* This callback function checks for a saf file; if it exists the function   */
/* inport its name and header. Than close the file.                          */
/*****************************************************************************/
```

```
void saf_file_open (GtkWidget *widget, gpointer abstrptr)
{
    FILE *fp;
    gchar header_string[] = "SESAME ASCII data format (saf) v. 1";
    gchar control_string[100];
    const gchar *selected_filename;
    GtkWidget *file_selector = GTK_WIDGET(abstrptr);

    /* Get the saf file name from file selection widget */
    selected_filename = gtk_file_selection_get_filename
                        (GTK_FILE_SELECTION(file_selector));

    /* Check if the file exists */
    if((fp = fopen (selected_filename, "r")) != NULL)
    {
        fgets (control_string,50,fp);
        fclose (fp);

        /* Check if the file is a saf file; if so, close the */
        /* file selection and create a list item.            */
        if (strstr (control_string, header_string) == control_string)
        {
            gtk_widget_destroy (GTK_WIDGET(file_selector));
            add_to_c_list (selected_filename);
            add_to_gtk_list (first_saf);
        }
    }
}

/*****************************************************************************/
/* ADD_TO_C_LIST:                                                            */
/* This function first creates a new item in the internal c list and then    */
/* add the item to the gtk list through the ADD_TO_GTK_LIST function.        */
/*****************************************************************************/

void add_to_c_list (const gchar *selected_filename)
{
    gint strsize;
    SAF_LIST_PTR new_saf;

    /* Initializie the new list item */
    {
        new_saf = NULL;
        new_saf = (SAF_LIST_PTR) malloc (sizeof(SAF_LIST));
    }

    /* Get filename and header information */
    {
        strsize = sizeof (gchar) * (strlen(selected_filename) + 1);
        new_saf->file = (gchar *) malloc (strsize);
        strcpy (new_saf->file,selected_filename);

        /* Import information from header */
        saf_file_header_read (selected_filename, new_saf);
    }

    /* Add item to c list */
    {
        new_saf->next_saf = first_saf;
        first_saf = new_saf;
    }
}
```

```
/*****************************************************************************/
/* SAF_FILE_HEADER_READ:                                                     */
/* This function reads the header of a saf file extracting only the usefull  */
/* information.                                                              */
/*****************************************************************************/

void *saf_file_header_read (const gchar *selected_filename,SAF_LIST_PTR new_saf)
{
    FILE *fp;
    char string[100];
    char *extract = NULL;

    /* Open saf file */
    if((fp = fopen (selected_filename, "r")) != NULL)
    {
        /* Getting header information untill the data separator is found */
        do
        {
            fgets (string,100,fp);

            extract = string_extract (fp, string);

            /* Saving information in the structure */
            {
                if (strstr(string,"STA_CODE") == string)
                {
                    strcpy(new_saf->sta_code,extract);
                }
                if (strstr(string,"START_TIME") == string)
                {
                    strcpy(new_saf->start_time,extract);
                }
                if (strstr(string,"SAMP_FREQ") == string)
                {
                    strcpy(new_saf->samp_freq,extract);
                }
                if (strstr(string,"NDAT") == string)
                {
                    strcpy(new_saf->ndat,extract);
                }
                if (strstr(string,"CH0_ID") == string)
                {
                    strcpy(new_saf->ch0_id,extract);
                }
                if (strstr(string,"CH1_ID") == string)
                {
                    strcpy(new_saf->ch1_id,extract);
                }
                if (strstr(string,"CH2_ID") == string)
                {
                    strcpy(new_saf->ch2_id,extract);
                }
                if (strstr(string,"UNITS") == string)
                {
                    strcpy(new_saf->units,extract);
                }
            }
        }
        while(strstr(string,"####----") != string);

    fclose (fp);
    }
```

```
    return new_saf;
}
```

1.13 - Saf.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef SAF_H
#define SAF_H

    void saf_file_selection         (void);

    void saf_file_open              (GtkWidget *widget,
                                     gpointer abstrptr);

    void add_to_c_list              (const gchar *selected_filename);

    void *saf_file_header_read      (const gchar *selected_filename,
                                     SAF_LIST_PTR new_saf);

# endif
```

1.14 - Winselcfg.c

```
# include <gtk/gtk.h>

# include "globals.h"
# include "winselcfg.h"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

GtkWidget *ws_window;
GtkWidget *ws_vbox1;
GtkWidget *ws_vbox2;
GtkWidget *ws_vbox3;
GtkWidget *ws_frame1;
GtkWidget *ws_table1;
GtkWidget *ws_label0;
GtkWidget *ws_label1;
GtkWidget *ws_label2;
GtkWidget *ws_label3;
GtkWidget *ws_label4;
GtkWidget *ws_label5;
GtkWidget *ws_label6;
GtkWidget *ws_label7;
GtkWidget *ws_label8;
GtkWidget *ws_spinbutton1;
GtkWidget *ws_spinbutton2;
GtkWidget *ws_spinbutton3;
GtkWidget *ws_spinbutton4;
GtkWidget *ws_spinbutton5;
GtkWidget *ws_spinbutton6;
GtkWidget *ws_spinbutton7;
GtkObject *ws_spinbutton1_adj;
GtkObject *ws_spinbutton2_adj;
```

```
GtkObject *ws_spinbutton3_adj;
GtkObject *ws_spinbutton4_adj;
GtkObject *ws_spinbutton5_adj;
GtkObject *ws_spinbutton6_adj;
GtkObject *ws_spinbutton7_adj;
GtkWidget *ws_hseparator1;
GtkWidget *ws_checkbutton1;
GtkWidget *ws_checkbutton2;
GtkWidget *ws_hbuttonbox1;
GtkWidget *ws_button1;
GtkWidget *ws_button2;
GtkWidget *ws_button3;
GtkWidget *ws_alignment1;
GtkWidget *ws_alignment2;
GtkWidget *ws_alignment3;
GtkWidget *ws_hbox1;
GtkWidget *ws_hbox2;
GtkWidget *ws_hbox3;
GtkWidget *ws_image1;
GtkWidget *ws_image2;
GtkWidget *ws_image3;
GtkWidget *ws_button_label1;
GtkWidget *ws_button_label2;
GtkWidget *ws_button_label3;

/*****************************************************************************/
/* WS_WINDOW_SHOWHIDE:                                                       */
/*****************************************************************************/

void ws_window_showhide (void)
{
    if (GTK_WIDGET_VISIBLE(ws_window))
    {
         gtk_widget_hide (ws_window);
    }
    else
    {
        gtk_widget_show (ws_window);
    }
}

/*****************************************************************************/
/* CREATE_WS_PAR_WINDOW:                                                     */
/*****************************************************************************/

void create_ws_par_window (void)
{
    /* WS patameters configuration window */
    {
        ws_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title (GTK_WINDOW (ws_window), ("Configurations"));
        gtk_window_set_resizable (GTK_WINDOW (ws_window), FALSE);
        gtk_window_move (GTK_WINDOW (ws_window), 50, 400);
        g_signal_connect (GTK_OBJECT (ws_window), "delete_event",
                          G_CALLBACK (ws_window_showhide), NULL);

        ws_vbox1 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (ws_vbox1);
        gtk_container_add (GTK_CONTAINER (ws_window), ws_vbox1);
    }

    /* Frame */
    {
```

```
    ws_frame1 = gtk_frame_new (NULL);
    gtk_widget_show (ws_frame1);
    gtk_box_pack_start (GTK_BOX (ws_vbox1), ws_frame1, TRUE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (ws_frame1), 5);

    ws_label0 = gtk_label_new (("<b>Window Selection Parameters</b>"));
    gtk_widget_show (ws_label0);
    gtk_frame_set_label_widget (GTK_FRAME (ws_frame1), ws_label0);
    gtk_label_set_use_markup (GTK_LABEL (ws_label0), TRUE);
    gtk_misc_set_alignment (GTK_MISC (ws_label0), 0.43, 0.52);

    ws_vbox2 = gtk_vbox_new (FALSE, 0);
    gtk_widget_show (ws_vbox2);
    gtk_container_add (GTK_CONTAINER (ws_frame1), ws_vbox2);

    ws_table1 = gtk_table_new (7, 2, FALSE);
    gtk_widget_show (ws_table1);
    gtk_box_pack_start (GTK_BOX (ws_vbox2), ws_table1, TRUE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (ws_table1), 5);
    gtk_table_set_col_spacings (GTK_TABLE (ws_table1), 5);
}

/* Window length for STA (in seconds) */
{
    ws_label1 = gtk_label_new
                (("Window length for STA (in seconds):"));
    gtk_widget_show (ws_label1);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label1, 0, 1, 0, 1,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label1), 0, 0.5);

    ws_spinbutton1_adj = gtk_adjustment_new
                         (0.0, 0.5, 2.0, 0.01, 10, 10);
    ws_spinbutton1 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton1_adj), 1, 2);
    gtk_widget_show (ws_spinbutton1);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton1, 1, 2, 0, 1,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Window length for LTA (in seconds) */
{
    ws_label2 = gtk_label_new
                (("Window length for LTA (in seconds):"));
    gtk_widget_show (ws_label2);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label2, 0, 1, 1, 2,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label2), 0, 0.5);

    ws_spinbutton2_adj = gtk_adjustment_new
                         (0.0, 20.0, 100.0, 0.01, 10, 10);
    ws_spinbutton2 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton2_adj), 1, 2);
    gtk_widget_show (ws_spinbutton2);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton2, 1, 2, 1, 2,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Minimum level for STA/LTA threshold */
```

```
{
    ws_label3 = gtk_label_new
                ((("Minimum level for STA/LTA threshold:"));
    gtk_widget_show (ws_label3);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label3, 0, 1, 2, 3,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label3), 0, 0.5);

    ws_spinbutton3_adj = gtk_adjustment_new
                         (0.0, 0.1, 0.75, 0.01, 10, 10);
    ws_spinbutton3 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton3_adj), 1, 2);
    gtk_widget_show (ws_spinbutton3);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton3, 1, 2, 2, 3,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Maximum level for STA/LTA threshold */
{
    ws_label4 = gtk_label_new
                ((("Maximum level for STA/LTA threshold:"));
    gtk_widget_show (ws_label4);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label4, 0, 1, 3, 4,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label4), 0, 0.5);

    ws_spinbutton4_adj = gtk_adjustment_new
                         (0.0, 1.5, 2.5, 0.01, 10, 10);
    ws_spinbutton4 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton4_adj), 1, 2);
    gtk_widget_show (ws_spinbutton4);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton4, 1, 2, 3, 4,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Window length (in seconds) */
{
    ws_label5 = gtk_label_new
                ((("Window length (in seconds):"));
    gtk_widget_show (ws_label5);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label5, 0, 1, 4, 5,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label5), 0, 0.5);

    ws_spinbutton5_adj = gtk_adjustment_new
                         (0.0, 10.0, 100.0, 0.01, 10, 10);
    ws_spinbutton5 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton5_adj), 1, 2);
    gtk_widget_show (ws_spinbutton5);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton5, 1, 2, 4, 5,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Overlap percentage for selected windows */
{
    ws_label6 = gtk_label_new
                ((("Overlap percentage for selected windows:"));
```

```
    gtk_widget_show (ws_label6);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label6, 0, 1, 5, 6,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label6), 0, 0.5);

    ws_spinbutton6_adj = gtk_adjustment_new
                         (0.0, 0.0, 50.0, 0.01, 10, 10);
    ws_spinbutton6 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton6_adj), 1, 2);
    gtk_widget_show (ws_spinbutton6);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton6, 1, 2, 5, 6,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Tolerance level for bad points (in seconds) */
{
    ws_label7 = gtk_label_new
                (("Tolerance level for bad points (in seconds):"));
    gtk_widget_show (ws_label7);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_label7, 0, 1, 6, 7,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (ws_label7), 0, 0.5);

    ws_spinbutton7_adj = gtk_adjustment_new
                         (0.0, 0.0, 0.2, 0.01, 10, 10);
    ws_spinbutton7 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (ws_spinbutton7_adj), 1, 2);
    gtk_widget_show (ws_spinbutton7);
    gtk_table_attach (GTK_TABLE (ws_table1), ws_spinbutton7, 1, 2, 6, 7,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Separator */
{
    ws_hseparator1 = gtk_hseparator_new ();
    gtk_widget_show (ws_hseparator1);
    gtk_box_pack_start (GTK_BOX (ws_vbox2), ws_hseparator1, TRUE, TRUE, 0);
}

{
    ws_vbox3 = gtk_vbox_new (FALSE, 0);
    gtk_widget_show (ws_vbox3);
    gtk_box_pack_start (GTK_BOX (ws_vbox2), ws_vbox3, TRUE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (ws_vbox3), 5);
}

/* Saturation check */
{
    ws_checkbutton1 = gtk_check_button_new_with_mnemonic
                      (("Saturation check"));
    gtk_widget_show (ws_checkbutton1);
    gtk_box_pack_start (GTK_BOX(ws_vbox3),ws_checkbutton1,FALSE,FALSE,0);
}

/* Removal of very noisy window */
{
    ws_checkbutton2 = gtk_check_button_new_with_mnemonic
                      (("Removal of very noisy window"));
    gtk_widget_show (ws_checkbutton2);
```

```
    gtk_box_pack_start (GTK_BOX(ws_vbox3),ws_checkbutton2,FALSE,FALSE,0);
}

/* Button box */
{
    ws_hbuttonbox1 = gtk_hbutton_box_new ();
    gtk_widget_show (ws_hbuttonbox1);
    gtk_box_pack_start (GTK_BOX (ws_vbox1), ws_hbuttonbox1, FALSE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (ws_hbuttonbox1), 5);
    gtk_button_box_set_layout (GTK_BUTTON_BOX (ws_hbuttonbox1),
                               GTK_BUTTONBOX_SPREAD);
}

/* Apply button */
{
    ws_button1 = gtk_button_new ();
    g_signal_connect (G_OBJECT (ws_button1), "clicked",
                      G_CALLBACK (ws_apply_value), NULL);
    gtk_widget_show (ws_button1);
    gtk_container_add (GTK_CONTAINER (ws_hbuttonbox1), ws_button1);
    GTK_WIDGET_SET_FLAGS (ws_button1, GTK_CAN_DEFAULT);

    ws_alignment1 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (ws_alignment1);
    gtk_container_add (GTK_CONTAINER (ws_button1), ws_alignment1);

    ws_hbox1 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (ws_hbox1);
    gtk_container_add (GTK_CONTAINER (ws_alignment1), ws_hbox1);

    ws_image1 = gtk_image_new_from_stock
                ("gtk-apply", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (ws_image1);
    gtk_box_pack_start (GTK_BOX (ws_hbox1), ws_image1, FALSE, FALSE, 0);

    ws_button_label1 = gtk_label_new_with_mnemonic ("_Apply");
    gtk_widget_show (ws_button_label1);
    gtk_box_pack_start (GTK_BOX (ws_hbox1), ws_button_label1,
                        FALSE, FALSE, 0);
}

/* Cancel button */
{
    ws_button2 = gtk_button_new ();
    g_signal_connect (G_OBJECT (ws_button2), "clicked",
                      G_CALLBACK (ws_undo_value), NULL);
    gtk_widget_show (ws_button2);
    gtk_container_add (GTK_CONTAINER (ws_hbuttonbox1), ws_button2);
    GTK_WIDGET_SET_FLAGS (ws_button2, GTK_CAN_DEFAULT);

    ws_alignment2 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (ws_alignment2);
    gtk_container_add (GTK_CONTAINER (ws_button2), ws_alignment2);

    ws_hbox2 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (ws_hbox2);
    gtk_container_add (GTK_CONTAINER (ws_alignment2), ws_hbox2);

    ws_image2 = gtk_image_new_from_stock
                ("gtk-cancel", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (ws_image2);
    gtk_box_pack_start (GTK_BOX (ws_hbox2), ws_image2, FALSE, FALSE, 0);
```

```
        ws_button_label2 = gtk_label_new_with_mnemonic ("_Cancel");
        gtk_widget_show (ws_button_label2);
        gtk_box_pack_start (GTK_BOX (ws_hbox2), ws_button_label2,
                            FALSE, FALSE, 0);
    }

    /* Default button */
    {
        ws_button3 = gtk_button_new ();
        g_signal_connect (G_OBJECT (ws_button3), "clicked",
                          G_CALLBACK (ws_default_value), NULL);
        gtk_widget_show (ws_button3);
        gtk_container_add (GTK_CONTAINER (ws_hbuttonbox1), ws_button3);
        GTK_WIDGET_SET_FLAGS (ws_button3, GTK_CAN_DEFAULT);

        ws_alignment3 = gtk_alignment_new (0.5, 0.5, 0, 0);
        gtk_widget_show (ws_alignment3);
        gtk_container_add (GTK_CONTAINER (ws_button3), ws_alignment3);

        ws_hbox3 = gtk_hbox_new (FALSE, 2);
        gtk_widget_show (ws_hbox3);
        gtk_container_add (GTK_CONTAINER (ws_alignment3), ws_hbox3);

        ws_image3 = gtk_image_new_from_stock
                    ("gtk-undo", GTK_ICON_SIZE_BUTTON);
        gtk_widget_show (ws_image3);
        gtk_box_pack_start (GTK_BOX (ws_hbox3), ws_image3, FALSE, FALSE, 0);

        ws_button_label3 = gtk_label_new_with_mnemonic ("_Default");
        gtk_widget_show (ws_button_label3);
        gtk_box_pack_start (GTK_BOX (ws_hbox3), ws_button_label3,
                            FALSE, FALSE, 0);
    }

    /* Setting default value */
    ws_default_value (NULL, NULL);

    gtk_widget_show (ws_window);
}

/*****************************************************************************/
/* WS_PAR_INIT:                                                              */
/*****************************************************************************/

void ws_par_init (void)
{
    confpar.stalen_def      = 1;
    confpar.ltalen_def      = 25;
    confpar.minstalta_def   = 0.5;
    confpar.maxstalta_def   = 2;
    confpar.winlen_def      = 30;
    confpar.overlap_def     = 0.5;
    confpar.tolerance_def   = 0;
    confpar.saturation_def  = 0;
    confpar.noisywin_def    = 0;

    confpar.stalen          = confpar.stalen_def;
    confpar.ltalen          = confpar.ltalen_def;
    confpar.minstalta       = confpar.minstalta_def;
    confpar.maxstalta       = confpar.maxstalta_def;
    confpar.winlen          = confpar.winlen_def;
    confpar.overlap         = confpar.overlap_def;
    confpar.tolerance       = confpar.tolerance_def;
```

```
    confpar.saturation      = confpar.saturation_def;
    confpar.noisywin        = confpar.noisywin_def;
}

/*****************************************************************************/
/* WS_APPLY_VALUE:                                                           */
/*****************************************************************************/

void ws_apply_value (GtkWidget *widget, gpointer data)
{
    confpar.stalen     = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton1));
    confpar.ltalen     = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton2));
    confpar.minstalta  = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton3));
    confpar.maxstalta  = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton4));
    confpar.winlen     = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton5));
    confpar.overlap    = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton6));
    confpar.tolerance  = gtk_spin_button_get_value
                         (GTK_SPIN_BUTTON(ws_spinbutton7));
    confpar.saturation = gtk_toggle_button_get_active
                         (GTK_TOGGLE_BUTTON (ws_checkbutton1));
    confpar.noisywin   = gtk_toggle_button_get_active
                         (GTK_TOGGLE_BUTTON (ws_checkbutton2));
}

/*****************************************************************************/
/* WS_UNDO_VALUE:                                                            */
/*****************************************************************************/

void ws_undo_value (GtkWidget *widget, gpointer data)
{
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton1),
                               confpar.stalen);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton2),
                               confpar.ltalen);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton3),
                               confpar.minstalta);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton4),
                               confpar.maxstalta);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton5),
                               confpar.winlen);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton6),
                               confpar.overlap);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(ws_spinbutton7),
                               confpar.tolerance);

    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (ws_checkbutton1),
                                  confpar.saturation);
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (ws_checkbutton2),
                                  confpar.noisywin);
}

/*****************************************************************************/
/* WS_DEFAULT_VALUE:                                                         */
/*****************************************************************************/

void ws_default_value (GtkWidget *widget, gpointer data)
{
```

```
    ws_par_init ();
    ws_undo_value (widget, data);
}
```

1.15 - Winselcfg.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef WINSELCFG_H
#define WINSELCFG_H

    void    ws_window_showhide      (void);

    void    create_ws_par_window    (void);

    void    ws_par_init             (void);

    void    ws_apply_value          (GtkWidget *widget,
                                     gpointer data );

    void    ws_undo_value           (GtkWidget *widget,
                                     gpointer data);

    void    ws_default_value        (GtkWidget *widget,
                                     gpointer data);

# endif
```

1.16 - Hvproccfg.c

```
# include <gtk/gtk.h>

# include "globals.h"
# include "hvproccfg.h"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

GtkWidget *hv_window;
GtkWidget *hv_frame1;
GtkWidget *hv_frame2;
GtkWidget *hv_frame3;
GtkWidget *hv_frame4;
GtkWidget *hv_frame5;
GtkWidget *hv_vbox1;
GtkWidget *hv_vbox2;
GtkWidget *hv_hbox1;
GtkWidget *hv_hbox2;
GtkWidget *hv_table1;
GtkWidget *hv_table2;
GtkWidget *hv_table3;
GtkWidget *hv_table4;
GtkWidget *hv_label0;
GtkWidget *hv_label1;
GtkWidget *hv_label2;
GtkWidget *hv_label3;
```

```
GtkWidget *hv_label4;
GtkWidget *hv_label5;
GtkWidget *hv_label6;
GtkWidget *hv_label7;
GtkWidget *hv_label8;
GtkWidget *hv_label9;
GtkWidget *hv_label10;
GtkWidget *hv_label11;
GtkWidget *hv_label12;
GtkWidget *hv_label13;
GtkWidget *hv_label14;
GtkWidget *hv_label15;
GtkWidget *hv_label16;
GtkWidget *hv_spinbutton1;
GtkWidget *hv_spinbutton2;
GtkWidget *hv_spinbutton3;
GtkWidget *hv_spinbutton4;
GtkWidget *hv_spinbutton5;
GtkWidget *hv_spinbutton6;
GtkObject *hv_spinbutton1_adj;
GtkObject *hv_spinbutton2_adj;
GtkObject *hv_spinbutton3_adj;
GtkObject *hv_spinbutton4_adj;
GtkObject *hv_spinbutton5_adj;
GtkObject *hv_spinbutton6_adj;
GtkWidget *hv_combobox1;
GtkWidget *hv_combobox2;
GtkWidget *hv_combobox3;
GtkWidget *hv_combobox4;
GtkWidget *hv_combobox5;
GtkWidget *hv_combobox6;
GtkWidget *hv_hseparator1;
GtkWidget *hv_checkbutton1;
GtkWidget *hv_hbuttonbox1;
GtkWidget *hv_button1;
GtkWidget *hv_button2;
GtkWidget *hv_button3;
GtkWidget *hv_alignment1;
GtkWidget *hv_alignment2;
GtkWidget *hv_alignment3;
GtkWidget *hv_button_hbox1;
GtkWidget *hv_button_hbox2;
GtkWidget *hv_button_hbox3;
GtkWidget *hv_image1;
GtkWidget *hv_image2;
GtkWidget *hv_image3;
GtkWidget *hv_button_label1;
GtkWidget *hv_button_label2;
GtkWidget *hv_button_label3;

/*************************************************************************/
/* HV_WINDOW_SHOWHIDE:                                                   */
/*************************************************************************/

void hv_window_showhide (void)
{
    if (GTK_WIDGET_VISIBLE(hv_window))
    {
        gtk_widget_hide (hv_window);
    }
    else
    {
        gtk_widget_show (hv_window);
```

```
    }
}

/*****************************************************************************/
/* CREATE_HV_PAR_WINDOW:                                                     */
/*****************************************************************************/

void create_hv_par_window (void)
{
    /* HV patameters configuration window */
    {
        hv_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title (GTK_WINDOW (hv_window), ("Configurations"));
        gtk_window_set_resizable (GTK_WINDOW (hv_window), FALSE);
        gtk_window_move (GTK_WINDOW (hv_window), 650, 50);
        g_signal_connect (GTK_OBJECT (hv_window), "delete_event",
                          G_CALLBACK (hv_window_showhide), NULL);

        hv_vbox1 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (hv_vbox1);
        gtk_container_add (GTK_CONTAINER (hv_window), hv_vbox1);
    }

    /* Main Frame */
    {
        hv_frame1 = gtk_frame_new (NULL);
        gtk_widget_show (hv_frame1);
        gtk_box_pack_start (GTK_BOX (hv_vbox1), hv_frame1, TRUE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (hv_frame1), 5);

        hv_label0 = gtk_label_new (("<b>Processing Parameters</b>"));
        gtk_widget_show (hv_label0);
        gtk_frame_set_label_widget (GTK_FRAME (hv_frame1), hv_label0);
        gtk_label_set_use_markup (GTK_LABEL (hv_label0), TRUE);

        hv_vbox2 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (hv_vbox2);
        gtk_container_add (GTK_CONTAINER (hv_frame1), hv_vbox2);
        gtk_container_set_border_width (GTK_CONTAINER (hv_vbox2), 5);
    }

    /* Frequency spacing frame */
    {
        hv_frame2 = gtk_frame_new (NULL);
        gtk_widget_show (hv_frame2);
        gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_frame2, TRUE, TRUE, 0);

        hv_label1 = gtk_label_new (("Frequency Spacing"));
        gtk_widget_show (hv_label1);
        gtk_frame_set_label_widget (GTK_FRAME (hv_frame2), hv_label1);
        gtk_label_set_use_markup (GTK_LABEL (hv_label1), TRUE);

        hv_table1 = gtk_table_new (4, 2, FALSE);
        gtk_widget_show (hv_table1);
        gtk_container_add (GTK_CONTAINER (hv_frame2), hv_table1);
        gtk_container_set_border_width (GTK_CONTAINER (hv_table1), 5);
        gtk_table_set_col_spacings (GTK_TABLE (hv_table1), 5);
    }

    /* Frequency spacing: type */
    {
        hv_label2 = gtk_label_new (("Type:"));
        gtk_widget_show (hv_label2);
```

```
    gtk_table_attach (GTK_TABLE (hv_table1), hv_label2, 0, 1, 0, 1,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label2), 0, 0.5);

    hv_combobox1 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox1);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_combobox1, 1, 2, 0, 1,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    g_signal_connect (GTK_OBJECT (hv_combobox1), "changed",
                      G_CALLBACK (hv_freqsp_set_ctrl), NULL);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox1),("fft"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox1),("fft reduced"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox1),("linear"));

    /* FOR FUTURE USE ONLY!                          */
    /* gtk_combo_box_append_text                     */
    /* (GTK_COMBO_BOX (hv_combobox1),("log"));       */
}

/* Frequency spacing: frequency minimum */
{
    hv_label3 = gtk_label_new (("Frequency minumum:"));
    gtk_widget_show (hv_label3);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_label3, 0, 1, 1, 2,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label3), 0, 0.5);

    hv_spinbutton1_adj = gtk_adjustment_new (0, 0, 100, 0.1, 10, 10);
    hv_spinbutton1 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (hv_spinbutton1_adj), 1, 1);
    gtk_widget_show (hv_spinbutton1);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_spinbutton1, 1, 2, 1, 2,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Frequency spacing: frequency maximum */
{
    hv_label4 = gtk_label_new (("Frequency maximum:"));
    gtk_widget_show (hv_label4);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_label4, 0, 1, 2, 3,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label4), 0, 0.5);

    hv_spinbutton2_adj = gtk_adjustment_new (0, 0, 100, 0.1, 10, 10);
    hv_spinbutton2 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (hv_spinbutton2_adj), 1, 1);
    gtk_widget_show (hv_spinbutton2);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_spinbutton2, 1, 2, 2, 3,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Frequency spacing: number of points */
{
    hv_label5 = gtk_label_new (("Number of points:"));
```

```
    gtk_widget_show (hv_label5);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_label5, 0, 1, 3, 4,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label5), 0, 0.5);

    hv_spinbutton3_adj = gtk_adjustment_new (0, 1, 50000, 1, 10, 10);
    hv_spinbutton3 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (hv_spinbutton3_adj), 1, 0);
    gtk_widget_show (hv_spinbutton3);
    gtk_table_attach (GTK_TABLE (hv_table1), hv_spinbutton3, 1, 2, 3, 4,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Offset removal frame */
{
    hv_frame3 = gtk_frame_new (NULL);
    gtk_widget_show (hv_frame3);
    gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_frame3, TRUE, TRUE, 0);

    hv_label6 = gtk_label_new (("Offset Removal"));
    gtk_widget_show (hv_label6);
    gtk_frame_set_label_widget (GTK_FRAME (hv_frame3), hv_label6);
    gtk_label_set_use_markup (GTK_LABEL (hv_label6), TRUE);

    hv_table2 = gtk_table_new (2, 2, FALSE);
    gtk_widget_show (hv_table2);
    gtk_container_add (GTK_CONTAINER (hv_frame3), hv_table2);
    gtk_container_set_border_width (GTK_CONTAINER (hv_table2), 5);
    gtk_table_set_col_spacings (GTK_TABLE (hv_table2), 5);
}

/* Offset removal: type */
{
    hv_label7 = gtk_label_new (("Type:"));
    gtk_widget_show (hv_label7);
    gtk_table_attach (GTK_TABLE (hv_table2), hv_label7, 0, 1, 0, 1,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label7), 0, 0.5);

    hv_combobox2 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox2);
    gtk_table_attach (GTK_TABLE (hv_table2), hv_combobox2, 1, 2, 0, 1,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    g_signal_connect (GTK_OBJECT (hv_combobox2), "changed",
                      G_CALLBACK (hv_offrem_set_ctrl), NULL);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox2),("none"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox2),("mean"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox2),("high-pass"));
}

/* Offset removal: frequency */
{
    hv_label8 = gtk_label_new (("Frequency:"));
    gtk_widget_show (hv_label8);
    gtk_table_attach (GTK_TABLE (hv_table2), hv_label8, 0, 1, 1, 2,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
```

```
                    (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label8), 0, 0.5);

    hv_spinbutton4_adj = gtk_adjustment_new (0, 0, 100, 0.1, 10, 10);
    hv_spinbutton4 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (hv_spinbutton4_adj), 1, 1);
    gtk_widget_show (hv_spinbutton4);
    gtk_table_attach (GTK_TABLE (hv_table2), hv_spinbutton4, 1, 2, 1, 2,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Tapering frame */
{
    hv_frame4 = gtk_frame_new (NULL);
    gtk_widget_show (hv_frame4);
    gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_frame4, TRUE, TRUE, 0);

    hv_label9 = gtk_label_new (("Tapering"));
    gtk_widget_show (hv_label9);
    gtk_frame_set_label_widget (GTK_FRAME (hv_frame4), hv_label9);
    gtk_label_set_use_markup (GTK_LABEL (hv_label9), TRUE);

    hv_table3 = gtk_table_new (2, 2, FALSE);
    gtk_widget_show (hv_table3);
    gtk_container_add (GTK_CONTAINER (hv_frame4), hv_table3);
    gtk_container_set_border_width (GTK_CONTAINER (hv_table3), 5);
    gtk_table_set_col_spacings (GTK_TABLE (hv_table3), 5);
}

/* Tapering: type */
{
    hv_label10 = gtk_label_new (("Type:"));
    gtk_widget_show (hv_label10);
    gtk_table_attach (GTK_TABLE (hv_table3), hv_label10, 0, 1, 0, 1,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label10), 0, 0.5);

    hv_combobox3 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox3);
    gtk_table_attach (GTK_TABLE (hv_table3), hv_combobox3, 1, 2, 0, 1,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    g_signal_connect (GTK_OBJECT (hv_combobox3), "changed",
                      G_CALLBACK (hv_tape_set_ctrl), NULL);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox3),("box car (none)"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox3),("Hanning"));
}

/* Tapering: percentage */
{
    hv_label11 = gtk_label_new (("Percentage:"));
    gtk_widget_show (hv_label11);
    gtk_table_attach (GTK_TABLE (hv_table3), hv_label11, 0, 1, 1, 2,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label11), 0, 0.5);

    hv_spinbutton5_adj = gtk_adjustment_new (0, 0, 100, 1, 10, 10);
    hv_spinbutton5 = gtk_spin_button_new
```

```
                    (GTK_ADJUSTMENT (hv_spinbutton5_adj), 1, 0);
    gtk_widget_show (hv_spinbutton5);
    gtk_table_attach (GTK_TABLE (hv_table3), hv_spinbutton5, 1, 2, 1, 2,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
}

/* Smoothing frame */
{
    hv_frame5 = gtk_frame_new (NULL);
    gtk_widget_show (hv_frame5);
    gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_frame5, TRUE, TRUE, 0);

    hv_label12 = gtk_label_new (("Smoothing"));
    gtk_widget_show (hv_label12);
    gtk_frame_set_label_widget (GTK_FRAME (hv_frame5), hv_label12);
    gtk_label_set_use_markup (GTK_LABEL (hv_label12), TRUE);

    hv_table4 = gtk_table_new (3, 2, FALSE);
    gtk_widget_show (hv_table4);
    gtk_container_add (GTK_CONTAINER (hv_frame5), hv_table4);
    gtk_container_set_border_width (GTK_CONTAINER (hv_table4), 5);
    gtk_table_set_col_spacings (GTK_TABLE (hv_table4), 5);
}

/* Smoothing: type */
{
    hv_label13 = gtk_label_new (("Type:"));
    gtk_widget_show (hv_label13);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_label13, 0, 1, 0, 1,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label13), 0, 0.5);

    hv_combobox4 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox4);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_combobox4, 1, 2, 0, 1,
                      (GtkAttachOptions) (GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    g_signal_connect (GTK_OBJECT (hv_combobox4), "changed",
                      G_CALLBACK (hv_smth_set_ctrl), NULL);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox4),("none"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox4),("linear"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox4),("Konno-Ohmachi"));
}

/* Smoothing: bandwith */
{
    hv_label14 = gtk_label_new (("Bandwith:"));
    gtk_widget_show (hv_label14);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_label14, 0, 1, 1, 2,
                      (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                      (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label14), 0, 0.5);

    hv_spinbutton6_adj = gtk_adjustment_new (0, 0, 100, 0.1, 10, 10);
    hv_spinbutton6 = gtk_spin_button_new
                     (GTK_ADJUSTMENT (hv_spinbutton6_adj), 1, 1);
    gtk_widget_show (hv_spinbutton6);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_spinbutton6, 1, 2, 1, 2,
```

```
                        (GtkAttachOptions) (GTK_FILL),
                        (GtkAttachOptions) (0), 0, 0);
}

/* Smoothing: weight */
{
    hv_label15 = gtk_label_new (("Weight:"));
    gtk_widget_show (hv_label15);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_label15, 0, 1, 2, 3,
                        (GtkAttachOptions) (GTK_EXPAND | GTK_FILL),
                        (GtkAttachOptions) (0), 0, 0);
    gtk_misc_set_alignment (GTK_MISC (hv_label15), 0, 0.5);

    hv_combobox5 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox5);
    gtk_table_attach (GTK_TABLE (hv_table4), hv_combobox5, 1, 2, 2, 3,
                        (GtkAttachOptions) (GTK_FILL),
                        (GtkAttachOptions) (0), 0, 0);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox5),("box car (constant)"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox5),("triangular"));
}

/* Merging */
{
    hv_hbox1 = gtk_hbox_new (FALSE, 0);
    gtk_widget_show (hv_hbox1);
    gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_hbox1, FALSE, FALSE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (hv_hbox1), 5);

    hv_label16 = gtk_label_new (("Merging:"));
    gtk_widget_show (hv_label16);
    gtk_box_pack_start (GTK_BOX (hv_hbox1), hv_label16, FALSE, FALSE, 0);

    hv_combobox6 = gtk_combo_box_new_text ();
    gtk_widget_show (hv_combobox6);
    gtk_box_pack_end (GTK_BOX (hv_hbox1), hv_combobox6, FALSE, FALSE, 0);
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox6),("arithmetic"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox6),("geometric"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox6),("quadratic"));
    gtk_combo_box_append_text
    (GTK_COMBO_BOX (hv_combobox6),("complex"));
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox6), 0);
}

/* Separator */
{
    hv_hseparator1 = gtk_hseparator_new ();
    gtk_widget_show (hv_hseparator1);
    gtk_box_pack_start (GTK_BOX (hv_vbox2), hv_hseparator1, TRUE, TRUE, 0);
}

/* Output single window information */
{
    hv_checkbutton1 = gtk_check_button_new_with_mnemonic
                        (("Output single window information"));
    gtk_widget_show (hv_checkbutton1);
    gtk_box_pack_start (GTK_BOX (hv_vbox2),hv_checkbutton1,FALSE,FALSE,0);
    gtk_container_set_border_width (GTK_CONTAINER (hv_checkbutton1), 5);
```

```
}

/* Button box */
{
    hv_hbuttonbox1 = gtk_hbutton_box_new ();
    gtk_widget_show (hv_hbuttonbox1);
    gtk_box_pack_start (GTK_BOX(hv_vbox1), hv_hbuttonbox1, FALSE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (hv_hbuttonbox1), 5);
    gtk_button_box_set_layout (GTK_BUTTON_BOX (hv_hbuttonbox1),
                               GTK_BUTTONBOX_SPREAD);
}

/* Apply button */
{
    hv_button1 = gtk_button_new ();
    g_signal_connect (G_OBJECT (hv_button1), "clicked",
                      G_CALLBACK (hv_apply_value), NULL);
    gtk_widget_show (hv_button1);
    gtk_container_add (GTK_CONTAINER (hv_hbuttonbox1), hv_button1);
    GTK_WIDGET_SET_FLAGS (hv_button1, GTK_CAN_DEFAULT);

    hv_alignment1 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (hv_alignment1);
    gtk_container_add (GTK_CONTAINER (hv_button1), hv_alignment1);

    hv_button_hbox1 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (hv_button_hbox1);
    gtk_container_add (GTK_CONTAINER (hv_alignment1), hv_button_hbox1);

    hv_image1 = gtk_image_new_from_stock
                ("gtk-apply", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (hv_image1);
    gtk_box_pack_start (GTK_BOX (hv_button_hbox1), hv_image1,
                        FALSE, FALSE, 0);

    hv_button_label1 = gtk_label_new_with_mnemonic ("_Apply");
    gtk_widget_show (hv_button_label1);
    gtk_box_pack_start (GTK_BOX (hv_button_hbox1), hv_button_label1,
                        FALSE, FALSE, 0);
}

/* Cancel button */
{
    hv_button2 = gtk_button_new ();
    g_signal_connect (G_OBJECT (hv_button2), "clicked",
                      G_CALLBACK (hv_undo_value), NULL);
    gtk_widget_show (hv_button2);
    gtk_container_add (GTK_CONTAINER (hv_hbuttonbox1), hv_button2);
    GTK_WIDGET_SET_FLAGS (hv_button2, GTK_CAN_DEFAULT);

    hv_alignment2 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (hv_alignment2);
    gtk_container_add (GTK_CONTAINER (hv_button2), hv_alignment2);

    hv_button_hbox2 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (hv_button_hbox2);
    gtk_container_add (GTK_CONTAINER (hv_alignment2), hv_button_hbox2);

    hv_image2 = gtk_image_new_from_stock
                ("gtk-cancel", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (hv_image2);
    gtk_box_pack_start (GTK_BOX (hv_button_hbox2), hv_image2,
                        FALSE, FALSE, 0);
```

```
        hv_button_label2 = gtk_label_new_with_mnemonic ("_Cancel");
        gtk_widget_show (hv_button_label2);
        gtk_box_pack_start (GTK_BOX (hv_button_hbox2), hv_button_label2,
                            FALSE, FALSE, 0);
    }

    /* Default button */
    {
        hv_button3 = gtk_button_new ();
        g_signal_connect (G_OBJECT (hv_button3), "clicked",
                          G_CALLBACK (hv_default_value), NULL);
        gtk_widget_show (hv_button3);
        gtk_container_add (GTK_CONTAINER (hv_hbuttonbox1), hv_button3);
        GTK_WIDGET_SET_FLAGS (hv_button3, GTK_CAN_DEFAULT);

        hv_alignment3 = gtk_alignment_new (0.5, 0.5, 0, 0);
        gtk_widget_show (hv_alignment3);
        gtk_container_add (GTK_CONTAINER (hv_button3), hv_alignment3);

        hv_button_hbox3 = gtk_hbox_new (FALSE, 2);
        gtk_widget_show (hv_button_hbox3);
        gtk_container_add (GTK_CONTAINER (hv_alignment3), hv_button_hbox3);

        hv_image3 = gtk_image_new_from_stock
                    ("gtk-undo", GTK_ICON_SIZE_BUTTON);
        gtk_widget_show (hv_image3);
        gtk_box_pack_start (GTK_BOX (hv_button_hbox3), hv_image3,
                            FALSE, FALSE, 0);

        hv_button_label3 = gtk_label_new_with_mnemonic ("_Default");
        gtk_widget_show (hv_button_label3);
        gtk_box_pack_start (GTK_BOX (hv_button_hbox3), hv_button_label3,
                            FALSE, FALSE, 0);
    }

    /* Setting default value */
    hv_default_value (NULL, NULL);

    gtk_widget_show (hv_window);
}

/*****************************************************************************/
/* HV_PAR_INIT:                                                              */
/*****************************************************************************/

void hv_par_init (void)
{
    confpar.freqsp_type_def     = 2;
    confpar.freqsp_min_def      = 0;
    confpar.freqsp_max_def      = 20;
    confpar. freqsp_npnt_def    = 2000;
    confpar.offrem_type_def     = 1;
    confpar.offrem_freq_def     = 20;
    confpar.tape_type_def       = 1;
    confpar.tape_perc_def       = 5;
    confpar.smth_type_def       = 1;
    confpar.smth_band_def       = 2;
    confpar.smth_wgh_def        = 1;
    confpar.merging_def         = 1;
    confpar.outsinwin_def       = 0;

    confpar.freqsp_type         = confpar.freqsp_type_def;
```

```
    confpar.freqsp_min          = confpar.freqsp_min_def;
    confpar.freqsp_max          = confpar.freqsp_max_def;
    confpar. freqsp_npnt        = confpar. freqsp_npnt_def;
    confpar.offrem_type         = confpar.offrem_type_def;
    confpar.offrem_freq         = confpar.offrem_freq_def;
    confpar.tape_type           = confpar.tape_type_def;
    confpar.tape_perc           = confpar.tape_perc_def;
    confpar.smth_type           = confpar.smth_type_def;
    confpar.smth_band           = confpar.smth_band_def;
    confpar.smth_wgh            = confpar.smth_wgh_def;
    confpar.merging             = confpar.merging_def;
    confpar.outsinwin           = confpar.outsinwin_def;
}

/*****************************************************************************/
/* HV_APPLY_VALUE:                                                           */
/*****************************************************************************/

void hv_apply_value (GtkWidget *widget, gpointer data)
{
    confpar.freqsp_type  = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox1));
    confpar.offrem_type  = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox2));
    confpar.tape_type    = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox3));
    confpar.smth_type    = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox4));
    confpar.smth_wgh     = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox5));
    confpar.merging      = gtk_combo_box_get_active
                           (GTK_COMBO_BOX (hv_combobox6));

    confpar.freqsp_min   = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton1));
    confpar.freqsp_max   = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton2));
    confpar.freqsp_npnt  = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton3));
    confpar.offrem_freq  = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton4));
    confpar.tape_perc    = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton5));
    confpar.smth_band    = gtk_spin_button_get_value
                           (GTK_SPIN_BUTTON(hv_spinbutton6));

    confpar.outsinwin    = gtk_toggle_button_get_active
                           (GTK_TOGGLE_BUTTON (hv_checkbutton1));;
}

/*****************************************************************************/
/* HV_UNDO_VALUE:                                                            */
/*****************************************************************************/

void hv_undo_value (GtkWidget *widget, gpointer data)
{
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox1),
                              confpar.freqsp_type);
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox2),
                              confpar.offrem_type);
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox3),
                              confpar.tape_type);
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox4),
```

```
                              confpar.smth_type);
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox5),
                              confpar.smth_wgh);
    gtk_combo_box_set_active (GTK_COMBO_BOX (hv_combobox6),
                              confpar.merging);

    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton1),
                               confpar.freqsp_min);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton2),
                               confpar.freqsp_max);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton3),
                               confpar.freqsp_npnt);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton4),
                               confpar.offrem_freq);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton5),
                               confpar.tape_perc);
    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv_spinbutton6),
                               confpar.smth_band);

    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (hv_checkbutton1),
                                  confpar.outsinwin);
}

/*****************************************************************************/
/* HV_DEFAULT_VALUE:                                                         */
/*****************************************************************************/

void hv_default_value (GtkWidget *widget, gpointer data)
{
    hv_par_init ();
    hv_undo_value (widget, data);
}

/*****************************************************************************/
/* HV_FREQSP_SET_CTRL:                                                       */
/*****************************************************************************/

void hv_freqsp_set_ctrl (GtkWidget *widget, gpointer data)
{
    gint select;

    select = gtk_combo_box_get_active (GTK_COMBO_BOX (hv_combobox1));

    if (select == 0)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label3), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label4), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label5), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton1), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton2), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton3), FALSE);
    }

    if (select == 1)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label3), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label4), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label5), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton1), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton2), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton3), FALSE);
    }
```

```
    if ((select == 2) || (select == 3))
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label3), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label4), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label5), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton1), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton2), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton3), TRUE);
    }
}

/*****************************************************************************/
/* HV_OFFREM_SET_CTRL:                                                       */
/*****************************************************************************/

void hv_offrem_set_ctrl (GtkWidget *widget, gpointer data)
{
    gint select;

    select = gtk_combo_box_get_active (GTK_COMBO_BOX (hv_combobox2));

    if ((select == 0) || (select == 1))
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label8), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton4), FALSE);
    }

    if (select == 2)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label8), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton4), TRUE);
    }
}

/*****************************************************************************/
/* HV_TAPE_SET_CTRL:                                                         */
/*****************************************************************************/

void hv_tape_set_ctrl (GtkWidget *widget, gpointer data)
{
    gint select;

    select = gtk_combo_box_get_active (GTK_COMBO_BOX (hv_combobox3));

    if (select == 0)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label11), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton5), FALSE);
    }

    if (select == 1)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label11), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton5), TRUE);
    }
}

/*****************************************************************************/
/* HV_SMTH_SET_CTRL:                                                         */
/*****************************************************************************/

void hv_smth_set_ctrl (GtkWidget *widget, gpointer data)
{
```

```
    gint select;

    select = gtk_combo_box_get_active (GTK_COMBO_BOX (hv_combobox4));

    if (select == 0)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label14), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label15), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton6), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_combobox5), FALSE);
    }

    if (select == 1)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label14), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label15), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton6), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_combobox5), TRUE);
    }

    if (select == 2)
    {
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label14), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_label15), FALSE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_spinbutton6), TRUE);
        gtk_widget_set_sensitive (GTK_WIDGET (hv_combobox5), FALSE);
    }
}
```

1.17 - Hvproccfg.h

```
/*************************************************************************/
/* Function prototypes.                                                  */
/*************************************************************************/

#ifndef HVPROCCFG_H
#define HVPROCCFG_H

    void    hv_window_showhide    (void);

    void    create_hv_par_window  (void);

    void    hv_par_init           (void);

    void    hv_apply_value        (GtkWidget *widget,
                                   gpointer data );

    void    hv_undo_value         (GtkWidget *widget,
                                   gpointer data);

    void    hv_default_value      (GtkWidget *widget,
                                   gpointer data);

    void    hv_freqsp_set_ctrl    (GtkWidget *widget,
                                   gpointer data);

    void    hv_offrem_set_ctrl    (GtkWidget *widget,
                                   gpointer data);

    void    hv_tape_set_ctrl      (GtkWidget *widget,
                                   gpointer data);
```

```
    void    hv_smth_set_ctrl      (GtkWidget *widget,
                                   gpointer data);

# endif
```

1.18 - Mainproc.c

```
# include <gtk/gtk.h>
# include <gtk/gtkmain.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

# include "globals.h"
# include "mainproc.h"
# include "platdep.h"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

gint proc_type = 0;
gboolean proc_state = 0;

GtkWidget *cnsl_window;
GtkWidget *cnsl_vbox1;
GtkWidget *cnsl_frame;
GtkWidget *cnsl_alignment;
GtkWidget *cnsl_vbox2;
GtkWidget *cnsl_scrolledwindow;
GtkWidget *cnsl_textview;
GtkWidget *cnsl_progressbar;
GtkWidget *cnsl_label;
GtkWidget *cnsl_hbuttonbox;
GtkWidget *cnsl_button1;
GtkWidget *cnsl_alignment1;
GtkWidget *cnsl_hbox1;
GtkWidget *cnsl_image1;
GtkWidget *cnsl_button_label1;
GtkWidget *cnsl_button2;
GtkWidget *cnsl_alignment2;
GtkWidget *cnsl_hbox2;
GtkWidget *cnsl_image2;
GtkWidget *cnsl_button_label2;
GtkWidget *cnsl_button3;
GtkWidget *cnsl_alignment3;
GtkWidget *cnsl_hbox3;
GtkWidget *cnsl_image3;
GtkWidget *cnsl_button_label3;

/*****************************************************************************/
/* CREATE_CONSOLE                                                            */
/*****************************************************************************/

void create_console_window (void)
{
    /* Console window */
    {
        cnsl_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_widget_set_size_request (GTK_WIDGET(cnsl_window),500,400);
```

```
    gtk_window_position (GTK_WINDOW(cnsl_window), GTK_WIN_POS_CENTER);
    gtk_container_set_border_width (GTK_CONTAINER (cnsl_window), 10);
    gtk_window_set_title (GTK_WINDOW (cnsl_window), "Processing");
    g_signal_connect (GTK_OBJECT (cnsl_window), "delete_event",
                      G_CALLBACK (console_window_hide), NULL);

    cnsl_vbox1 = gtk_vbox_new (FALSE, 5);
    gtk_widget_show (cnsl_vbox1);
    gtk_container_add (GTK_CONTAINER (cnsl_window), cnsl_vbox1);

    cnsl_frame = gtk_frame_new (NULL);
    gtk_widget_show (cnsl_frame);
    gtk_box_pack_start (GTK_BOX (cnsl_vbox1), cnsl_frame, TRUE, TRUE, 0);

    cnsl_alignment = gtk_alignment_new (0.5, 0.5, 1, 1);
    gtk_widget_show (cnsl_alignment);
    gtk_container_add (GTK_CONTAINER (cnsl_frame), cnsl_alignment);
    gtk_alignment_set_padding (GTK_ALIGNMENT (cnsl_alignment), 0, 0, 5, 5);

    cnsl_vbox2 = gtk_vbox_new (FALSE, 0);
    gtk_widget_show (cnsl_vbox2);
    gtk_container_add (GTK_CONTAINER (cnsl_alignment), cnsl_vbox2);

    cnsl_label = gtk_label_new ("<b>CONSOLE MESSAGE</b>");
    gtk_widget_show (cnsl_label);
    gtk_frame_set_label_widget (GTK_FRAME (cnsl_frame), cnsl_label);
    gtk_label_set_use_markup (GTK_LABEL (cnsl_label), TRUE);
}

/* Console */
{
    cnsl_scrolledwindow = gtk_scrolled_window_new (NULL, NULL);
    gtk_widget_show (cnsl_scrolledwindow);
    gtk_box_pack_start (GTK_BOX (cnsl_vbox2), cnsl_scrolledwindow,
                        TRUE, TRUE, 0);
    gtk_scrolled_window_set_policy
    (GTK_SCROLLED_WINDOW (cnsl_scrolledwindow),
     GTK_POLICY_AUTOMATIC,
     GTK_POLICY_AUTOMATIC);
    gtk_scrolled_window_set_shadow_type
    (GTK_SCROLLED_WINDOW (cnsl_scrolledwindow), GTK_SHADOW_IN);

    cnsl_textview = gtk_text_view_new ();
    gtk_widget_show (cnsl_textview);
    gtk_container_add (GTK_CONTAINER (cnsl_scrolledwindow), cnsl_textview);
    gtk_text_view_set_editable (GTK_TEXT_VIEW (cnsl_textview), FALSE);

    PangoFontDescription *font_desc;
    font_desc = pango_font_description_from_string
                ("sans bold 10");
    gtk_widget_modify_font(cnsl_textview, font_desc);
    pango_font_description_free(font_desc);
}

/* Progress Bar */
{
    cnsl_progressbar = gtk_progress_bar_new ();
    gtk_widget_show (cnsl_progressbar);
    gtk_box_pack_start (GTK_BOX (cnsl_vbox2), cnsl_progressbar,
                        FALSE, FALSE, 5);
    gtk_progress_bar_set_fraction (GTK_PROGRESS_BAR (cnsl_progressbar), 0);
    gtk_progress_bar_set_text (GTK_PROGRESS_BAR (cnsl_progressbar),
                               "Progress...");
```

```
}

/* Button box */
cnsl_hbuttonbox = gtk_hbutton_box_new ();
gtk_widget_show (cnsl_hbuttonbox);
gtk_box_pack_start (GTK_BOX (cnsl_vbox1), cnsl_hbuttonbox,
                    FALSE, TRUE, 0);
gtk_button_box_set_layout (GTK_BUTTON_BOX (cnsl_hbuttonbox),
                           GTK_BUTTONBOX_SPREAD);

/* Start button */
{
    cnsl_button1 = gtk_button_new ();
    g_signal_connect (G_OBJECT (cnsl_button1), "clicked",
                      G_CALLBACK (genhv_start), NULL);
    gtk_widget_show (cnsl_button1);
    gtk_container_add (GTK_CONTAINER (cnsl_hbuttonbox), cnsl_button1);
    GTK_WIDGET_SET_FLAGS (cnsl_button1, GTK_CAN_DEFAULT);

    cnsl_alignment1 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (cnsl_alignment1);
    gtk_container_add (GTK_CONTAINER (cnsl_button1), cnsl_alignment1);

    cnsl_hbox1 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (cnsl_hbox1);
    gtk_container_add (GTK_CONTAINER (cnsl_alignment1), cnsl_hbox1);

    cnsl_image1 = gtk_image_new_from_stock
                  ("gtk-execute", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (cnsl_image1);
    gtk_box_pack_start (GTK_BOX (cnsl_hbox1),
                        cnsl_image1, FALSE, FALSE, 0);

    cnsl_button_label1 = gtk_label_new_with_mnemonic ("_Start");
    gtk_widget_show (cnsl_button_label1);
    gtk_box_pack_start (GTK_BOX (cnsl_hbox1), cnsl_button_label1,
                        FALSE, FALSE, 0);
}

/* Stop button */
{
    cnsl_button2 = gtk_button_new ();
    g_signal_connect (G_OBJECT (cnsl_button2), "clicked",
                      G_CALLBACK (genhv_stop), NULL);
    gtk_widget_show (cnsl_button2);
    gtk_container_add (GTK_CONTAINER (cnsl_hbuttonbox), cnsl_button2);
    GTK_WIDGET_SET_FLAGS (cnsl_button2, GTK_CAN_DEFAULT);

    cnsl_alignment2 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (cnsl_alignment2);
    gtk_container_add (GTK_CONTAINER (cnsl_button2), cnsl_alignment2);

    cnsl_hbox2 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (cnsl_hbox2);
    gtk_container_add (GTK_CONTAINER (cnsl_alignment2), cnsl_hbox2);

    cnsl_image2 = gtk_image_new_from_stock
                  ("gtk-stop", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (cnsl_image2);
    gtk_box_pack_start (GTK_BOX (cnsl_hbox2),
                        cnsl_image2, FALSE, FALSE, 0);

    cnsl_button_label2 = gtk_label_new_with_mnemonic ("_Stop");
```

```
        gtk_widget_show (cnsl_button_label2);
        gtk_box_pack_start (GTK_BOX (cnsl_hbox2), cnsl_button_label2,
                            FALSE, FALSE, 0);
    }

    /* Cancel button */
    {
        cnsl_button3 = gtk_button_new ();
        g_signal_connect (G_OBJECT (cnsl_button3), "clicked",
                          G_CALLBACK (console_window_hide), NULL);
        gtk_widget_show (cnsl_button3);
        gtk_container_add (GTK_CONTAINER (cnsl_hbuttonbox), cnsl_button3);
        GTK_WIDGET_SET_FLAGS (cnsl_button3, GTK_CAN_DEFAULT);

        cnsl_alignment3 = gtk_alignment_new (0.5, 0.5, 0, 0);
        gtk_widget_show (cnsl_alignment3);
        gtk_container_add (GTK_CONTAINER (cnsl_button3), cnsl_alignment3);

        cnsl_hbox3 = gtk_hbox_new (FALSE, 2);
        gtk_widget_show (cnsl_hbox3);
        gtk_container_add (GTK_CONTAINER (cnsl_alignment3), cnsl_hbox3);

        cnsl_image3 = gtk_image_new_from_stock
                      ("gtk-close", GTK_ICON_SIZE_BUTTON);
        gtk_widget_show (cnsl_image3);
        gtk_box_pack_start (GTK_BOX (cnsl_hbox3),
                            cnsl_image3, FALSE, FALSE, 0);

        cnsl_button_label3 = gtk_label_new_with_mnemonic ("_Close");
        gtk_widget_show (cnsl_button_label3);
        gtk_box_pack_start (GTK_BOX (cnsl_hbox3), cnsl_button_label3,
                            FALSE, FALSE, 0);
    }

    gtk_widget_show_all (cnsl_window);
    gtk_widget_hide (cnsl_window);
}

/*****************************************************************************/
/* CONSOLE_WINDOW_SHOW                                                       */
/*****************************************************************************/

void console_window_show (GtkWidget *widget, gchar *type)
{
    gchar cnsl_str[100];

    if (strstr(type,"single") == type)
    {
        proc_type = 0;
    }

    if (strstr(type,"all") == type)
    {
        proc_type = 1;
    }

    if (strstr(type,"average") == type)
    {
        proc_type = 2;
    }

    if (!GTK_WIDGET_VISIBLE(cnsl_window))
    {
```

```
        gtk_widget_show (cnsl_window);
    }

    sprintf
    (cnsl_str, "  GSESAME$  Current processing type:  %s\n", type);

    gtk_text_buffer_set_text
    (gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), cnsl_str, -1);

    gtk_progress_bar_set_fraction (GTK_PROGRESS_BAR (cnsl_progressbar), 0);

}

/*****************************************************************************/
/* CNSL_WINDOW_HIDE                                                          */
/*****************************************************************************/

void console_window_hide (void)
{
    if (GTK_WIDGET_VISIBLE(cnsl_window))
    {
        gtk_widget_hide (cnsl_window);
    }
}

/*****************************************************************************/
/* GENHV_START                                                               */
/*****************************************************************************/

void genhv_start (void)
{
    GtkTreeSelection *selection;
    GtkTreeModel *model;
    GtkTreeIter iter;
    SAF_LIST_PTR ptr_list;
    gdouble count_tot = 0;
    gdouble count = 0;
    gchar *ptr_filename;
    FILE *fp;

    proc_state = 0;

    /*************************************************************************/
    /* SINGLE ITEM                                                           */
    /*************************************************************************/

    if (proc_type == 0)
    {
        model = gtk_tree_view_get_model (GTK_TREE_VIEW(view));
        selection = gtk_tree_view_get_selection (GTK_TREE_VIEW(view));

        if (gtk_tree_selection_get_selected (selection, NULL, &iter))
        {
            gtk_tree_model_get (model, &iter, 9, &ptr_list, -1);

            write_ws_par ();
            write_hv_par ();
            write_file_list (ptr_list->file);
            genhv_core (ptr_list->file, 1, 1);
        }
    }

    /*************************************************************************/
```

```
    /* ALL ITEMS                                                                 */
    /*****************************************************************************/

    if (proc_type == 1)
    {
        if (first_saf != NULL)
        {
            write_ws_par ();
            write_hv_par ();

            ptr_list = first_saf;

            while (ptr_list != NULL)
            {
                count_tot++;
                ptr_list = ptr_list->next_saf;
            }

            ptr_list = first_saf;

            while (ptr_list != NULL)
            {
                if (proc_state != 1)
                {
                    count++;
                    write_file_list (ptr_list->file);
                    genhv_core (ptr_list->file, count, count_tot);
                }
                ptr_list = ptr_list->next_saf;
            }
        }
    }

    /*****************************************************************************/
    /* LIST AVERAGE                                                              */
    /*****************************************************************************/

    if (proc_type == 2)
    {
        if (first_saf != NULL)
        {
            write_ws_par ();
            write_hv_par ();

            ptr_list = first_saf;

            if ((fp = fopen(listpar_path, "w")) != NULL)
            {
                while (ptr_list != NULL)
                {
                    fprintf(fp,"%s\n",ptr_list->file);
                    ptr_filename = ptr_list->file;
                    ptr_list = ptr_list->next_saf;
                }
                fclose (fp);

                genhv_core (ptr_filename, 1, 1);
            }
        }
    }
}

/*****************************************************************************/
```

```
/* GENHV_STOP                                                                 */
/*****************************************************************************/

void genhv_stop (void)
{
    proc_state = 1;
}

/*****************************************************************************/
/* GENHV_CORE                                                                */
/*****************************************************************************/

void genhv_core (gchar *path_file, gdouble count, gdouble count_tot)
{
    gchar *file;
    GtkTextIter textiter;
    gchar winsel_str[500];
    gchar hvproc_str[500];
    gdouble frac1;
    gdouble frac2;
    GtkAdjustment* adj;
    double range;

    /*************************************************************************/
    /* Creating the command line string                                      */
    /*************************************************************************/

    file = saf_filename_extract (path_file);

    sprintf (winsel_str, "%s  %s  %s  %s",
                         winsel_path,
                         listpar_path,
                         wspar_path,
                         wsout_path);

    sprintf (hvproc_str, "%s  %s  %s  %s%s.hv",
                         hvproc_path,
                         wsout_path,
                         hvpar_path,
                         out_path, file);

    /*************************************************************************/
    /* Resetting the progress bar (if need...)                               */
    /* NOTE: the "gtk_main_iteration" function is used to force              */
    /*       the progress bar step increment during computation              */
    /*************************************************************************/

    if (count == 1)
    {
        gtk_progress_bar_set_fraction
        (GTK_PROGRESS_BAR (cnsl_progressbar), 0);

        while(gtk_events_pending()) {gtk_main_iteration();}
    }

    /*************************************************************************/
    /* Calculating the progess bar step                                      */
    /*************************************************************************/

    frac1 = (count / count_tot) - (1 / (2 * count_tot));
    frac2 = (count / count_tot);

    /*************************************************************************/
```

```
/* Windows selection computation                                              */
/****************************************************************************/

gtk_text_buffer_get_end_iter
(gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter);

gtk_text_buffer_insert
(gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter,
"  GSESAME$  Processing module:  WINSELECTION\n", -1);

while(gtk_events_pending()) {gtk_main_iteration();}

if (!create_process_pipe (cnsl_textview, winsel_str))
{
    gtk_progress_bar_set_fraction
    (GTK_PROGRESS_BAR (cnsl_progressbar), frac1);

    adj = gtk_scrolled_window_get_vadjustment
          (GTK_SCROLLED_WINDOW(cnsl_scrolledwindow));
    range = (adj->upper - adj->page_size);
    gtk_adjustment_set_value (adj, range);
    gtk_adjustment_value_changed (adj);

    while(gtk_events_pending()) {gtk_main_iteration();}
}
else
{
    gtk_text_buffer_get_end_iter
    (gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter);

    gtk_text_buffer_insert
    (gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter,
    "  GSESAME$  Warning:  Unable to create process pipe!\n", -1);
}

/****************************************************************************/
/* HV spectral ratio computation                                            */
/****************************************************************************/

gtk_text_buffer_get_end_iter
(gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter);

gtk_text_buffer_insert
(gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter,
"  GSESAME$  Processing module:  HVPROC0_1\n", -1);

while(gtk_events_pending()) {gtk_main_iteration();}

if (!create_process_pipe (cnsl_textview, hvproc_str))
{
    gtk_progress_bar_set_fraction
    (GTK_PROGRESS_BAR (cnsl_progressbar), frac2);

    adj = gtk_scrolled_window_get_vadjustment
          (GTK_SCROLLED_WINDOW(cnsl_scrolledwindow));
    range = (adj->upper - adj->page_size);
    gtk_adjustment_set_value (adj, range);
    gtk_adjustment_value_changed (adj);

    while(gtk_events_pending()) {gtk_main_iteration();}
}
else
{
```

```
        gtk_text_buffer_get_end_iter
        (gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter);

        gtk_text_buffer_insert
        (gtk_text_view_get_buffer (GTK_TEXT_VIEW (cnsl_textview)), &textiter,
        "  GSESAME$  Warning:  Unable to create process pipe!\n", -1);
    }
}

/*****************************************************************************/
/* WRITE_FILE_LIST                                                           */
/*****************************************************************************/

void write_file_list (gchar *file_str)
{
    FILE *fp;

    if ((fp = fopen(listpar_path, "w")) != NULL)
    {
        fprintf(fp,"%s\n",file_str);

        fclose (fp);
    }
}

/*****************************************************************************/
/* WRITE_WS_PAR                                                              */
/*****************************************************************************/

void write_ws_par (void)
{
    FILE *fp;

    if ((fp = fopen(wspar_path, "w")) != NULL)
    {
        fprintf(fp,"%.2f  ",confpar.stalen);
        fprintf(fp,"%.2f  ",confpar.ltalen);
        fprintf(fp,"%.2f  ",confpar.minstalta);
        fprintf(fp,"%.2f  ",confpar.maxstalta);
        fprintf(fp,"%.2f  ",confpar.winlen);
        fprintf(fp,"%.2f  ",confpar.overlap);
        fprintf(fp,"%d  ",confpar.saturation);
        fprintf(fp,"%d  ",confpar.noisywin);
        fprintf(fp,"%.2f  ",confpar.tolerance);
        fprintf(fp,"\n");

        fclose (fp);
    }
}

/*****************************************************************************/
/* WRITE_HV_PAR                                                              */
/*****************************************************************************/

void write_hv_par (void)
{
    FILE *fp;
    gchar freqsp_type_str[50];
    gchar offrem_type_str[50];
    gchar tape_type_str[50];
    gchar smth_type_str[50];
    gchar smth_wgh_str[50];
    gchar merging_str[50];
```

```
gchar outsinwin_str[50];

if ((fp = fopen(hvpar_path, "w")) != NULL)
{
    fprintf(fp,"### section processing\n");

    /* Frequency spacing */
    {
        if (confpar.freqsp_type == 0)
        {
            strcpy(freqsp_type_str,"fft");
            fprintf(fp,"freq_spacing:%s\n",
                    freqsp_type_str);
        }
        if (confpar.freqsp_type == 1)
        {
            strcpy(freqsp_type_str,"fft_red");
            fprintf(fp,"freq_spacing:%s:%.2f:%.2f\n",
                    freqsp_type_str,
                    confpar.freqsp_min,
                    confpar.freqsp_max);
        }
        if (confpar.freqsp_type == 2)
        {
            strcpy(freqsp_type_str,"linear");
            fprintf(fp,"freq_spacing:%s:%.2f:%.2f:%.0f\n",
                    freqsp_type_str,
                    confpar.freqsp_min,
                    confpar.freqsp_max,
                    confpar.freqsp_npnt);
        }
        if (confpar.freqsp_type == 3)
        {
            strcpy(freqsp_type_str,"log");
            fprintf(fp,"freq_spacing:%s:%.2f:%.2f:%.0f\n",
                    freqsp_type_str,
                    confpar.freqsp_min,
                    confpar.freqsp_max,
                    confpar.freqsp_npnt);
        }
    }

    /* Offset removal */
    {
        if (confpar.offrem_type == 0)
        {
            strcpy(offrem_type_str,"no");
            fprintf(fp,"offset_rem:%s\n",
                    offrem_type_str);
        }
        if (confpar.offrem_type == 1)
        {
            strcpy(offrem_type_str,"r_mean");
            fprintf(fp,"offset_rem:%s:all\n",
                    offrem_type_str);
        }
        if (confpar.offrem_type == 2)
        {
            strcpy(offrem_type_str,"high-pass");
            fprintf(fp,"offset_rem:%s:%.2f\n",
                    offrem_type_str,
                    confpar.offrem_freq);
        }
```

```
}

/* Tapering */
{
    if (confpar.tape_type == 0)
    {
        strcpy(tape_type_str,"boxcar");
        fprintf(fp,"taper:%s\n",
                tape_type_str);
    }
    if (confpar.tape_type == 1)
    {
        strcpy(tape_type_str,"cos");
        fprintf(fp,"taper:%s:%.0f\n",
                tape_type_str,
                confpar.tape_perc);
    }
}

/* Smoothing */
{
    if (confpar.smth_type == 0)
    {
        strcpy(smth_type_str,"none");
        fprintf(fp,"smooth:%s\n",
                smth_type_str);
    }
    if (confpar.smth_type == 1)
    {
        strcpy(smth_type_str,"linear");

        if (confpar.smth_wgh == 0)
            strcpy(smth_wgh_str,"box");

        if (confpar.smth_wgh == 1)
            strcpy(smth_wgh_str,"tri");

        fprintf(fp,"smooth:%s:%.2f:%s\n",
                smth_type_str,
                confpar.smth_band,
                smth_wgh_str);
    }
    if (confpar.smth_type == 2)
    {
        strcpy(smth_type_str,"konno-ohmachi");

        fprintf(fp,"smooth:%s:%.2f\n",
                smth_type_str,
                confpar.smth_band);
    }
}
/* Merging */
{
    if (confpar.merging == 0)
        strcpy(merging_str,"arithmetic");

    if (confpar.merging == 1)
        strcpy(merging_str,"geometric");

    if (confpar.merging == 2)
        strcpy(merging_str,"quadratic");

    if (confpar.merging == 3)
```

```
                strcpy(merging_str,"complex");

            fprintf(fp,"merge_type:%s\n",
                    merging_str);
        }

        /* Single window out */
        {
            if (confpar.outsinwin == 0)
                strcpy(outsinwin_str,"no");

            if (confpar.outsinwin == 1)
                strcpy(outsinwin_str,"yes");

            fprintf(fp,"single_win_out:%s\n",
                    outsinwin_str);
        }

        /* Average spectra out */
        {
            fprintf(fp,"average_spectra_out:yes\n");
        }

        fprintf(fp,"### end processing\n");

        fclose (fp);
    }
}
```

1.19 - Mainproc.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef MAINPROC_H
#define MAINPROC_H

void create_console_window  (void);

void console_window_show    (GtkWidget *widget,
                             gchar *type);

void console_window_hide    (void);

void genhv_start            (void);

void genhv_stop             (void);

void genhv_core             (gchar *path_file,
                             gdouble count,
                             gdouble count_tot);

void write_file_list        (gchar *file_str);

void write_ws_par           (void);

void write_hv_par           (void);

# endif
```

1.20 - Platdep.c (Linux version)

```
# include <gtk/gtk.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

# include "../platdep.h"
# include "../project.h"

/******************************************************************************/
/* SET_PATH:                                                                  */
/******************************************************************************/

void set_path (void)
{
    /**************************************************************************/
    /* Setting gsesame root directory                                         */
    /**************************************************************************/

    if (getenv("GSESAME"))
    {
        sprintf (root, "%s", getenv ("GSESAME"));
    }
    else
    {
        sprintf (root, "../");
    }

    /**************************************************************************/
    /* Setting internal files                                                 */
    /**************************************************************************/

    sprintf (winsel_path, "%sbin/winselection", root);
    sprintf (hvproc_path, "%sbin/hvproc0_1", root);

    sprintf (listpar_path, "%stmp/list.par", root);
    sprintf (wspar_path, "%stmp/ws.par", root);
    sprintf (hvpar_path, "%stmp/hv.par", root);
    sprintf (wsout_path, "%stmp/ws.out", root);

    sprintf (prj_path, "%sprj/", root);
    sprintf (saf_path, "%ssaf/", root);
    sprintf (out_path, "%sout/", root);
    sprintf (img_path, "%simg/", root);
}

/******************************************************************************/
/* SAF_FILENAME_EXTRACT:                                                      */
/* This function extracts information from header's string variable.          */
/* Works both for dos and unix path.                                          */
/******************************************************************************/

void *saf_filename_extract (gchar *string)
{
    gchar *extract;

    extract = strrchr(string, '/');

    if (extract == NULL)
    {
        return string;
```

```
    }
    else
    {
        extract = extract + sizeof(char);
        return extract;
    }
}

/*****************************************************************************/
/* CREATE_PROCESS_PIPE:                                                      */
/*****************************************************************************/

gboolean create_process_pipe (GtkWidget *textview, gchar *process_str)
{
    FILE *ptr;
    gchar buf[BUFSIZ];
    gchar cnsl_buffer[BUFSIZ];
    GtkTextIter textiter;

    if ((ptr = popen (process_str, "r")) != NULL)
    {
        while (fgets(buf, BUFSIZ, ptr) != NULL)
        {
            sprintf(cnsl_buffer, "  GSESAME$  %s", buf);

            gtk_text_buffer_get_end_iter
            (gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)),
            &textiter);

            gtk_text_buffer_insert
            (gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)),
            &textiter,cnsl_buffer, -1);

            while(gtk_events_pending()) {gtk_main_iteration();}
        }

        pclose (ptr);
        return 0;
    }
    else
    {
        return 1;
    }
}
```

1.21 - Platdep.c (Windows version)

```
# include <gtk/gtk.h>
# include <windows.h>
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
# include <ctype.h>

# include "../platdep.h"
# include "../project.h"

/*****************************************************************************/
/* SET_PATH:                                                                 */
/*****************************************************************************/
```

```
void set_path (void)
{
    /*************************************************************************/
    /* Setting gsesame root directory                                        */
    /*************************************************************************/

    if (getenv("GSESAME"))
    {
        sprintf (root, "%s", getenv ("GSESAME"));
    }
    else
    {
        sprintf (root, "..\\");
    }

    /*************************************************************************/
    /* Setting internal files                                                */
    /*************************************************************************/

    sprintf (winsel_path, "%sbin\\winselection.exe", root);
    sprintf (hvproc_path, "%sbin\\hvproc0_1.exe", root);

    sprintf (listpar_path, "%stmp\\list.par", root);
    sprintf (wspar_path, "%stmp\\ws.par", root);
    sprintf (hvpar_path, "%stmp\\hv.par", root);
    sprintf (wsout_path, "%stmp\\ws.out", root);

    sprintf (prj_path, "%sprj\\", root);
    sprintf (saf_path, "%ssaf\\", root);
    sprintf (out_path, "%sout\\", root);
    sprintf (img_path, "%simg\\", root);
}

/*************************************************************************/
/* SAF_FILENAME_EXTRACT:                                                 */
/* This function extracts information from header's string variable.     */
/* Works both for dos and unix path.                                     */
/*************************************************************************/

void *saf_filename_extract (gchar *string)
{
    gchar *extract;

    extract = strrchr(string, '\\');

    if (extract == NULL)
    {
        return string;
    }
    else
    {
        extract = extract + sizeof(char);
        return extract;
    }
}

/*************************************************************************/
/* CREATE_PROCESS_PIPE:                                                  */
/*************************************************************************/

gboolean create_process_pipe (GtkWidget *textview, gchar *process_str)
{
    /*************************************************************************/
```

```
/* WARNING! WARNING! WARNING!                                                */
/* I'm not familiar with WIN32 API! So if you know a better method to     */
/* do a process pipe (but not popen, that create a popup console!) in     */
/* a portable way, please implement it. Thanks!                           */
/**************************************************************************/

GtkTextIter textiter;
gint buf_len = 4096;
gchar buf[buf_len];
gchar cnsl_buffer[buf_len];
gboolean bFuncRetn = 0;
DWORD bytes_read;
gint i;

SECURITY_ATTRIBUTES pipe_attribute;
HANDLE read_pipe, write_pipe;
PROCESS_INFORMATION information;
STARTUPINFO startup;


pipe_attribute.nLength = sizeof(SECURITY_ATTRIBUTES);
pipe_attribute.bInheritHandle = TRUE;
pipe_attribute.lpSecurityDescriptor = NULL;

if (CreatePipe(&read_pipe, &write_pipe, &pipe_attribute, 0))
{
    SetHandleInformation(read_pipe, HANDLE_FLAG_INHERIT, 0);

    ZeroMemory( &information, sizeof(PROCESS_INFORMATION) );
    ZeroMemory( &startup, sizeof(STARTUPINFO) );
    {
        startup.cb = sizeof(STARTUPINFO);
        startup.hStdError = write_pipe;
        startup.hStdOutput = write_pipe;
        startup.dwFlags |= STARTF_USESTDHANDLES;
    }

    bFuncRetn = CreateProcess(NULL, process_str, NULL,
                              NULL, TRUE, 0, NULL,
                              NULL, &startup, &information);

    if (!bFuncRetn)
    {
        CloseHandle(information.hProcess);
        CloseHandle(information.hThread);

        return 1;
    }

    if (CloseHandle(write_pipe))
    {
        while(ReadFile(read_pipe,buf,buf_len,&bytes_read,NULL))
        {
            for (i = 0; i < (buf_len - 1); i++)
            {
                if (!isascii (buf[i]))
                {
                    buf [i] = '\0';
                }
            }

            sprintf(cnsl_buffer, "  GSESAME$  %s", buf);
```

```
                gtk_text_buffer_get_end_iter
                (gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)),
                &textiter);

                gtk_text_buffer_insert
                (gtk_text_view_get_buffer (GTK_TEXT_VIEW (textview)),
                &textiter,cnsl_buffer, -1);

                while(gtk_events_pending()) {gtk_main_iteration();}
            }
        }

        return 0;
    }
    else
    {
        return 1;
    }
}
```

1.22 - Platdep.h

```
#ifndef PATH_H
#define PATH_H

/*****************************************************************************/
/* Global variables.                                                         */
/*****************************************************************************/

gchar root[100];
gchar cfg_path[150];
gchar listpar_path[150];
gchar winsel_path[150];
gchar hvproc_path[150];
gchar wspar_path[150];
gchar hvpar_path[150];
gchar wsout_path[150];
gchar prj_path[150];
gchar saf_path[150];
gchar out_path[150];
gchar img_path[150];

/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

void set_path                (void);

void *saf_filename_extract   (gchar *string);

gboolean create_process_pipe (GtkWidget *textview,
                              gchar *process_str);

# endif
```

1.23 - Gsescfg.c

```
# include <gtk/gtk.h>
```

```
# include "globals.h"
# include "gsescfg.h"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

GtkWidget *gs_window;
GtkWidget *gs_main_frame;
GtkWidget *gs_vbox1;
GtkWidget *gs_vbox2;
GtkWidget *gs_vbox3;
GtkWidget *gs_vbox4;
GtkWidget *gs_hbox1;
GtkWidget *gs_hbox2;
GtkWidget *gs_hbox3;
GtkWidget *gs_frame1;
GtkWidget *gs_frame2;
GtkWidget *gs_frame3;
GtkWidget *gs_table1;
GtkWidget *gs_alignment1;
GtkWidget *gs_alignment2;
GtkWidget *gs_alignment3;
GtkWidget *gs_alignment4;
GtkWidget *gs_alignment5;
GtkWidget *gs_alignment6;
GtkWidget *gs_label1;
GtkWidget *gs_label2;
GtkWidget *gs_label3;
GtkWidget *gs_label4;
GtkWidget *gs_label5;
GtkWidget *gs_label6;
GtkWidget *gs_label7;
GtkWidget *gs_label8;
GtkWidget *gs_label9;
GtkWidget *gs_label10;
GtkWidget *gs_label11;
GtkWidget *gs_entry1;
GtkWidget *gs_entry2;
GtkWidget *gs_entry3;
GtkWidget *gs_entry4;
GtkWidget *gs_checkbutton1;
GtkWidget *gs_checkbutton2;
GtkWidget *gs_checkbutton3;
GtkWidget *gs_radiobutton1;
GtkWidget *gs_radiobutton2;
GSList    *gs_radiobutton1_group = NULL;
GtkWidget *gs_hbuttonbox1;
GtkWidget *gs_button1;
GtkWidget *gs_button2;
GtkWidget *gs_button3;
GtkWidget *gs_image1;
GtkWidget *gs_image2;
GtkWidget *gs_image3;

/*****************************************************************************/
/* GS_WINDOW_SHOWHIDE:                                                       */
/*****************************************************************************/

void gs_window_showhide (void)
{
    if (GTK_WIDGET_VISIBLE(gs_window))
    {
```

```
         gtk_widget_hide (gs_window);
    }
    else
    {
        gtk_widget_show (gs_window);
    }
}

/*************************************************************************/
/* CREATE_GS_PAR_WINDOW:                                                 */
/*************************************************************************/

void create_gs_par_window (void)
{
    /* GSesame Configuration Window */
    {
        gs_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_set_title (GTK_WINDOW (gs_window), "Configurations");
        gtk_window_set_resizable (GTK_WINDOW (gs_window), FALSE);
        gtk_window_position (GTK_WINDOW(gs_window), GTK_WIN_POS_CENTER);
        g_signal_connect (GTK_OBJECT (gs_window), "delete_event",
                          G_CALLBACK (gs_window_showhide), NULL);

        gs_vbox1 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (gs_vbox1);
        gtk_container_add (GTK_CONTAINER (gs_window), gs_vbox1);

        gs_main_frame = gtk_frame_new (NULL);
        gtk_widget_show (gs_main_frame);
        gtk_box_pack_start (GTK_BOX (gs_vbox1), gs_main_frame, TRUE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (gs_main_frame), 5);

        gs_label1 = gtk_label_new ("<b>GSesame General Setting</b>");
        gtk_widget_show (gs_label1);
        gtk_frame_set_label_widget (GTK_FRAME (gs_main_frame), gs_label1);
        gtk_label_set_use_markup (GTK_LABEL (gs_label1), TRUE);
        gtk_misc_set_alignment (GTK_MISC (gs_label1), 0.43, 0.52);

        gs_vbox2 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (gs_vbox2);
        gtk_container_add (GTK_CONTAINER (gs_main_frame), gs_vbox2);
        gtk_container_set_border_width (GTK_CONTAINER (gs_vbox2), 5);
    }

    /* H/V Plot Configuration frame */
    {
        gs_frame2 = gtk_frame_new (NULL);
        gtk_widget_show (gs_frame2);
        gtk_box_pack_start (GTK_BOX (gs_vbox2), gs_frame2, TRUE, TRUE, 0);

        gs_label7 = gtk_label_new ("H/V Plot Configuration");
        gtk_widget_show (gs_label7);
        gtk_frame_set_label_widget (GTK_FRAME (gs_frame2), gs_label7);
        gtk_label_set_use_markup (GTK_LABEL (gs_label7), TRUE);

        gs_alignment2 = gtk_alignment_new (0.5, 0.5, 1, 1);
        gtk_widget_show (gs_alignment2);
        gtk_container_add (GTK_CONTAINER (gs_frame2), gs_alignment2);

        gs_vbox3 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (gs_vbox3);
        gtk_container_add (GTK_CONTAINER (gs_alignment2), gs_vbox3);
        gtk_container_set_border_width (GTK_CONTAINER (gs_vbox3), 5);
```

```
}

/* Image Saving Configuration frame */
{
    gs_frame3 = gtk_frame_new (NULL);
    gtk_widget_show (gs_frame3);
    gtk_box_pack_start (GTK_BOX (gs_vbox2), gs_frame3, TRUE, TRUE, 0);

    gs_label8 = gtk_label_new ("Save Image As");
    gtk_widget_show (gs_label8);
    gtk_frame_set_label_widget (GTK_FRAME (gs_frame3), gs_label8);
    gtk_label_set_use_markup (GTK_LABEL (gs_label8), TRUE);

    gs_alignment3 = gtk_alignment_new (0.5, 0.5, 1, 1);
    gtk_widget_show (gs_alignment3);
    gtk_container_add (GTK_CONTAINER (gs_frame3), gs_alignment3);

    gs_vbox4 = gtk_vbox_new (FALSE, 0);
    gtk_widget_show (gs_vbox4);
    gtk_container_add (GTK_CONTAINER (gs_alignment3), gs_vbox4);
    gtk_container_set_border_width (GTK_CONTAINER (gs_vbox4), 5);
}

/* H/V Plot checkbuttons */
{
    gs_checkbutton1 = gtk_check_button_new_with_mnemonic
    ("Foundamental Frequency (f0)");
    gtk_widget_show (gs_checkbutton1);
    gtk_box_pack_start (GTK_BOX (gs_vbox3), gs_checkbutton1,
                        FALSE, FALSE, 0);

    gs_checkbutton2 = gtk_check_button_new_with_mnemonic
    ("H/V Standard Deviation");
    gtk_widget_show (gs_checkbutton2);
    gtk_box_pack_start (GTK_BOX (gs_vbox3), gs_checkbutton2,
                        FALSE, FALSE, 0);

    gs_checkbutton3 = gtk_check_button_new_with_mnemonic
    ("f0 Standard Deviation");
    gtk_widget_show (gs_checkbutton3);
    gtk_box_pack_start (GTK_BOX (gs_vbox3), gs_checkbutton3,
                        FALSE, FALSE, 0);
}

/* Image type radiobuttons */
{
    gs_radiobutton1 = gtk_radio_button_new_with_mnemonic (NULL, "Jpeg");
    gtk_widget_show (gs_radiobutton1);
    gtk_box_pack_start (GTK_BOX (gs_vbox4), gs_radiobutton1,
                        FALSE, FALSE, 0);
    gtk_radio_button_set_group (GTK_RADIO_BUTTON (gs_radiobutton1),
                                gs_radiobutton1_group);
    gs_radiobutton1_group = gtk_radio_button_get_group
                            (GTK_RADIO_BUTTON (gs_radiobutton1));

    gs_radiobutton2 = gtk_radio_button_new_with_mnemonic (NULL, "Png");
    gtk_widget_show (gs_radiobutton2);
    gtk_box_pack_start (GTK_BOX (gs_vbox4), gs_radiobutton2,
                        FALSE, FALSE, 0);
    gtk_radio_button_set_group (GTK_RADIO_BUTTON (gs_radiobutton2),
                                gs_radiobutton1_group);
    gs_radiobutton1_group = gtk_radio_button_get_group
                            (GTK_RADIO_BUTTON (gs_radiobutton2));
```

```
}

/* Button box */
{
    gs_hbuttonbox1 = gtk_hbutton_box_new ();
    gtk_widget_show (gs_hbuttonbox1);
    gtk_box_pack_start (GTK_BOX (gs_vbox1), gs_hbuttonbox1, FALSE, TRUE, 0);
    gtk_container_set_border_width (GTK_CONTAINER (gs_hbuttonbox1), 5);
    gtk_button_box_set_layout (GTK_BUTTON_BOX (gs_hbuttonbox1),
                               GTK_BUTTONBOX_SPREAD);
}

/* Apply button */
{
    gs_button1 = gtk_button_new ();
    g_signal_connect (G_OBJECT (gs_button1), "clicked",
                      G_CALLBACK (gs_apply_value), NULL);
    gtk_widget_show (gs_button1);
    gtk_container_add (GTK_CONTAINER (gs_hbuttonbox1), gs_button1);
    GTK_WIDGET_SET_FLAGS (gs_button1, GTK_CAN_DEFAULT);

    gs_alignment4 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (gs_alignment4);
    gtk_container_add (GTK_CONTAINER (gs_button1), gs_alignment4);

    gs_hbox1 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (gs_hbox1);
    gtk_container_add (GTK_CONTAINER (gs_alignment4), gs_hbox1);

    gs_image1 = gtk_image_new_from_stock
                ("gtk-apply", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (gs_image1);
    gtk_box_pack_start (GTK_BOX (gs_hbox1), gs_image1, FALSE, FALSE, 0);

    gs_label9 = gtk_label_new_with_mnemonic ("_Apply");
    gtk_widget_show (gs_label9);
    gtk_box_pack_start (GTK_BOX (gs_hbox1), gs_label9,
                        FALSE, FALSE, 0);
}

/* Cancel button */
{
    gs_button2 = gtk_button_new ();
    g_signal_connect (G_OBJECT (gs_button2), "clicked",
                      G_CALLBACK (gs_undo_value), NULL);
    gtk_widget_show (gs_button2);
    gtk_container_add (GTK_CONTAINER (gs_hbuttonbox1), gs_button2);
    GTK_WIDGET_SET_FLAGS (gs_button2, GTK_CAN_DEFAULT);

    gs_alignment5 = gtk_alignment_new (0.5, 0.5, 0, 0);
    gtk_widget_show (gs_alignment5);
    gtk_container_add (GTK_CONTAINER (gs_button2), gs_alignment5);

    gs_hbox2 = gtk_hbox_new (FALSE, 2);
    gtk_widget_show (gs_hbox2);
    gtk_container_add (GTK_CONTAINER (gs_alignment5), gs_hbox2);

    gs_image2 = gtk_image_new_from_stock
                ("gtk-cancel", GTK_ICON_SIZE_BUTTON);
    gtk_widget_show (gs_image2);
    gtk_box_pack_start (GTK_BOX (gs_hbox2), gs_image2, FALSE, FALSE, 0);

    gs_label10 = gtk_label_new_with_mnemonic ("_Cancel");
```

```
        gtk_widget_show (gs_label10);
        gtk_box_pack_start (GTK_BOX (gs_hbox2), gs_label10,
                            FALSE, FALSE, 0);
    }

    /* Default button */
    {
        gs_button3 = gtk_button_new ();
        g_signal_connect (G_OBJECT (gs_button3), "clicked",
                          G_CALLBACK (gs_default_value), NULL);
        gtk_widget_show (gs_button3);
        gtk_container_add (GTK_CONTAINER (gs_hbuttonbox1), gs_button3);
        GTK_WIDGET_SET_FLAGS (gs_button3, GTK_CAN_DEFAULT);

        gs_alignment6 = gtk_alignment_new (0.5, 0.5, 0, 0);
        gtk_widget_show (gs_alignment6);
        gtk_container_add (GTK_CONTAINER (gs_button3), gs_alignment6);

        gs_hbox3 = gtk_hbox_new (FALSE, 2);
        gtk_widget_show (gs_hbox3);
        gtk_container_add (GTK_CONTAINER (gs_alignment6), gs_hbox3);

        gs_image3 = gtk_image_new_from_stock
                    ("gtk-undo", GTK_ICON_SIZE_BUTTON);
        gtk_widget_show (gs_image3);
        gtk_box_pack_start (GTK_BOX (gs_hbox3), gs_image3, FALSE, FALSE, 0);

        gs_label11 = gtk_label_new_with_mnemonic ("_Default");
        gtk_widget_show (gs_label11);
        gtk_box_pack_start (GTK_BOX (gs_hbox3), gs_label11,
                            FALSE, FALSE, 0);
    }

    /* Setting default value */
    gs_default_value (NULL, NULL);
}

/*****************************************************************************/
/* GS_PAR_INIT:                                                              */
/*****************************************************************************/

void gs_par_init (void)
{
    plotfo_def    = 1;
    plotsdhv_def  = 1;
    plotsdfo_def  = 1;
    imgfmt_def    = 0;

    plotfo    = plotfo_def;
    plotsdhv  = plotsdhv_def;
    plotsdfo  = plotsdfo_def;
    imgfmt    = imgfmt_def;

    if (imgfmt == 0)
    {
        sprintf (imgext, "jpg");
    }
    if (imgfmt == 1)
    {
        sprintf (imgext, "png");
    }
}
```

```
/*****************************************************************************/
/* GS_APPLY_VALUE:                                                           */
/*****************************************************************************/

void gs_apply_value (GtkWidget *widget, gpointer data)
{
    plotfo   = gtk_toggle_button_get_active
               (GTK_TOGGLE_BUTTON (gs_checkbutton1));
    plotsdhv = gtk_toggle_button_get_active
               (GTK_TOGGLE_BUTTON (gs_checkbutton2));
    plotsdfo = gtk_toggle_button_get_active
               (GTK_TOGGLE_BUTTON (gs_checkbutton3));

    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (gs_radiobutton1)))
    {
        imgfmt = 0;
        sprintf (imgext, "jpg");
    }
    if (gtk_toggle_button_get_active (GTK_TOGGLE_BUTTON (gs_radiobutton2)))
    {
        imgfmt = 1;
        sprintf (imgext, "png");
    }
}

/*****************************************************************************/
/* GS_UNDO_VALUE:                                                            */
/*****************************************************************************/

void gs_undo_value (GtkWidget *widget, gpointer data)
{
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (gs_checkbutton1),
                                  plotfo);
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (gs_checkbutton2),
                                  plotsdhv);
    gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (gs_checkbutton3),
                                  plotsdfo);

    if ( imgfmt == 0)
    {
        gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (gs_radiobutton1),
                                      TRUE);
    }
    if ( imgfmt == 1)
    {
        gtk_toggle_button_set_active (GTK_TOGGLE_BUTTON (gs_radiobutton2),
                                      TRUE);
    }
}

/*****************************************************************************/
/* GS_DEFAULT_VALUE:                                                         */
/*****************************************************************************/

void gs_default_value (GtkWidget *widget, gpointer data)
{
    gs_par_init ();
    gs_undo_value (widget, data);
}
```

1.24 - Gsescfg.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef GSESCFG_H
#define GSESCFG_H

    void    gs_window_showhide      (void);

    void    create_gs_par_window    (void);

    void    gs_par_init             (void);

    void    gs_apply_value          (GtkWidget *widget,
                                     gpointer data );

    void    gs_undo_value           (GtkWidget *widget,
                                     gpointer data);

    void    gs_default_value        (GtkWidget *widget,
                                     gpointer data);

# endif
```

1.25 - Primitives.c

```
# include <gtk/gtk.h>

# include "globals.h"
# include "primitives.h"

/*****************************************************************************/
/* RGB_BUFFER_DRAW_POINT:                                                    */
/*****************************************************************************/

void rgb_buffer_draw_point (guchar *rgb_buffer,
                            gint image_width,
                            gint image_higth,
                            gint x,
                            gint y,
                            gint *rgb_color)
{
    gint index;

    index = (((y * image_width) + x) * 3);
    if (x >= 0 && y >= 0)
    {
        if (index < (image_width * image_higth * 3))
        {
            rgb_buffer [index + 0] = rgb_color[0];
            rgb_buffer [index + 1] = rgb_color[1];
            rgb_buffer [index + 2] = rgb_color[2];
        }
    }
}

/*****************************************************************************/
/* RGB_BUFFER_DRAW_LINE:                                                     */
```

```
/*************************************************************************/

void rgb_buffer_draw_line (guchar *rgb_buffer,
                           gint image_width,
                           gint image_higth,
                           gint x1, gint y1,
                           gint x2, gint y2,
                           gint *rgb_color)
{
    /*****************************************************************/
    /* This code has been created by "joed"; it's a free interpretation */
    /* of the Bresenham's algorithm for drawing line.                */
    /* I found it in a forum at the url:                             */
    /* http://cboard.cprogramming.com/printthread.php?t=67832        */
    /* Thanks to joed!                                               */
    /*****************************************************************/

    gint dx, dy, inx, iny, e;

    dx = x2 - x1;
    dy = y2 - y1;
    inx = dx > 0 ? 1 : -1;
    iny = dy > 0 ? 1 : -1;

    dx = ABS(dx);
    dy = ABS(dy);

    if(dx >= dy)
    {
        dy <<= 1;
        e = dy - dx;
        dx <<= 1;
        while (x1 != x2)
        {
            rgb_buffer_draw_point (rgb_buffer,image_width,image_higth,
                                   x1,y1,rgb_color);
            if(e >= 0)
            {
                y1 += iny;
                e-= dx;
            }
            e += dy;
            x1 += inx;
        }
    }
    else
    {
        dx <<= 1;
        e = dx - dy;
        dy <<= 1;
        while (y1 != y2)
        {
            rgb_buffer_draw_point (rgb_buffer,image_width,image_higth,
                                   x1,y1,rgb_color);
            if(e >= 0)
            {
                x1 += inx;
                e -= dy;
            }
            e += dx;
            y1 += iny;
        }
    }
```

```
    rgb_buffer_draw_point (rgb_buffer,image_width,image_higth,
                           x1,y1,rgb_color);
}

/*****************************************************************************/
/* RGB_BUFFER_SET_BG_COLOR:                                                  */
/*****************************************************************************/

void rgb_buffer_set_bg_color (guchar *rgb_buffer,
                              gint image_width,
                              gint image_higth,
                              gint *rgb_color)
{
    gint x,y;

    for (y = 0; y < image_higth; y++)
    {
        for (x = 0; x < image_width; x++)
        {
            *rgb_buffer++ = rgb_color[0];
            *rgb_buffer++ = rgb_color[1];
            *rgb_buffer++ = rgb_color[2];
        }
    }
}

/*****************************************************************************/
/* SET_GC_COLOR:                                                             */
/*****************************************************************************/

GdkGC *set_gc_color (GtkWidget *widget,
                     gushort red,
                     gushort green,
                     gushort blue)
{
    GdkGC *gc;
    GdkColor color;
    GdkColormap *cmap;

    color.red   = red;
    color.green = green;
    color.blue  = blue;

    cmap = gdk_colormap_get_system ();
    gdk_color_alloc (cmap, &color);

    gc = gdk_gc_new (widget->window);
    gdk_gc_copy (gc, widget->style->black_gc);

    gdk_gc_set_foreground (gc,&color);

    return gc;
}

/*****************************************************************************/
/* DRAW_STRING:                                                              */
/*****************************************************************************/

void draw_string (GtkWidget *widget,
                  GdkPixmap *pixmap,
                  gchar *string,
                  gint x,
                  gint y,
```

```
                  gint *rgb_color)
{
    GdkGC *gc_color;
    GdkFont *font;

    rgb_color[0] = ((rgb_color[0] / 255) * 65535);
    rgb_color[1] = ((rgb_color[1] / 255) * 65535);
    rgb_color[2] = ((rgb_color[2] / 255) * 65535);

    gc_color = set_gc_color (widget,rgb_color[0],rgb_color[1],rgb_color[2]);
    font = gdk_font_load
           ( "-adobe-helvetica-bold-r-normal-*-10-*-*-*-*-*-*-*" );

    gdk_draw_string (pixmap, font, gc_color, x, y, string );

    gdk_font_unref (font);
}

/*****************************************************************************/
/* GDK_PIXMAP_SAVE:                                                          */
/*****************************************************************************/

void gdk_pixmap_save (GdkPixmap *pixmap,
                      gchar *string)
{
    GdkPixbuf *pixbuf;

    pixbuf = gdk_pixbuf_get_from_drawable (NULL,pixmap,NULL,0,0,0,0,-1,-1);

    if (imgfmt == 0)
    {
        gdk_pixbuf_save (pixbuf,string,"jpeg",NULL,"quality","100",NULL);
    }

    if (imgfmt == 1)
    {
        gdk_pixbuf_save (pixbuf,string,"png",NULL,NULL,NULL,NULL);
    }

    g_object_unref (pixbuf);
}
```

1.26 - Primitives.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef PRIMITIVES_H
#define PRIMITIVES_H

void rgb_buffer_draw_point   (guchar *rgb_buffer,
                              gint image_width,
                              gint image_higth,
                              gint x,
                              gint y,
                              gint *rgb_color);

void rgb_buffer_draw_line    (guchar *rgb_buffer,
                              gint image_width,
                              gint image_higth,
```

```
                                gint x1, gint y1,
                                gint x2, gint y2,
                                gint *rgb_color);

void rgb_buffer_set_bg_color (guchar *rgb_buffer,
                                gint image_width,
                                gint image_higth,
                                gint *rgb_color);

GdkGC *set_gc_color           (GtkWidget *widget,
                                gushort red,
                                gushort green,
                                gushort blue);

void draw_string              (GtkWidget *widget,
                                GdkPixmap *pixmap,
                                gchar *string,
                                gint x,
                                gint y,
                                gint *rgb_color);

void gdk_pixmap_save          (GdkPixmap *pixmap,
                                gchar *string);

# endif
```

1.27 - Plotch.c

```
# include <gtk/gtk.h>
# include <stdlib.h>
# include <string.h>
# include <math.h>

# include "globals.h"
# include "plotch.h"
# include "primitives.h"
# include "platdep.h"
# include "warning.h"

/*****************************************************************************/
/* PLOT_CHANNEL:                                                             */
/*****************************************************************************/

void plot_channel (GtkWidget *widget, gchar *channel_str)
{
    gchar title_bar[200];
    gint x;
    gint y;

    /*************************************************************************/
    /* Inizializing                                                          */
    /*************************************************************************/

    /* Inizializing the widget structure */

    PLOT_STRUCT *plot = NULL;
    plot = g_try_malloc (sizeof (PLOT_STRUCT));
    plot->window = NULL;
    plot->pixmap = NULL;

    /* RGB initialization for trace plot */
```

```
gdk_rgb_init();

/* Setting the widget appearance */

plot->image_width  = 900;    /* image width  */
plot->image_height = 220;    /* image height */
plot->lb = 20;               /* left border  */
plot->rb = 20;               /* right border */
plot->ub = 20;               /* upper border */
plot->db = 20;               /* lower border */
x = 50;                      /* x position   */
y = 400;                     /* y position   */

/* Item selection Check */

plot->model = gtk_tree_view_get_model (GTK_TREE_VIEW(view));
plot->selection = gtk_tree_view_get_selection (GTK_TREE_VIEW(view));

if (gtk_tree_selection_get_selected (plot->selection,NULL,&(plot->iter)))
{
    gtk_tree_model_get (plot->model,&(plot->iter),9,&(plot->ptr_list),-1);
    gtk_tree_model_get (plot->model,&(plot->iter),0,&(plot->file),-1);
    gtk_tree_model_get (plot->model,&(plot->iter),8,&(plot->unit),-1);

    /* Setting the channel */

    if (strstr(channel_str,"Channel 0") == channel_str)
    {
        plot->ch = 0;
    }
    if (strstr(channel_str,"Channel 1") == channel_str)
    {
        plot->ch = 1;
    }
    if (strstr(channel_str,"Channel 2") == channel_str)
    {
        plot->ch = 2;
    }

    /* Window title bar */

    sprintf (title_bar,"%s  File: %s", channel_str, plot->ptr_list->file);

    /* Getting information from selected SAF file */

    ch_data_read (plot);

    /*********************************************************************/
    /* Plot window                                                       */
    /*********************************************************************/
    {
        plot->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_window_move (GTK_WINDOW (plot->window), x, y);
        gtk_window_set_resizable (GTK_WINDOW (plot->window), FALSE);
        gtk_window_set_title (GTK_WINDOW(plot->window),title_bar);
    }

    {
        plot->vbox1 = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (plot->vbox1);
        gtk_container_add (GTK_CONTAINER (plot->window),
                           plot->vbox1);
```

```
}

{
    plot->toolbar1 = gtk_toolbar_new ();
    gtk_widget_show (plot->toolbar1);
    gtk_box_pack_start (GTK_BOX (plot->vbox1), plot->toolbar1,
                        FALSE, TRUE, 0);
    gtk_toolbar_set_style (GTK_TOOLBAR (plot->toolbar1),
                           GTK_TOOLBAR_ICONS);
}

{
    plot->button1 = (GtkWidget*)gtk_tool_button_new_from_stock
                    ("gtk-save");
    gtk_widget_show (plot->button1);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->button1);
}

{
    plot->toolitem1 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem1);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem1);
    gtk_widget_set_size_request (plot->toolitem1, 10, -1);

    plot->vseparator1 = gtk_vseparator_new ();
    gtk_widget_show (plot->vseparator1);
    gtk_container_add (GTK_CONTAINER (plot->toolitem1),
                       plot->vseparator1);
}

{
    plot->toolitem2 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem2);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem2);

    plot->label1 = gtk_label_new ("<b> Plot start (seconds): </b>");
    gtk_widget_show (plot->label1);
    gtk_container_add (GTK_CONTAINER (plot->toolitem2),
                       plot->label1);
    gtk_label_set_use_markup (GTK_LABEL (plot->label1), TRUE);
}

{
    plot->toolitem3 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem3);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem3);

    plot->spinbutton1_adj = gtk_adjustment_new
                            (0.0, 0.0,
                            (gdouble)(plot->ndat - 1)/
                            (gdouble)(plot->freq),
                             1, 10, 10);

    plot->spinbutton1 = gtk_spin_button_new
    (GTK_ADJUSTMENT (plot->spinbutton1_adj), 1, 2);
    gtk_widget_show (plot->spinbutton1);
    gtk_container_add (GTK_CONTAINER (plot->toolitem3),
                       plot->spinbutton1);
}
```

```
{
    plot->toolitem4 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem4);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem4);
    gtk_widget_set_size_request (plot->toolitem4, 10, -1);

    plot->vseparator2 = gtk_vseparator_new ();
    gtk_widget_show (plot->vseparator2);
    gtk_container_add (GTK_CONTAINER (plot->toolitem4),
                       plot->vseparator2);
}

{
    plot->toolitem5 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem5);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem5);

    plot->label2 = gtk_label_new ("<b> Plot end (seconds): </b>");
    gtk_widget_show (plot->label2);
    gtk_container_add (GTK_CONTAINER (plot->toolitem5),
                       plot->label2);
    gtk_label_set_use_markup (GTK_LABEL (plot->label2), TRUE);
}

{
    plot->toolitem6 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem6);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem6);

    plot->spinbutton2_adj = gtk_adjustment_new
                            ((gdouble)(plot->ndat - 1),
                             (gdouble)(1)/
                             (gdouble)(plot->freq),
                             (gdouble)(plot->ndat)/
                             (gdouble)(plot->freq),
                              1, 10, 10);

    plot->spinbutton2 = gtk_spin_button_new
    (GTK_ADJUSTMENT (plot->spinbutton2_adj), 1, 2);
    gtk_widget_show (plot->spinbutton2);
    gtk_container_add (GTK_CONTAINER (plot->toolitem6),
                       plot->spinbutton2);
}

{
    plot->toolitem7 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (plot->toolitem7);
    gtk_container_add (GTK_CONTAINER (plot->toolbar1),
                       plot->toolitem7);
    gtk_widget_set_size_request (plot->toolitem7, 10, -1);

    plot->vseparator3 = gtk_vseparator_new ();
    gtk_widget_show (plot->vseparator3);
    gtk_container_add (GTK_CONTAINER (plot->toolitem7),
                       plot->vseparator3);
}

{
    plot->drawing_area = gtk_drawing_area_new ();
```

```
            gtk_drawing_area_size (GTK_DRAWING_AREA (plot->drawing_area),
                                   plot->image_width,
                                   plot->image_height);
            gtk_box_pack_start (GTK_BOX (plot->vbox1), plot->drawing_area,
                                FALSE, TRUE, 0);
        }

        /*************************************************************************/
        /* Setting signals and events                                            */
        /*************************************************************************/
        {
            g_signal_connect (GTK_OBJECT (plot->window),
                              "delete_event",
                              G_CALLBACK (ch_delete_event),
                             (gpointer) plot);
            g_signal_connect (G_OBJECT (plot->button1),"clicked",
                              G_CALLBACK (ch_save_image),
                             (gpointer) plot);
            g_signal_connect (GTK_OBJECT (plot->spinbutton1),
                              "value-changed",
                              G_CALLBACK (ch_set_start_event),
                             (gpointer) plot);
            g_signal_connect (GTK_OBJECT (plot->spinbutton2),
                              "value-changed",
                              G_CALLBACK (ch_set_stop_event),
                             (gpointer) plot);
            g_signal_connect (G_OBJECT (plot->drawing_area),
                              "configure_event",
                              G_CALLBACK (ch_configure_event),
                             (gpointer) plot);
            g_signal_connect (GTK_OBJECT (plot->drawing_area),
                              "expose_event",
                              G_CALLBACK (ch_expose_event),
                             (gpointer) plot);
            g_signal_connect (GTK_OBJECT (plot->drawing_area),
                              "button_press_event",
                              G_CALLBACK (ch_plot_press_event),
                             (gpointer) plot);
            g_signal_connect (GTK_OBJECT (plot->drawing_area),
                              "button_release_event",
                              G_CALLBACK (ch_plot_release_event),
                             (gpointer) plot);


            gtk_widget_set_events (plot->drawing_area,
                                     GDK_EXPOSURE_MASK
                                   | GDK_LEAVE_NOTIFY_MASK
                                   | GDK_BUTTON_PRESS_MASK
                                   | GDK_BUTTON_RELEASE_MASK
                                   | GDK_POINTER_MOTION_MASK
                                   | GDK_POINTER_MOTION_HINT_MASK);
        }

        gtk_widget_show_all (plot->window);
    }
    else
    /* Emit Warning: no item selected */
    {
        warning_no_item_selected ("<b>No item selected!</b>");
    }
}

/*****************************************************************************/
```

```
/* CH_CONFIGURE_EVENT:                                                        */
/******************************************************************************/

gboolean ch_configure_event (GtkWidget *widget,
                             GdkEventConfigure *event,
                             PLOT_STRUCT *plot)
{
    /**************************************************************************/
    /* Inizializing the pixmap                                                */
    /**************************************************************************/

    if (plot->pixmap)
    {
        g_object_unref (G_OBJECT (plot->pixmap));
    }

    plot->pixmap = gdk_pixmap_new (widget->window,plot->image_width,
                                   plot->image_height,-1);

    /**************************************************************************/
    /* Trace plot                                                             */
    /**************************************************************************/

    plot->start = 0;
    plot->stop = (plot->ndat - 1);

    draw_trace_plot (widget, plot);

    return TRUE;
}

/******************************************************************************/
/* CH_EXPOSE_EVENT:                                                           */
/******************************************************************************/

gboolean ch_expose_event (GtkWidget *widget,
                          GdkEventExpose *event,
                          PLOT_STRUCT *plot)
{
    gdk_draw_drawable (widget->window,
                       widget->style->white_gc,
                       plot->pixmap,
                       event->area.x, event->area.y,
                       event->area.x, event->area.y,
                       event->area.width, event->area.height);

    return FALSE;
}

/******************************************************************************/
/* CH_PLOT_PRESS_EVENT:                                                       */
/******************************************************************************/

gint ch_plot_press_event (GtkWidget *widget,
                          GdkEventButton *event,
                          PLOT_STRUCT *plot)
{
    if ((event->button) == 1)
    {
        if ((event->x) > (plot->lb) &&
            (event->x) < (plot->image_width - plot->lb))
        {
            plot->tmp_sta =
```

```
            (((plot->stop - plot->start) * (event->x - plot->lb)) /
            (plot->image_width - plot->lb - plot->rb)) + plot->start;
        }
    }

    return TRUE;
}

/*****************************************************************************/
/* CH_PLOT_RELEASE_EVENT:                                                    */
/*****************************************************************************/

gint ch_plot_release_event (GtkWidget *widget,
                            GdkEventButton *event,
                            PLOT_STRUCT *plot)
{
    if ((event->button) == 1)
    {
        if ((event->x) > (plot->lb) &&
            (event->x) < (plot->image_width - plot->lb))
        {
            plot->tmp_sto =
            (((plot->stop - plot->start) * (event->x - plot->lb)) /
            (plot->image_width - plot->lb - plot->rb)) + plot->start + 1;

            if (plot->tmp_sto > plot->tmp_sta)
            {
                plot->stop = plot->tmp_sto;
                plot->start = plot->tmp_sta;
            }
            else
            {
                plot->start = plot->tmp_sto;
                plot->stop = plot->tmp_sta;
            }
        }
    }
    else
    {
        plot->start = 0;
        plot->stop = (plot->ndat - 1);
    }

    gtk_spin_button_set_value (GTK_SPIN_BUTTON(plot->spinbutton1),
                               (gdouble)(plot->start)/
                               (gdouble)(plot->freq));

    gtk_spin_button_set_value (GTK_SPIN_BUTTON(plot->spinbutton2),
                               (gdouble)(plot->stop + 1)/
                               (gdouble)(plot->freq));

    /*************************************************************************/
    /*  The following lines are optional, because redrawing is forced        */
    /*  by the previous "gtk_spin_button_set_value" calls.                   */
    /*************************************************************************/
    /*  draw_trace_plot (widget, plot);                                      */
    /*  gtk_widget_queue_draw_area(widget, 0, 0,                             */
    /*                             plot->image_width,                        */
    /*                             plot->image_height);                      */
    /*************************************************************************/

    return TRUE;
}
```

```
/*****************************************************************************/
/* CH_SET_START_EVENT:                                                       */
/*****************************************************************************/

gint ch_set_start_event (GtkWidget *widget,
                         PLOT_STRUCT *plot)
{
    gint tmp_start = (gdouble)(gtk_spin_button_get_value
                     (GTK_SPIN_BUTTON(plot->spinbutton1))*
                     (gdouble)(plot->freq));

    if (tmp_start < plot->stop)
    {
        plot->start = tmp_start;

        draw_trace_plot (plot->drawing_area, plot);

        gtk_widget_queue_draw_area(plot->drawing_area, 0, 0,
                                   plot->image_width,
                                   plot->image_height);
    }

    return 0;
}

/*****************************************************************************/
/* CH_SET_STOP_EVENT:                                                        */
/*****************************************************************************/

gint ch_set_stop_event (GtkWidget *widget,
                        PLOT_STRUCT *plot)
{
    gint tmp_stop = (gdouble)(gtk_spin_button_get_value
                    (GTK_SPIN_BUTTON(plot->spinbutton2))*
                    (gdouble)(plot->freq));

    if (tmp_stop > plot->start)
    {
        plot->stop = tmp_stop;

        draw_trace_plot (plot->drawing_area, plot);

        gtk_widget_queue_draw_area(plot->drawing_area, 0, 0,
                                   plot->image_width,
                                   plot->image_height);
    }

    return 0;
}

/*****************************************************************************/
/* CH_DELETE_EVENT:                                                          */
/*****************************************************************************/

gboolean ch_delete_event (GtkWidget *widget,
                          GdkEvent  *event,
                          PLOT_STRUCT *plot)
{
    gint j;

    /* Signal disconnection */
```

```
    gtk_signal_disconnect_by_func (GTK_OBJECT (plot->spinbutton1),
                                   G_CALLBACK (ch_set_start_event),
                                   (gpointer) plot);

    gtk_signal_disconnect_by_func (GTK_OBJECT (plot->spinbutton2),
                                   G_CALLBACK (ch_set_stop_event),
                                   (gpointer) plot);

    /* Memory deallocation */

    for (j=0;j<3;j++)
    {
        if (plot->data[j] != NULL)
        {
            g_free (plot->data[j]);;
        }
    }

    return FALSE;
}

/*****************************************************************************/
/* DRAW_TRACE_PLOT:                                                          */
/*****************************************************************************/

void draw_trace_plot (GtkWidget *widget,
                      PLOT_STRUCT *plot)
{
    gint sta = plot->start;
    gint sto  = plot->stop;

    gint w = plot->image_width;    /* Window width  */
    gint h = plot->image_height;   /* Window height */

    gint lb = plot->lb;        /* left border  */
    gint rb = plot->rb;        /* right border */
    gint ub = plot->ub;        /* upper border */
    gint db = plot->db;        /* lower border */

    gint tw = w - lb - rb;   /* Trace width  */
    gint th = h - ub - db;   /* Trace height */

    gint x1, y1, x2, y2;

    gdouble p1, p2;

    gdouble max = plot->data_max[plot->ch];

    guchar rgbbuf [w * h * 3];

    gint blue  [3] = {100, 100, 255};
    gint black [3] = {  0,   0,   0};
    gint white [3] = {255, 255, 255};

    gint nt = 10;     /* Number of time marks                    */
    gint na = 6;      /* Number of amplitude marks: must be odd!*/

    gint dt;      /* Time mark increment      */
    gint da;      /* Amplitude mark increment */

    gdouble t_label;

    gint i;
```

```
gchar header[200];
gchar string[200];

/*************************************************************************/
/* Setting the background color of the image                             */
/*************************************************************************/

rgb_buffer_set_bg_color (rgbbuf,w,h,white);

/*************************************************************************/
/* Traces plot                                                           */
/*************************************************************************/

if ((plot->data[plot->ch] != NULL))
{
    for (i = sta; i < sto; i++)
    {
        p1 = plot->data[plot->ch][i];
        p2 = plot->data[plot->ch][i+1];

        x1 = (((i - sta) * tw) / (sto - sta)) + lb;
        x2 = ((((i + 1) - sta) * tw) / (sto - sta)) + lb;

        y1 = ((p1 * (th / 2)) / max) + (th / 2) + ub;
        y2 = ((p2 * (th / 2)) / max) + (th / 2) + ub;

        rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,blue);
    }
}

/*************************************************************************/
/* Traces frame                                                          */
/*************************************************************************/

/* BOX */;
{
    rgb_buffer_draw_line (rgbbuf,w,h,lb,ub,(lb+tw),ub,black);
    rgb_buffer_draw_line (rgbbuf,w,h,lb,(ub+th),(lb+tw),(ub+th),black);
    rgb_buffer_draw_line (rgbbuf,w,h,lb,ub,lb,(ub+th),black);
    rgb_buffer_draw_line (rgbbuf,w,h,(lb+tw),ub,(lb+tw),(ub+th),black);

}

/* TIME MARK */

for(i = 0; i < nt; i++)
{
    dt = ((i * tw) / nt);

    rgb_buffer_draw_line
    (rgbbuf,w,h,(lb+dt),ub,(lb+dt),(ub+5),black);

    rgb_buffer_draw_line
    (rgbbuf,w,h,(lb+dt),(ub+th),(lb+dt),(ub+th-5),black);
}

/* AMPLITUDE MARK */

for(i = 0; i <= na; i++)
{
    da = ((i * th) / na);

    rgb_buffer_draw_line
```

```
        (rgbbuf,w,h,lb,(ub+da),(lb+5),(ub+da),black);

        rgb_buffer_draw_line
        (rgbbuf,w,h,(lb+tw-5),(ub+da),(lb+tw),(ub+da),black);
    }

    /*************************************************************************/
    /* Copying rgm to pixmap                                                 */
    /*************************************************************************/

    if (plot->pixmap)
    {
        gdk_draw_rgb_image (plot->pixmap,
                            widget->style->white_gc,
                            0, 0,
                            plot->image_width, plot->image_height,
                            GDK_RGB_DITHER_MAX,
                            rgbbuf,
                            plot->image_width * 3);
    }

    /*************************************************************************/
    /* Label                                                                 */
    /*************************************************************************/

    /* HEADER */

    {
        sprintf (header, "CHANNEL:  %d      ", plot->ch);

        sprintf (string, "FILE:  %s      ", plot->file);
        strcat (header, string);

        sprintf (string, "START TIME:  %s      ", plot->ptr_list->start_time);
        strcat (header, string);

        sprintf (string, "TIME UNIT:  Second      ");
        strcat (header, string);

        sprintf (string, "FULL SCALE:  %.2e %s", max, plot->unit);
        strcat (header, string);
    }

    draw_string (widget,plot->pixmap,header,lb,(ub-5),black);

    /* TIME LABEL */

    for(i = 0; i < nt; i++)
    {
        dt = ((i * tw) / nt);

        t_label = ((gdouble) sta / (gdouble) plot->freq) +
                  ((gdouble) ((sto - sta) * i) /
                   (gdouble) (plot->freq * nt));

        sprintf (string, "%.2f", t_label);
        draw_string (widget, plot->pixmap,string,(lb+dt),(ub+th+14),black);
    }
}

/*************************************************************************/
/* CH_DATA_READ:                                                         */
/*************************************************************************/
```

```
PLOT_STRUCT *ch_data_read (PLOT_STRUCT *plot)
{
    FILE *fp;
    gint i,j;
    gchar string[100];

    plot->ndat = atof (plot->ptr_list->ndat);
    plot->freq = atof (plot->ptr_list->samp_freq);

    /* Initialization */
    {
        plot->data_max_all = 0;

        for (j=0;j<3;j++)
        {
            plot->data_max[j] = 0;
            plot->data[j] = (gdouble *)g_try_malloc
                            (sizeof(gdouble)*(plot->ndat));
        }
    }

    if (((fp = fopen (plot->ptr_list->file,"r")) != NULL) &&
        (plot->data[0] != NULL) &&
        (plot->data[1] != NULL) &&
        (plot->data[2] != NULL))
    {
        /* Header */
        while(strstr (string,"####----") != string)
        {
            fgets (string,100,fp);
        }

        /* Data and Max value */
        for (i=0;i<(plot->ndat);i++)
        {
            if (!feof(fp))
            {
                for (j=0;j<3;j++)
                {
                    fscanf(fp,"%lf",&(plot->data[j][i]));

                    /* Single Channel Max */
                    if (fabs (plot->data[j][i]) > fabs (plot->data_max[j]))
                    {
                        plot->data_max[j] = plot->data[j][i];
                    }
                }
            }
        }

        /* Total Max */
        for (j=0;j<3;j++)
        {
            if (fabs (plot->data_max[j]) > fabs (plot->data_max_all))
            {
                plot->data_max_all = plot->data_max[j];
            }
        }

        fclose(fp);
    }
    return plot;
```

```
}

/*****************************************************************************/
/* CH_SAVE_IMAGE:                                                            */
/*****************************************************************************/

void ch_save_image (GtkWidget *widget,
                    PLOT_STRUCT *plot)
{
    gchar img_name[500];

    sprintf(img_name, "%sCh_%d.%s.%s", img_path, plot->ch, plot->file, imgext);

    /* File selection dialog window */
    plot->img_selection = gtk_file_selection_new ("Save Image");

    /* Set the default filename */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(plot->img_selection),
                                     img_name);

    /* OK button signal connection */
    g_signal_connect (GTK_FILE_SELECTION(plot->img_selection)->ok_button,
                      "clicked",
                      G_CALLBACK(ch_overwrite_image),
                      (gpointer)plot);

    /* Cancel button signal connection */
    g_signal_connect_swapped
                     (GTK_FILE_SELECTION(plot->img_selection)->cancel_button,
                      "clicked",
                      G_CALLBACK(gtk_widget_destroy),
                      plot->img_selection);

    gtk_widget_show (plot->img_selection);
}

/*****************************************************************************/
/* CH_OVERWRITE_IMAGE:                                                       */
/*****************************************************************************/

void ch_overwrite_image (GtkWidget *widget,
                         PLOT_STRUCT *plot)
{
    const gchar *selected_filename;
    gchar img_filename[200];
    GtkWidget *overwrite;
    GtkWidget *hbox;
    GtkWidget *stock_image;
    GtkWidget *label;
    gint response;
    FILE *fp;

    selected_filename = gtk_file_selection_get_filename
                        (GTK_FILE_SELECTION(plot->img_selection));

    strcpy (img_filename, selected_filename);

    /* File exists */
    if ((fp = fopen(selected_filename, "r")) != NULL)
    {
        /* Dialog window */
        overwrite = gtk_dialog_new_with_buttons
                    ("Warning!",
```

```
                    NULL,
                    GTK_DIALOG_MODAL |
                    GTK_DIALOG_DESTROY_WITH_PARENT,
                    GTK_STOCK_APPLY,
                    GTK_RESPONSE_ACCEPT,
                    GTK_STOCK_CANCEL,
                    GTK_RESPONSE_REJECT,
                    NULL);

        /* Horizontal box */
        {
            hbox = gtk_hbox_new (FALSE,10);
            gtk_container_set_border_width (GTK_CONTAINER(hbox),10);
            gtk_box_pack_start (GTK_BOX(GTK_DIALOG(overwrite)->vbox),
                                hbox,FALSE,FALSE,0);
        }

        /* Dialog question stock image and label */
        {
            stock_image = gtk_image_new_from_stock (GTK_STOCK_DIALOG_QUESTION,
                                                    GTK_ICON_SIZE_DIALOG);
            gtk_box_pack_start (GTK_BOX(hbox),stock_image,FALSE,FALSE,0);

            label = gtk_label_new_with_mnemonic ("File exists: overwrite?");
            gtk_box_pack_start (GTK_BOX(hbox),label,FALSE,FALSE,0);
        }

        gtk_widget_show_all (overwrite);

        /* Response evaluation */
        {
        response = gtk_dialog_run (GTK_DIALOG(overwrite));

            if (response == GTK_RESPONSE_ACCEPT)
            {
                if (plot->pixmap)
                {
                    gdk_pixmap_save (plot->pixmap, img_filename);
                }
                gtk_widget_destroy (GTK_WIDGET(plot->img_selection));
            }
        }

        gtk_widget_destroy (overwrite);
        fclose (fp);
    }

    /* File doesn't exist */
    else
    {
        if (plot->pixmap)
        {
            gdk_pixmap_save (plot->pixmap, img_filename);
        }
        gtk_widget_destroy (GTK_WIDGET(plot->img_selection));
    }
}
```

1.28 - Plotch.h

```
#ifndef PLOTCH_H
```

```
#define PLOTCH_H

/*****************************************************************************/
/* Variables.                                                                */
/*****************************************************************************/

typedef struct PLOT_STRUCT
{
    GtkWidget          *window;
    GdkPixmap          *pixmap;
    GtkWidget          *vbox1;
    GtkWidget          *toolbar1;
    GtkWidget          *button1;
    GtkWidget          *toolitem1;
    GtkWidget          *vseparator1;
    GtkWidget          *toolitem2;
    GtkWidget          *label1;
    GtkWidget          *toolitem3;
    GtkObject          *spinbutton1_adj;
    GtkWidget          *spinbutton1;
    GtkWidget          *toolitem4;
    GtkWidget          *vseparator2;
    GtkWidget          *toolitem5;
    GtkWidget          *label2;
    GtkWidget          *toolitem6;
    GtkObject          *spinbutton2_adj;
    GtkWidget          *spinbutton2;
    GtkWidget          *toolitem7;
    GtkWidget          *vseparator3;
    GtkWidget          *drawing_area;
    GtkWidget          *img_selection;
    GtkTreeSelection   *selection;
    GtkTreeModel       *model;
    GtkTreeIter         iter;
    gdouble            *data[3];
    gdouble             data_max[3];
    gdouble             data_max_all;
    gint                ndat;
    gint                start;
    gint                tmp_sta;
    gint                stop;
    gint                tmp_sto;
    gint                freq;
    guint               ch;
    gchar              *file;
    gchar              *unit;
    gint                image_width;
    gint                image_height;
    gint                lb;
    gint                rb;
    gint                ub;
    gint                db;
    SAF_LIST_PTR        ptr_list;
} PLOT_STRUCT;

/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

void            plot_channel            (GtkWidget *widget,
                                         gchar *channel_str);

gboolean        ch_configure_event      (GtkWidget *widget,
```

```
                                        GdkEventConfigure *event,
                                        PLOT_STRUCT *plot);

gboolean        ch_expose_event         (GtkWidget *widget,
                                        GdkEventExpose *event,
                                        PLOT_STRUCT *plot);

gint            ch_plot_press_event     (GtkWidget *widget,
                                        GdkEventButton *event,
                                        PLOT_STRUCT *plot);

gint            ch_plot_release_event   (GtkWidget *widget,
                                        GdkEventButton *event,
                                        PLOT_STRUCT *plot);

gint            ch_set_start_event      (GtkWidget *widget,
                                        PLOT_STRUCT *plot);

gint            ch_set_stop_event       (GtkWidget *widget,
                                        PLOT_STRUCT *plot);

gboolean        ch_delete_event         (GtkWidget *widget,
                                        GdkEvent  *event,
                                        PLOT_STRUCT *plot);

void            draw_trace_plot         (GtkWidget *widget,
                                        PLOT_STRUCT *plot);

PLOT_STRUCT    *ch_data_read            (PLOT_STRUCT *plot);

void            ch_save_image           (GtkWidget *widget,
                                        PLOT_STRUCT *plot);

void            ch_overwrite_image      (GtkWidget *widget,
                                        PLOT_STRUCT *plot);

# endif
```

1.29 - Plothv.c

```
# include <gtk/gtk.h>
# include <stdlib.h>
# include <string.h>
# include <math.h>

# include "globals.h"
# include "plothv.h"
# include "primitives.h"
# include "platdep.h"
# include "warning.h"

/*****************************************************************************/
/* PLOT_HVSP:                                                                */
/*****************************************************************************/

void plot_hvsp (GtkWidget *widget, gchar *hvsp_str)
{
    gchar title_bar[200];
    gchar file_name_tmp[500];
    gchar *file_name;
    gint x;
```

```
gint y;
FILE *fp;

/*************************************************************************/
/* Inizializing                                                          */
/*************************************************************************/

/* Inizializing the widget structure */

HV_STRUCT *hv = NULL;
hv = g_try_malloc (sizeof (HV_STRUCT));
hv->window = NULL;
hv->pixmap = NULL;

/* RGB initialization for trace plot */

gdk_rgb_init();

/* Setting the plot type */

hv->hvsp_type = hvsp_str;

/* Plot type switch: spectral ratio vs. absolute spectra */

if (strstr(hvsp_str,"Average H/V") == hvsp_str)
{
    hv->type = 0;
    hv->ch = 0;
}
if (strstr(hvsp_str,"H(1)/V") == hvsp_str)
{
    hv->type = 0;
    hv->ch = 1;
}
if (strstr(hvsp_str,"H(2)/V") == hvsp_str)
{
    hv->type = 0;
    hv->ch = 2;
}
if (strstr(hvsp_str,"Ch 0 Spectrum") == hvsp_str)
{
    hv->type = 1;
    hv->ch = 0;
}
if (strstr(hvsp_str,"Ch 1 Spectrum") == hvsp_str)
{
    hv->type = 1;
    hv->ch = 1;
}
if (strstr(hvsp_str,"Ch 2 Spectrum") == hvsp_str)
{
    hv->type = 1;
    hv->ch = 2;
}

/* Setting the widget appearance */

hv->image_width  = 700;    /* image width  */
hv->image_height = 400;    /* image height */

if (hv->type == 0)
{
    hv->lb = 35;           /* left border  */
```

```
}
else
{
    hv->lb = 70;           /* left border  */
}

hv->rb = 35;               /* right border */
hv->ub = 30;               /* upper border */
hv->db = 30;               /* lower border */
x = 100;                   /* x position   */
y = 100;                   /* y position   */

/* Item selection Check */

hv->model = gtk_tree_view_get_model (GTK_TREE_VIEW(view));
hv->selection = gtk_tree_view_get_selection (GTK_TREE_VIEW(view));

if (gtk_tree_selection_get_selected (hv->selection,NULL,&(hv->iter)))
{
    gtk_tree_model_get (hv->model,&(hv->iter),0,&file_name,-1);
    gtk_tree_model_get (hv->model,&(hv->iter),8,&(hv->units),-1);

    if (hv->type == 0)
    {
        sprintf(file_name_tmp, "%s%s.hv", out_path, file_name);
        hv->saf  = file_name;
        hv->file = file_name_tmp;
    }
    else
    {
        sprintf(file_name_tmp, "%s%s.hv_sp", out_path, file_name);
        hv->saf  = file_name;
        hv->file = file_name_tmp;
    }

    /* Checking if the file has just been processed */

    if ((fp = fopen (hv->file,"r")) != NULL)
    {
        fclose (fp);

        /* Getting information from selected file */

        hv_data_read (hv);

        /* Window title bar */

        sprintf (title_bar,"%s  File: %s", hvsp_str, file_name);

        /*****************************************************************/
        /* Plot window                                                   */
        /*****************************************************************/
        {
            hv->window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
            gtk_window_move (GTK_WINDOW (hv->window), x, y);
            gtk_window_set_resizable (GTK_WINDOW (hv->window), FALSE);
            gtk_window_set_title (GTK_WINDOW(hv->window),title_bar);
        }

        {
            hv->vbox1 = gtk_vbox_new (FALSE, 0);
            gtk_widget_show (hv->vbox1);
            gtk_container_add (GTK_CONTAINER (hv->window), hv->vbox1);
```

```
}

{
    hv->toolbar1 = gtk_toolbar_new ();
    gtk_widget_show (hv->toolbar1);
    gtk_box_pack_start (GTK_BOX (hv->vbox1), hv->toolbar1,
                        FALSE, TRUE, 0);
    gtk_toolbar_set_style (GTK_TOOLBAR (hv->toolbar1),
                           GTK_TOOLBAR_ICONS);
}

{
    hv->button1 = (GtkWidget*)gtk_tool_button_new_from_stock
                 ("gtk-save");
    gtk_widget_show (hv->button1);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1), hv->button1);
}

{
    hv->toolitem1 = (GtkWidget*)gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem1);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem1);
    gtk_widget_set_size_request (hv->toolitem1, 10, -1);

    hv->vseparator1 = gtk_vseparator_new ();
    gtk_widget_show (hv->vseparator1);
    gtk_container_add (GTK_CONTAINER (hv->toolitem1),
                       hv->vseparator1);
}

{
    hv->toolitem2 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem2);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem2);

    hv->label1 = gtk_label_new ("<b> Plot start (Hz): </b>");
    gtk_widget_show (hv->label1);
    gtk_container_add (GTK_CONTAINER (hv->toolitem2),
                       hv->label1);
    gtk_label_set_use_markup (GTK_LABEL (hv->label1), TRUE);
}

{
    hv->toolitem3 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem3);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem3);

    hv->spinbutton1_adj = gtk_adjustment_new
                          (hv->data[0][0],
                           hv->data[0][0],
                           hv->data[0][hv->ndat - 1],
                           0.1, 10, 10);

    hv->spinbutton1 = gtk_spin_button_new
                      (GTK_ADJUSTMENT (hv->spinbutton1_adj),
                       1, 2);
    gtk_widget_show (hv->spinbutton1);
    gtk_container_add (GTK_CONTAINER (hv->toolitem3),
                       hv->spinbutton1);
}
```

```
{
    hv->toolitem4 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem4);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem4);
    gtk_widget_set_size_request (hv->toolitem4, 10, -1);

    hv->vseparator2 = gtk_vseparator_new ();
    gtk_widget_show (hv->vseparator2);
    gtk_container_add (GTK_CONTAINER (hv->toolitem4),
    hv->vseparator2);
}

{
    hv->toolitem5 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem5);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem5);

    hv->label2 = gtk_label_new ("<b> Plot end (Hz): </b>");
    gtk_widget_show (hv->label2);
    gtk_container_add (GTK_CONTAINER (hv->toolitem5),
                       hv->label2);
    gtk_label_set_use_markup (GTK_LABEL (hv->label2), TRUE);
}

{
    hv->toolitem6 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem6);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem6);

    hv->spinbutton2_adj = gtk_adjustment_new
                          (hv->data[0][hv->ndat - 1],
                           hv->data[0][0],
                           hv->data[0][hv->ndat - 1],
                           0.1, 10, 10);

    hv->spinbutton2 = gtk_spin_button_new
                      (GTK_ADJUSTMENT (hv->spinbutton2_adj),
                       1, 2);
    gtk_widget_show (hv->spinbutton2);
    gtk_container_add (GTK_CONTAINER (hv->toolitem6),
                       hv->spinbutton2);
}

{
    hv->toolitem7 = (GtkWidget*) gtk_tool_item_new ();
    gtk_widget_show (hv->toolitem7);
    gtk_container_add (GTK_CONTAINER (hv->toolbar1),
                       hv->toolitem7);
    gtk_widget_set_size_request (hv->toolitem7, 10, -1);

    hv->vseparator3 = gtk_vseparator_new ();
    gtk_widget_show (hv->vseparator3);
    gtk_container_add (GTK_CONTAINER (hv->toolitem7),
                       hv->vseparator3);
}

{
    hv->drawing_area = gtk_drawing_area_new ();
    gtk_drawing_area_size (GTK_DRAWING_AREA (hv->drawing_area),
```

```
                                hv->image_width,
                                hv->image_height);
            gtk_box_pack_start (GTK_BOX (hv->vbox1), hv->drawing_area,
                                FALSE, TRUE, 0);
        }

        /*****************************************************************/
        /* Setting signals and events                                    */
        /*****************************************************************/
        {
            g_signal_connect (GTK_OBJECT (hv->window),
                              "delete_event",
                              G_CALLBACK (hv_delete_event),
                              (gpointer) hv);
            g_signal_connect (G_OBJECT (hv->button1),"clicked",
                              G_CALLBACK (hv_save_image),
                              (gpointer) hv);
            g_signal_connect (GTK_OBJECT (hv->spinbutton1),
                              "value-changed",
                              G_CALLBACK (hv_set_start_event),
                             (gpointer) hv);
            g_signal_connect (GTK_OBJECT (hv->spinbutton2),
                              "value-changed",
                              G_CALLBACK (hv_set_stop_event),
                             (gpointer) hv);
            g_signal_connect (G_OBJECT (hv->drawing_area),
                              "configure_event",
                              G_CALLBACK (hv_configure_event),
                              (gpointer) hv);
            g_signal_connect (GTK_OBJECT (hv->drawing_area),
                              "expose_event",
                              G_CALLBACK (hv_expose_event),
                              (gpointer) hv);
            g_signal_connect (GTK_OBJECT (hv->drawing_area),
                              "button_press_event",
                              G_CALLBACK (hv_plot_press_event),
                              (gpointer) hv);
            g_signal_connect (GTK_OBJECT (hv->drawing_area),
                              "button_release_event",
                              G_CALLBACK (hv_plot_release_event),
                              (gpointer) hv);

            gtk_widget_set_events (hv->drawing_area,
                                     GDK_EXPOSURE_MASK
                                   | GDK_LEAVE_NOTIFY_MASK
                                   | GDK_BUTTON_PRESS_MASK
                                   | GDK_BUTTON_RELEASE_MASK
                                   | GDK_POINTER_MOTION_MASK
                                   | GDK_POINTER_MOTION_HINT_MASK);
        }

        gtk_widget_show_all (hv->window);
    }
    else
    /* Emit Warning: item not processed */
    {
        warning_no_item_selected ("<b>Item not already processed!</b>");
    }
}
else
/* Emit Warning: no item selected */
{
    warning_no_item_selected ("<b>No item selected!</b>");
```

```
    }
}

/*****************************************************************************/
/* HV_CONFIGURE_EVENT:                                                       */
/*****************************************************************************/

gboolean hv_configure_event (GtkWidget *widget,
                             GdkEventConfigure *event,
                             HV_STRUCT *hv)
{
    /*************************************************************************/
    /* Inizializing the pixmap                                               */
    /*************************************************************************/

    if (hv->pixmap)
    {
        g_object_unref (G_OBJECT (hv->pixmap));
    }

    hv->pixmap = gdk_pixmap_new (widget->window,hv->image_width,
                                 hv->image_height,-1);

    /*************************************************************************/
    /* HV plot                                                               */
    /*************************************************************************/

    hv->start = 0;
    hv->stop = (hv->ndat - 1);

    draw_hv_plot (widget, hv);

    return TRUE;
}

/*****************************************************************************/
/* HV_EXPOSE_EVENT:                                                          */
/*****************************************************************************/

gboolean hv_expose_event (GtkWidget *widget,
                          GdkEventExpose *event,
                          HV_STRUCT *hv)
{
    gdk_draw_drawable (widget->window,
                       widget->style->white_gc,
                       hv->pixmap,
                       event->area.x, event->area.y,
                       event->area.x, event->area.y,
                       event->area.width, event->area.height);

    return FALSE;
}

/*****************************************************************************/
/* HV_PLOT_PRESS_EVENT:                                                      */
/*****************************************************************************/

gint hv_plot_press_event (GtkWidget *widget,
                          GdkEventButton *event,
                          HV_STRUCT *hv)
{
    if ((event->button) == 1)
    {
```

```
        if ((event->x) > (hv->lb) &&
            (event->x) < (hv->image_width - hv->lb))
        {
            hv->tmp_sta =
            (((hv->stop - hv->start) * (event->x - hv->lb)) /
            (hv->image_width - hv->lb - hv->rb)) + hv->start;
        }
    }

    return TRUE;
}

/*****************************************************************************/
/* HV_PLOT_RELEASE_EVENT:                                                    */
/*****************************************************************************/

gint hv_plot_release_event (GtkWidget *widget,
                            GdkEventButton *event,
                            HV_STRUCT *hv)
{
    if ((event->button) == 1)
    {
        if ((event->x) > (hv->lb) &&
            (event->x) < (hv->image_width - hv->lb))
        {
            hv->tmp_sto =
            (((hv->stop - hv->start) * (event->x - hv->lb)) /
            (hv->image_width - hv->lb - hv->rb)) + hv->start + 1;

            if (hv->tmp_sto > hv->tmp_sta)
            {
                hv->stop = hv->tmp_sto;
                hv->start = hv->tmp_sta;
            }
            else
            {
                hv->start = hv->tmp_sto;
                hv->stop = hv->tmp_sta;
            }
        }
    }
    else
    {
        hv->start = 0;
        hv->stop = (hv->ndat - 1);
    }

    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv->spinbutton1),
                               ((gdouble)(hv->start) *
                                (gdouble)(hv->data[0][1] - hv->data[0][0])) +
                                hv->data[0][0]);

    gtk_spin_button_set_value (GTK_SPIN_BUTTON(hv->spinbutton2),
                               ((gdouble)(hv->stop + 1) *
                                (gdouble)(hv->data[0][1] - hv->data[0][0])) +
                                hv->data[0][0]);

    /*************************************************************************/
    /*  The following lines are optional, because redrawing is forced        */
    /*  by the previous "gtk_spin_button_set_value" calls.                   */
    /*************************************************************************/
    /*  draw_hv_plot (widget, hv);                                           */
    /*  gtk_widget_queue_draw_area(widget, 0, 0,                             */
```

```
    /*                              hv->image_width,                            */
    /*                              hv->image_height);                          */
    /*****************************************************************************/

    return TRUE;
}

/*****************************************************************************/
/* HV_SET_START_EVENT:                                                       */
/*****************************************************************************/

gint hv_set_start_event (GtkWidget *widget,
                         HV_STRUCT *hv)
{
    gdouble but_val = (gdouble) gtk_spin_button_get_value
                      (GTK_SPIN_BUTTON (hv->spinbutton1));

    gdouble tmp_start = (gdouble) (but_val - hv->data[0][0]) /
                        (gdouble) (hv->data[0][1] - hv->data[0][0]);

    if (tmp_start < hv->stop)
    {
        hv->start = (gint) tmp_start;

        draw_hv_plot (hv->drawing_area, hv);

        gtk_widget_queue_draw_area(hv->drawing_area, 0, 0,
                                   hv->image_width,
                                   hv->image_height);
    }

    return 0;
}

/*****************************************************************************/
/* HV_SET_STOP_EVENT:                                                        */
/*****************************************************************************/

gint hv_set_stop_event (GtkWidget *widget,
                        HV_STRUCT *hv)
{
    gdouble but_val = (gdouble) gtk_spin_button_get_value
                      (GTK_SPIN_BUTTON (hv->spinbutton2));

    gdouble tmp_stop = (gdouble) (but_val - hv->data[0][0]) /
                       (gdouble) (hv->data[0][1] - hv->data[0][0]);

    if (tmp_stop > hv->start)
    {
        hv->stop = (gint) tmp_stop;

        draw_hv_plot (hv->drawing_area, hv);

        gtk_widget_queue_draw_area(hv->drawing_area, 0, 0,
                                   hv->image_width,
                                   hv->image_height);
    }

    return 0;
}

/*****************************************************************************/
/* HV_DELETE_EVENT:                                                          */
```

```
/*****************************************************************************/

gboolean hv_delete_event (GtkWidget *widget,
                          GdkEvent  *event,
                          HV_STRUCT *hv)
{
    gint j;

    /* Signal disconnection */

    gtk_signal_disconnect_by_func (GTK_OBJECT (hv->spinbutton1),
                                   G_CALLBACK (hv_set_start_event),
                                   (gpointer) hv);

    gtk_signal_disconnect_by_func (GTK_OBJECT (hv->spinbutton2),
                                   G_CALLBACK (hv_set_stop_event),
                                   (gpointer) hv);

    /* Memory deallocation */

    for (j=0;j<6;j++)
    {
        if (hv->data[j] != NULL)
        {
            g_free (hv->data[j]);
        }
    }

    return FALSE;
}

/*****************************************************************************/
/* DRAW_HV_PLOT:                                                             */
/*****************************************************************************/

void draw_hv_plot (GtkWidget *widget,
                   HV_STRUCT *hv)
{
    gint w = hv->image_width;    /* Window width  */
    gint h = hv->image_height;   /* Window height */

    gint lb = hv->lb;        /* left border  */
    gint rb = hv->rb;        /* right border */
    gint ub = hv->ub;        /* upper border */
    gint db = hv->db;        /* lower border */

    gint tw = w - lb - rb;   /* Trace width  */
    gint th = h - ub - db;   /* Trace height */

    gint fosd_sta, fosd_sto;
    gint x1, y1, x2, y2;

    gdouble p1, p2;

    gdouble max = hv->data_max[hv->ch];

    guchar rgbbuf [w * h * 3];

    gint blue   [3] = {100, 100, 255};
    gint green  [3] = {100, 255, 100};
    gint red    [3] = {255, 0, 0};
    gint yellow [3] = {255, 200, 0};
    gint black  [3] = {  0,   0,   0};
```

```
gint white  [3] = {255, 255, 255};

gint nf = 15;
gint na = 10;

gdouble df, da;

gint sta = hv->start;
gint sto = hv->stop;
gint i;

/*************************************************************************/
/* Setting the background color of the image                             */
/*************************************************************************/

rgb_buffer_set_bg_color (rgbbuf,w,h,white);

/*************************************************************************/
/* F0 and F0 standard deviation                                          */
/*************************************************************************/

if ( hv->fo != -9 )
{
    /* F0 standard deviation */

    fosd_sta = (lb + ((gdouble)((hv->fosd_sta - hv->data[0][sta]) * tw ) /
                      (gdouble)(hv->data[0][sto] - hv->data[0][sta])));

    fosd_sto = (lb + ((gdouble)((hv->fosd_sto - hv->data[0][sta]) * tw ) /
                      (gdouble)(hv->data[0][sto] - hv->data[0][sta])));

    if (plotsdfo != 0)
    {
        for (x1 = fosd_sta; x1 < fosd_sto; x1++)
        {
            x2 = x1;
            y1 = ub;
            y2 = ub + th;

            if ((x1 > lb) && (x1 < (lb + tw)))
            {
                rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,yellow);
            }
        }
    }

    /* F0 */
    {
        x1 = lb + ((gdouble)((hv->fo - hv->data[0][sta]) * tw ) /
                   (gdouble)(hv->data[0][sto] - hv->data[0][sta]));
        x2 = x1;
        y1 = ub;
        y2 = ub + th;

        if (plotfo != 0)
        {
            if ((x1 > lb) && (x1 < (lb + tw)))
            {
                rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,red);
            }
        }
    }
}
```

```
/*************************************************************************/
/* Spectral ratio - Absolute spectrum plot                               */
/*************************************************************************/

/* HV Spectral ratio - Absolute spectrum */

if ((hv->data[hv->ch + 1] != NULL))
{
    for (i = sta; i < sto; i++)
    {
        p1 = hv->data[(hv->ch + 1)][i];
        p2 = hv->data[(hv->ch + 1)][i+1];

        x1 = (((i - sta) * tw) / (sto - sta)) + lb;
        x2 = ((((i + 1) - sta) * tw) / (sto - sta)) + lb;

        y1 = th + ub - ((p1 * th) / max);
        y2 = th + ub - ((p2 * th) / max);

        rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,blue);
    }
}

if (plotsdhv != 0)
{
    /* + Standard deviation */

    if ((hv->data[hv->ch + 1] != NULL))
    {
        for (i = sta; i < sto; i++)
        {
            /* Spectral ratio */
            if (hv->type == 0)
            {
                p1 = hv->data[hv->ch + 1][i] * hv->data[hv->ch + 4][i];
                p2 = hv->data[hv->ch + 1][i+1] * hv->data[hv->ch + 4][i+1];
            }
            /* Absolute spectrum */
            else
            {
                p1 = hv->data[hv->ch + 1][i] + hv->data[hv->ch + 4][i];
                p2 = hv->data[hv->ch + 1][i+1] + hv->data[hv->ch + 4][i+1];
            }

            x1 = (((i - sta) * tw) / (sto - sta)) + lb;
            x2 = ((((i + 1) - sta) * tw) / (sto - sta)) + lb;

            y1 = th + ub - ((p1 * th) / max);
            y2 = th + ub - ((p2 * th) / max);

            rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,green);
        }
    }

    /* - Standard deviation */

    if ((hv->data[hv->ch] != NULL))
    {
        for (i = sta; i < sto; i++)
        {
            /* Spectral ratio */
            if (hv->type == 0)
```

```
            {
                p1 = hv->data[hv->ch + 1][i] / hv->data[hv->ch + 4][i];
                p2 = hv->data[hv->ch + 1][i+1] / hv->data[hv->ch + 4][i+1];
            }
            /* Absolute spectrum */
            else
            {
                p1 = hv->data[hv->ch + 1][i] - hv->data[hv->ch + 4][i];
                p2 = hv->data[hv->ch + 1][i+1] - hv->data[hv->ch + 4][i+1];
            }

            x1 = (((i - sta) * tw) / (sto - sta)) + lb;
            x2 = ((((i + 1) - sta) * tw) / (sto - sta)) + lb;

            y1 = th + ub - ((p1 * th) / max);
            y2 = th + ub - ((p2 * th) / max);

            rgb_buffer_draw_line (rgbbuf,w,h,x1,y1,x2,y2,green);
        }
    }
}

/*************************************************************************/
/* Traces frame                                                          */
/*************************************************************************/

/* BOX */;
{
    rgb_buffer_draw_line (rgbbuf,w,h,lb,(ub+th),(lb+tw),(ub+th),black);
    rgb_buffer_draw_line (rgbbuf,w,h,lb,ub,lb,(ub+th),black);
}

/* TIME MARK */

for(i = 0; i <= nf; i++)
{
    df = ((i * tw) / nf);

    rgb_buffer_draw_line
    (rgbbuf,w,h,(lb+df),(ub+th),(lb+df),(ub+th-5),black);
}

/* AMPLITUDE MARK */

for(i = 0; i <= na; i++)
{
    da = ((i * th) / na);

    rgb_buffer_draw_line
    (rgbbuf,w,h,lb,(ub+da),(lb+5),(ub+da),black);
}

/*************************************************************************/
/* Copying rgm to pixmap                                                 */
/*************************************************************************/

if (hv->pixmap)
{
    gdk_draw_rgb_image (hv->pixmap,
                        widget->style->white_gc,
                        0, 0,
                        hv->image_width, hv->image_height,
                        GDK_RGB_DITHER_MAX,
```

```
                        rgbbuf,
                        hv->image_width * 3);
}

/*****************************************************************************/
/* Label                                                                     */
/*****************************************************************************/

gchar header[200];
gchar string[200];

/* HEADER */

{
    sprintf (header, "TYPE: %s      ", hv->hvsp_type);

    sprintf (string, "FILE:  %s      ", hv->saf);
    strcat (header, string);

    sprintf (string, "FREQUENCY:  Hz      ");
    strcat (header, string);

    if ( hv->type != 0)
    {
        sprintf (string, "AMPLITUDE:  %s      ", hv->units);
        strcat (header, string);
    }

    if ( hv->fo != -9)
    {
        sprintf (string, "COMPUTED F(0):  %.2lf  Hz", hv->fo);
        strcat (header, string);
    }

    draw_string (widget,hv->pixmap,header,lb,15,black);
}

/* FREQUENCY MARK LABEL */

gdouble f_label;

for(i = 0; i <= nf; i++)
{
    df = ((i * tw) / nf);

    f_label = ((gdouble) ((hv->data[0][sto] - hv->data[0][sta]) * df) /
              ((gdouble) tw)) + hv->data[0][sta];

    sprintf (string, "%.2f", f_label);
    draw_string (widget,hv->pixmap,string,(lb+df),(ub+th+15),black);
}

/* AMPLITUDE MARK LABEL */

gdouble a_label;

for(i = 0; i <= na; i++)
{
    da = ((i * th) / na);

    a_label = (max - ((gdouble) (max * da) / ((gdouble) th)));

    if (hv->type == 0)
```

```
        {
            sprintf (string, "%.1f", a_label);
        }
        else
        {
            sprintf (string, "%.2e", a_label);
        }

        draw_string (widget,hv->pixmap,string,10,(ub+da),black);
    }

}

/*****************************************************************************/
/* DATA_READ:                                                                */
/*****************************************************************************/

HV_STRUCT *hv_data_read (HV_STRUCT *hv)
{
    FILE *fp;
    gint i,j;
    gchar string[500];

    if ((fp = fopen (hv->file,"r")) != NULL)
    {
        while(strstr (string,"#----------") == NULL)
        {
            fgets (string,500,fp);
        }

        /*********************************************************************/
        /*  Number of windows                                                */
        /*  Prototype: " # n_windows:  112 "                                 */
        /*********************************************************************/

        fgets (string,500,fp);

        sscanf(string,"%s%s%d",string,string,&(hv->nwindows));

        /*********************************************************************/
        /*  Windows length                                                   */
        /*  Prototype: " # window_len:  30.000 "                             */
        /*********************************************************************/

        fgets (string,500,fp);

        sscanf(string,"%s%s%lf",string,string,&(hv->winlen));

        /*********************************************************************/
        /*  Labels                                                           */
        /*  Prototype: " # labels:  Volt  U-D  N-S  E-O "                    */
        /*********************************************************************/

        fgets (string,500,fp);

        /*********************************************************************/
        /*  Number of freqeuncy samples                                      */
        /*  Prototype: " # n_freq_samples:  1906 ...                         */
        /*  Fo:  4.0232 range +/- 1 ...                                      */
        /*  s.d.:  3.6465  -  4.4389 [Hz] "                                  */
        /*********************************************************************/

        fgets (string,500,fp);
```

```
hv->fo = -9;

sscanf(string,"%s%s%d       \
               %s%lf%s%s%lf  \
               %s%lf%s%lf%s",
               string,string,&(hv->ndat),
               string,&(hv->fo),string,string,&(hv->forange),
               string,&(hv->fosd_sta),string,&(hv->fosd_sto),string);

/*************************************************************************/
/*  Labels                                                               */
/*  Prototype: " # freq  av_HV  ns_HV  ew_HV ...                         */
/*  av_HV_logstd  ns_HV_logstd  ew_HV_logstd"                            */
/*************************************************************************/

fgets (string,500,fp);

/*************************************************************************/
/*  Data                                                                 */
/*************************************************************************/

/* initialization */

for (j = 0; j < 3; j++)
{
    hv->data_max[j] = 0;
}

for (j = 0; j < 7; j++)
{
    hv->data[j] = (gdouble *) g_try_malloc
                  (sizeof(gdouble)*(hv->ndat));
}

/* Reading data */

if ((hv->data[0] != NULL) &&
    (hv->data[1] != NULL) &&
    (hv->data[2] != NULL) &&
    (hv->data[3] != NULL) &&
    (hv->data[4] != NULL) &&
    (hv->data[5] != NULL) &&
    (hv->data[6] != NULL))
{
    for (i = 0; i < (hv->ndat); i++)
    {
        if (!feof (fp))
        {
            for (j = 0 ; j < 7; j++)
            {
                fscanf(fp,"%lf",&(hv->data[j][i]));
            }
        }

        /* Max value (whitin standard deviation) */

        for (j = 0; j < 3; j++)
        {
            /* for spectral ratio */
            if (hv->type == 0)
            {
                if (((hv->data[j+1][i]) * (hv->data[j+4][i])) >
```

```
                            (hv->data_max[j]))
                        {
                            hv->data_max[j] = ((hv->data[j+1][i]) *
                                               (hv->data[j+4][i]));
                        }
                    }

                    /* for absolute spectra */
                    else
                    {
                        if (((hv->data[j+1][i]) + (hv->data[j+4][i])) >
                            (hv->data_max[j]))
                        {
                            hv->data_max[j] = ((hv->data[j+1][i]) +
                                               (hv->data[j+4][i]));
                        }
                    }
                }
            }
        }

        fclose (fp);
    }

    return hv;
}

/*****************************************************************************/
/* HV_SAVE_IMAGE:                                                            */
/*****************************************************************************/

void hv_save_image (GtkWidget *widget,
                    HV_STRUCT *hv)
{
    gchar img_name[500];

    if (hv->type == 0)
    {
        if (hv->ch == 0)
        {
            sprintf(img_name, "%sAverageHV.%s.%s", img_path, hv->saf, imgext);
        }
        if (hv->ch == 1)
        {
            sprintf(img_name, "%sH(1)V.%s.%s", img_path, hv->saf, imgext);
        }
        if (hv->ch == 2)
        {
            sprintf(img_name, "%sH(2)V.%s.%s", img_path, hv->saf, imgext);
        }
    }
    else
    {
        if (hv->ch == 0)
        {
            sprintf(img_name, "%sCh0Sp.%s.%s", img_path, hv->saf, imgext);
        }
        if (hv->ch == 1)
        {
            sprintf(img_name, "%sCh1Sp.%s.%s", img_path, hv->saf, imgext);
        }
        if (hv->ch == 2)
        {
```

```
            sprintf(img_name, "%sCh2Sp.%s.%s", img_path, hv->saf, imgext);
        }
    }


    /* File selection dialog window */
    hv->img_selection = gtk_file_selection_new ("Save Image");

    /* Set the default filename */
    gtk_file_selection_set_filename (GTK_FILE_SELECTION(hv->img_selection),
                                     img_name);

    /* OK button signal connection */
    g_signal_connect (GTK_FILE_SELECTION(hv->img_selection)->ok_button,
                      "clicked",
                      G_CALLBACK(hv_overwrite_image),
                      (gpointer)hv);

    /* Cancel button signal connection */
    g_signal_connect_swapped
                     (GTK_FILE_SELECTION(hv->img_selection)->cancel_button,
                      "clicked",
                      G_CALLBACK(gtk_widget_destroy),
                      hv->img_selection);

    gtk_widget_show (hv->img_selection);
}

/*****************************************************************************/
/* HV_OVERWRITE_IMAGE:                                                       */
/*****************************************************************************/

void hv_overwrite_image (GtkWidget *widget,
                         HV_STRUCT *hv)
{
    const gchar *selected_filename;
    gchar img_filename[200];
    GtkWidget *overwrite;
    GtkWidget *hbox;
    GtkWidget *stock_image;
    GtkWidget *label;
    gint response;
    FILE *fp;

    selected_filename = gtk_file_selection_get_filename
                        (GTK_FILE_SELECTION(hv->img_selection));

    strcpy (img_filename, selected_filename);

    /* File exists */
    if ((fp = fopen(selected_filename, "r")) != NULL)
    {
        /* Dialog window */
        overwrite = gtk_dialog_new_with_buttons
                    ("Warning!",
                     NULL,
                     GTK_DIALOG_MODAL |
                     GTK_DIALOG_DESTROY_WITH_PARENT,
                     GTK_STOCK_APPLY,
                     GTK_RESPONSE_ACCEPT,
                     GTK_STOCK_CANCEL,
                     GTK_RESPONSE_REJECT,
                     NULL);
```

```
        /* Horizontal box */
        {
            hbox = gtk_hbox_new (FALSE,10);
            gtk_container_set_border_width (GTK_CONTAINER(hbox),10);
            gtk_box_pack_start (GTK_BOX(GTK_DIALOG(overwrite)->vbox),
                                hbox,FALSE,FALSE,0);
        }

        /* Dialog question stock image and label */
        {
            stock_image = gtk_image_new_from_stock (GTK_STOCK_DIALOG_QUESTION,
                                                    GTK_ICON_SIZE_DIALOG);
            gtk_box_pack_start (GTK_BOX(hbox),stock_image,FALSE,FALSE,0);

            label = gtk_label_new_with_mnemonic ("File exists: overwrite?");
            gtk_box_pack_start (GTK_BOX(hbox),label,FALSE,FALSE,0);
        }

        gtk_widget_show_all (overwrite);

        /* Response evaluation */
        {
        response = gtk_dialog_run (GTK_DIALOG(overwrite));

            if (response == GTK_RESPONSE_ACCEPT)
            {
                if (hv->pixmap)
                {
                    gdk_pixmap_save (hv->pixmap, img_filename);
                }
                gtk_widget_destroy (GTK_WIDGET(hv->img_selection));
            }
        }

        gtk_widget_destroy (overwrite);
        fclose (fp);
    }

    /* File doesn't exist */
    else
    {
        if (hv->pixmap)
        {
            gdk_pixmap_save (hv->pixmap, img_filename);
        }
        gtk_widget_destroy (GTK_WIDGET(hv->img_selection));
    }
}
```

1.30 - Plothv.h

```
#ifndef PLOTHV_H
#define PLOTHV_H

/*****************************************************************************/
/* Variables.                                                                */
/*****************************************************************************/

typedef struct HV_STRUCT
{
```

```
    GtkWidget           *window;
    GdkPixmap           *pixmap;
    GtkWidget           *vbox1;
    GtkWidget           *toolbar1;
    GtkWidget           *button1;
    GtkWidget           *toolitem1;
    GtkWidget           *vseparator1;
    GtkWidget           *toolitem2;
    GtkWidget           *label1;
    GtkWidget           *toolitem3;
    GtkObject           *spinbutton1_adj;
    GtkWidget           *spinbutton1;
    GtkWidget           *toolitem4;
    GtkWidget           *vseparator2;
    GtkWidget           *toolitem5;
    GtkWidget           *label2;
    GtkWidget           *toolitem6;
    GtkObject           *spinbutton2_adj;
    GtkWidget           *spinbutton2;
    GtkWidget           *toolitem7;
    GtkWidget           *vseparator3;
    GtkWidget           *drawing_area;
    GtkWidget           *img_selection;
    GtkTreeSelection    *selection;
    GtkTreeModel        *model;
    GtkTreeIter          iter;
    gint                 nwindows;
    gdouble              winlen;
    gint                 ndat;
    gint                 start;
    gint                 tmp_sta;
    gint                 stop;
    gint                 tmp_sto;
    gdouble              fo;
    gdouble              forange;
    gdouble              fosd_sta;
    gdouble              fosd_sto;
    gdouble             *data[7];
    gdouble              data_max[3];
    gchar               *hvsp_type;
    guint                ch;
    guint                type;
    gchar               *file;
    gchar               *units;
    gchar               *saf;
    gint                 image_width;
    gint                 image_height;
    gint                 lb;
    gint                 rb;
    gint                 ub;
    gint                 db;
    SAF_LIST_PTR         ptr_list;
} HV_STRUCT;

/*************************************************************************/
/* Function prototypes.                                                  */
/*************************************************************************/

void            plot_hvsp               (GtkWidget *widget,
                                         gchar *channel_str);

gboolean        hv_configure_event      (GtkWidget *widget,
                                         GdkEventConfigure *event,
```

```
                                        HV_STRUCT *hv);

gboolean        hv_expose_event         (GtkWidget *widget,
                                         GdkEventExpose *event,
                                         HV_STRUCT *hv);

gint            hv_plot_press_event     (GtkWidget *widget,
                                         GdkEventButton *event,
                                         HV_STRUCT *hv);

gint            hv_plot_release_event   (GtkWidget *widget,
                                         GdkEventButton *event,
                                         HV_STRUCT *hv);

gint            hv_set_start_event      (GtkWidget *widget,
                                         HV_STRUCT *hv);

gint            hv_set_stop_event       (GtkWidget *widget,
                                         HV_STRUCT *hv);

gboolean        hv_delete_event         (GtkWidget *widget,
                                         GdkEvent  *event,
                                         HV_STRUCT *hv);

void            draw_hv_plot            (GtkWidget *widget,
                                         HV_STRUCT *hv);

HV_STRUCT      *hv_data_read            (HV_STRUCT *hv);


void            hv_save_image           (GtkWidget *widget,
                                         HV_STRUCT *hv);

void            hv_overwrite_image      (GtkWidget *widget,
                                         HV_STRUCT *hv);

# endif
```

1.31 - Warning.c

```
# include <gtk/gtk.h>

# include "warning.h"

/*****************************************************************************/
/* WARNING_UNSAVED_WORK:                                                     */
/* This dialog warn the user for any unsaved change before quitting          */
/*****************************************************************************/

void warning_unsaved_work (void)
{
    GtkWidget *warn_dialog;
    GtkWidget *warn_dialog_vbox;
    GtkWidget *warn_hbox;
    GtkWidget *warn_image;
    GtkWidget *warn_label;
    GtkWidget *warn_dialog_action_area;
    GtkWidget *warn_button1;
    GtkWidget *warn_button2;

    {
```

```
        warn_dialog = gtk_dialog_new ();
        gtk_window_set_title (GTK_WINDOW (warn_dialog), "GSesame Warning!");
        gtk_window_set_position (GTK_WINDOW (warn_dialog), GTK_WIN_POS_CENTER);
        gtk_window_set_resizable (GTK_WINDOW (warn_dialog), FALSE);
        gtk_window_set_type_hint (GTK_WINDOW (warn_dialog),
                                  GDK_WINDOW_TYPE_HINT_DIALOG);
    }

    {
        warn_dialog_vbox = GTK_DIALOG (warn_dialog)->vbox;
        gtk_widget_show (warn_dialog_vbox);

        warn_hbox = gtk_hbox_new (FALSE, 0);
        gtk_widget_show (warn_hbox);
        gtk_box_pack_start (GTK_BOX (warn_dialog_vbox),
                            warn_hbox, TRUE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (warn_hbox), 10);
    }

    {
        warn_image = gtk_image_new_from_stock ("gtk-dialog-warning",
                                               GTK_ICON_SIZE_DIALOG);
        gtk_widget_show (warn_image);
        gtk_box_pack_start (GTK_BOX (warn_hbox), warn_image, TRUE, TRUE, 0);

        warn_label = gtk_label_new
        ("<b>Warning!\nAll unsaved work will be lost.\nContinue anyway?</b>");
        gtk_widget_show (warn_label);
        gtk_box_pack_start (GTK_BOX (warn_hbox), warn_label, FALSE, FALSE, 10);
        gtk_label_set_use_markup (GTK_LABEL (warn_label), TRUE);
        gtk_label_set_justify (GTK_LABEL (warn_label), GTK_JUSTIFY_CENTER);
    }

    {
        warn_dialog_action_area = GTK_DIALOG (warn_dialog)->action_area;
        gtk_widget_show (warn_dialog_action_area);
        gtk_button_box_set_layout (GTK_BUTTON_BOX (warn_dialog_action_area),
                                   GTK_BUTTONBOX_SPREAD);

        warn_button1 = gtk_button_new_from_stock ("gtk-undo");
        g_signal_connect (GTK_OBJECT (warn_button1), "clicked",
                          G_CALLBACK (dialog_destroy),
                         (gpointer) warn_dialog);
        gtk_widget_show (warn_button1);
        gtk_dialog_add_action_widget (GTK_DIALOG (warn_dialog),
                                      warn_button1, GTK_RESPONSE_OK);
        GTK_WIDGET_SET_FLAGS (warn_button1, GTK_CAN_DEFAULT);

        warn_button2 = gtk_button_new_from_stock ("gtk-quit");
        g_signal_connect (GTK_OBJECT (warn_button2), "clicked",
                          G_CALLBACK (gtk_main_quit), NULL);
        gtk_widget_show (warn_button2);
        gtk_dialog_add_action_widget (GTK_DIALOG (warn_dialog),
                                      warn_button2, 0);
        GTK_WIDGET_SET_FLAGS (warn_button2, GTK_CAN_DEFAULT);
    }

    gtk_widget_show_all (warn_dialog);
}

/*****************************************************************************/
/* WARNING_NO_ITEM:                                                          */
/* This dialog warn the user that no item was selected.                      */
```

```
/*****************************************************************************/

void warning_no_item_selected (gchar *string)
{
    GtkWidget *warn_dialog;
    GtkWidget *warn_dialog_vbox;
    GtkWidget *warn_hbox;
    GtkWidget *warn_image;
    GtkWidget *warn_label;
    GtkWidget *warn_dialog_action_area;
    GtkWidget *warn_okbutton;

    {
        warn_dialog = gtk_dialog_new ();
        gtk_window_set_title (GTK_WINDOW (warn_dialog), "GSesame Warning!");
        gtk_window_set_position (GTK_WINDOW (warn_dialog), GTK_WIN_POS_CENTER);
        gtk_window_set_resizable (GTK_WINDOW (warn_dialog), FALSE);
        gtk_window_set_type_hint (GTK_WINDOW (warn_dialog),
                                  GDK_WINDOW_TYPE_HINT_DIALOG);
    }

    {
        warn_dialog_vbox = GTK_DIALOG (warn_dialog)->vbox;
        gtk_widget_show (warn_dialog_vbox);

        warn_hbox = gtk_hbox_new (FALSE, 0);
        gtk_widget_show (warn_hbox);
        gtk_box_pack_start (GTK_BOX (warn_dialog_vbox),
                            warn_hbox, TRUE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (warn_hbox), 10);

        warn_image = gtk_image_new_from_stock ("gtk-dialog-warning",
                                               GTK_ICON_SIZE_DIALOG);
        gtk_widget_show (warn_image);
        gtk_box_pack_start (GTK_BOX (warn_hbox), warn_image, TRUE, TRUE, 0);

        warn_label = gtk_label_new (string);
        gtk_widget_show (warn_label);
        gtk_box_pack_start (GTK_BOX (warn_hbox), warn_label, FALSE, FALSE, 10);
        gtk_label_set_use_markup (GTK_LABEL (warn_label), TRUE);
    }

    {
        warn_dialog_action_area = GTK_DIALOG (warn_dialog)->action_area;
        gtk_widget_show (warn_dialog_action_area);

        warn_okbutton = gtk_button_new_from_stock ("gtk-ok");
        g_signal_connect (GTK_OBJECT (warn_okbutton), "clicked",
                          G_CALLBACK (dialog_destroy),
                          (gpointer) warn_dialog);
        gtk_widget_show (warn_okbutton);
        gtk_dialog_add_action_widget (GTK_DIALOG (warn_dialog),
                                      warn_okbutton, GTK_RESPONSE_OK);
        GTK_WIDGET_SET_FLAGS (warn_okbutton, GTK_CAN_DEFAULT);
    }

    gtk_widget_show_all (warn_dialog);
}

/*****************************************************************************/
/* DIALOG_DESTROY:                                                           */
/* This callback destroy is used to destroy the warning dialog widget.       */
/*****************************************************************************/
```

```
gboolean dialog_destroy (GtkWidget *widget, GtkWidget *dialog)
{
    gtk_widget_destroy (dialog);
    return TRUE;
}
```

1.32 - Warning.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef WARNING_H
#define WARNING_H

    void        warning_unsaved_work        (void);

    void        warning_no_item_selected    (gchar *string);

    gboolean    dialog_destroy              (GtkWidget *widget,
                                             GtkWidget *dialog);

# endif
```

1.33 - About.c

```
# include <gtk/gtk.h>

# include "about.h"
# include "thanks.h"
# include "icons/logo.xpm"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

GtkWidget *about_window;
GtkWidget *about_vbox;
GtkWidget *about_hbox1;
GtkWidget *about_hbox2;
GtkWidget *about_image1;
GtkWidget *about_image2;
GtkWidget *about_label1;
GtkWidget *about_label2;
GtkWidget *about_label3;
GtkWidget *about_hbuttonbox;
GtkWidget *about_button1;
GtkWidget *about_button2;
GtkWidget *about_button_img1;
GtkWidget *about_button_img2;
GtkWidget *about_button_label1;
GtkWidget *about_button_label2;
GtkWidget *about_button_align1;
GtkWidget *about_button_align2;

/*****************************************************************************/
/* ABOUT_WINDOW_SHOWHIDE:                                                    */
```

```
/*****************************************************************************/

void about_window_showhide (void)
{
    if (GTK_WIDGET_VISIBLE(about_window))
    {
        gtk_widget_hide (about_window);
    }
    else
    {
        gtk_widget_show (about_window);
    }
}

/*****************************************************************************/
/* CREATE_ABOUT_WINDOW:                                                      */
/*****************************************************************************/

void create_about_window (void)
{
    GdkPixmap *pixmap;
    GdkBitmap *mask;
    GtkStyle *style;

    /* About window */
    {
        about_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_container_set_border_width (GTK_CONTAINER (about_window), 10);
        gtk_window_set_title (GTK_WINDOW (about_window), "About GSesame");
        gtk_window_set_position (GTK_WINDOW (about_window),GTK_WIN_POS_CENTER);
        gtk_window_set_resizable (GTK_WINDOW (about_window), FALSE);
        g_signal_connect (GTK_OBJECT (about_window), "delete_event",
                          G_CALLBACK (about_window_showhide), NULL);

        gtk_widget_show (about_window);
        gtk_widget_hide (about_window);

        about_vbox = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (about_vbox);
        gtk_container_add (GTK_CONTAINER (about_window), about_vbox);
    }

    /* Gsesame icon image */
    {
        style = gtk_widget_get_style(about_window);
        pixmap = gdk_pixmap_create_from_xpm_d (about_window->window, &mask,
                                               &style->bg[GTK_STATE_NORMAL],
                                               (gchar **) logo_xpm);
        about_image1 = gtk_image_new_from_pixmap (pixmap, mask);
        gtk_widget_show (about_image1);
        gtk_box_pack_start (GTK_BOX (about_vbox), about_image1,
                            TRUE, TRUE, 10);
    }

    /* Labels */
    {
        about_label1 = gtk_label_new
        ("<b><big><big><big>GSesame 1.0</big></big></big></b>");
        gtk_widget_show (about_label1);
        gtk_box_pack_start (GTK_BOX (about_vbox), about_label1,
                            TRUE, TRUE, 5);
        gtk_label_set_use_markup (GTK_LABEL (about_label1), TRUE);
        gtk_label_set_justify (GTK_LABEL (about_label1),
```

```
                            GTK_JUSTIFY_CENTER);

    about_label2 = gtk_label_new
    ("A graphical frontend for SESAME\nmodules written in GTK+");
    gtk_widget_show (about_label2);
    gtk_box_pack_start (GTK_BOX (about_vbox), about_label2,
                        TRUE, TRUE, 0);
    gtk_label_set_justify (GTK_LABEL (about_label2),
                           GTK_JUSTIFY_CENTER);

    about_label3 = gtk_label_new
    ("\302\251 2006 Poggi Valerio");
    gtk_widget_show (about_label3);
    gtk_box_pack_start (GTK_BOX (about_vbox), about_label3,
                        FALSE, FALSE,5);
    gtk_label_set_justify (GTK_LABEL (about_label3),
                           GTK_JUSTIFY_CENTER);
}

/* Button box and buttons */
{
    about_hbuttonbox = gtk_hbutton_box_new ();
    gtk_widget_show (about_hbuttonbox);
    gtk_box_pack_start (GTK_BOX (about_vbox), about_hbuttonbox,
                        TRUE, TRUE, 0);
    gtk_button_box_set_layout (GTK_BUTTON_BOX (about_hbuttonbox),
                               GTK_BUTTONBOX_EDGE);

    about_button1 = gtk_button_new ();
    g_signal_connect (G_OBJECT (about_button1), "clicked",
                      G_CALLBACK (thanks_window_showhide), NULL);
    gtk_widget_show (about_button1);
    gtk_container_add (GTK_CONTAINER (about_hbuttonbox), about_button1);
    GTK_WIDGET_SET_FLAGS (about_button1, GTK_CAN_DEFAULT);

    {
        about_button_align1 = gtk_alignment_new (0.5, 0.5, 0, 0);
        gtk_widget_show (about_button_align1);
        gtk_container_add (GTK_CONTAINER (about_button1),
                           about_button_align1);

        about_hbox1 = gtk_hbox_new (FALSE, 2);
        gtk_widget_show (about_hbox1);
        gtk_container_add (GTK_CONTAINER (about_button_align1),
                           about_hbox1);

        about_button_img1 = gtk_image_new_from_stock
                          ("gtk-about", GTK_ICON_SIZE_BUTTON);
        gtk_widget_show (about_button_img1);
        gtk_box_pack_start (GTK_BOX (about_hbox1), about_button_img1,
                            FALSE, FALSE, 0);

        about_button_label1 = gtk_label_new_with_mnemonic ("_Thanks");
        gtk_widget_show (about_button_label1);
        gtk_box_pack_start (GTK_BOX (about_hbox1), about_button_label1,
                            FALSE, FALSE, 0);
    }

    about_button2 = gtk_button_new ();
    g_signal_connect (G_OBJECT (about_button2), "clicked",
                      G_CALLBACK (about_window_showhide), NULL);
    gtk_widget_show (about_button2);
    gtk_container_add (GTK_CONTAINER (about_hbuttonbox), about_button2);
```

```
        GTK_WIDGET_SET_FLAGS (about_button2, GTK_CAN_DEFAULT);

        {
            about_button_align2 = gtk_alignment_new (0.5, 0.5, 0, 0);
            gtk_widget_show (about_button_align2);
            gtk_container_add (GTK_CONTAINER (about_button2),
                               about_button_align2);

            about_hbox2 = gtk_hbox_new (FALSE, 2);
            gtk_widget_show (about_hbox2);
            gtk_container_add (GTK_CONTAINER (about_button_align2),
                               about_hbox2);

            about_button_img2 = gtk_image_new_from_stock
                          ("gtk-close", GTK_ICON_SIZE_BUTTON);
            gtk_widget_show (about_button_img2);
            gtk_box_pack_start (GTK_BOX (about_hbox2), about_button_img2,
                                FALSE, FALSE, 0);

            about_button_label2 = gtk_label_new_with_mnemonic ("_Close");
            gtk_widget_show (about_button_label2);
            gtk_box_pack_start (GTK_BOX (about_hbox2), about_button_label2,
                                FALSE, FALSE, 0);
        }
    }
}
```

1.34 - About.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef ABOUT_H
#define ABOUT_H

    void about_window_showhide  (void);

    void create_about_window    (void);

# endif
```

1.35 - Thanks.c

```
# include <gtk/gtk.h>

# include "thanks.h"

/*****************************************************************************/
/* Local variables                                                           */
/*****************************************************************************/

GtkWidget *thanks_window;
GtkWidget *thanks_vbox;
GtkWidget *thanks_notebook;
GtkWidget *thanks_textview1;
GtkWidget *thanks_label1;
GtkWidget *thanks_textview2;
GtkWidget *thanks_label2;
```

```
GtkWidget *thanks_hbuttonbox;
GtkWidget *thanks_button;
GtkWidget *thanks_button_align;
GtkWidget *thanks_hbox;
GtkWidget *thanks_button_img;
GtkWidget *thanks_button_label;

/*****************************************************************************/
/* THANKS_WINDOW_SHOWHIDE:                                                   */
/*****************************************************************************/

void thanks_window_showhide (void)
{
    if (GTK_WIDGET_VISIBLE(thanks_window))
    {
         gtk_widget_hide (thanks_window);
    }
    else
    {
        gtk_widget_show (thanks_window);
    }
}

/*****************************************************************************/
/* THANKS_ABOUT_WINDOW:                                                      */
/*****************************************************************************/

void create_thanks_window (void)
{
gchar *string1 =
"\nPoggi Valerio <valerio.poggi@idpa.cnr.it>";

gchar *string2 =
"\nProf. Alberto Marcellini (IDPA CNR) \
\nProf. Alberto Tento (IDPA CNR) \
\nThe SESAME Team \
\nThe GTK Team";

    /* Thanks window */
    {
        thanks_window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
        gtk_widget_set_size_request (thanks_window, 350, 250);
        gtk_container_set_border_width (GTK_CONTAINER (thanks_window), 10);
        gtk_window_set_title (GTK_WINDOW (thanks_window), "Thanks");
        gtk_window_set_position (GTK_WINDOW (thanks_window),GTK_WIN_POS_CENTER);
        gtk_window_set_resizable (GTK_WINDOW (thanks_window), FALSE);
        g_signal_connect (GTK_OBJECT (thanks_window), "delete_event",
                          G_CALLBACK (thanks_window_showhide), NULL);

        gtk_widget_show (thanks_window);
        gtk_widget_hide (thanks_window);

        thanks_vbox = gtk_vbox_new (FALSE, 0);
        gtk_widget_show (thanks_vbox);
        gtk_container_add (GTK_CONTAINER (thanks_window), thanks_vbox);
    }

    /* The notebook */
    {
        thanks_notebook = gtk_notebook_new ();
        gtk_widget_show (thanks_notebook);
        gtk_box_pack_start (GTK_BOX (thanks_vbox), thanks_notebook,
                            TRUE, TRUE, 5);
```

```
    }

    /* "Written by" page */
    {
        thanks_textview1 = gtk_text_view_new ();
        gtk_widget_show (thanks_textview1);
        gtk_container_add (GTK_CONTAINER (thanks_notebook), thanks_textview1);
        gtk_text_view_set_editable (GTK_TEXT_VIEW (thanks_textview1), FALSE);
        gtk_text_view_set_indent (GTK_TEXT_VIEW (thanks_textview1), 10);
        gtk_text_buffer_set_text
        (gtk_text_view_get_buffer (GTK_TEXT_VIEW (thanks_textview1)),
        string1, -1);

        thanks_label1 = gtk_label_new ("Written by");
        gtk_widget_show (thanks_label1);
        gtk_notebook_set_tab_label (GTK_NOTEBOOK (thanks_notebook),
        gtk_notebook_get_nth_page (GTK_NOTEBOOK (thanks_notebook), 0),
        thanks_label1);
    }

    /* "Special Thanks" page */
    {
        thanks_textview2 = gtk_text_view_new ();
        gtk_widget_show (thanks_textview2);
        gtk_container_add (GTK_CONTAINER (thanks_notebook), thanks_textview2);
        gtk_text_view_set_editable (GTK_TEXT_VIEW (thanks_textview2), FALSE);
        gtk_text_view_set_indent (GTK_TEXT_VIEW (thanks_textview2), 10);
        gtk_text_buffer_set_text
        (gtk_text_view_get_buffer (GTK_TEXT_VIEW (thanks_textview2)),
        string2, -1);

        thanks_label2 = gtk_label_new ("Special thanks");
        gtk_widget_show (thanks_label2);
        gtk_notebook_set_tab_label (GTK_NOTEBOOK (thanks_notebook),
        gtk_notebook_get_nth_page (GTK_NOTEBOOK (thanks_notebook), 1),
        thanks_label2);
    }

    /* Close button */
    {
        thanks_hbuttonbox = gtk_hbutton_box_new ();
        gtk_widget_show (thanks_hbuttonbox);
        gtk_box_pack_start (GTK_BOX (thanks_vbox), thanks_hbuttonbox,
                            FALSE, TRUE, 0);
        gtk_container_set_border_width (GTK_CONTAINER (thanks_hbuttonbox), 0);
        gtk_button_box_set_layout (GTK_BUTTON_BOX (thanks_hbuttonbox),
                                   GTK_BUTTONBOX_END);

        thanks_button = gtk_button_new ();
        g_signal_connect (G_OBJECT (thanks_button), "clicked",
                          G_CALLBACK (thanks_window_showhide), NULL);
        gtk_widget_show (thanks_button);
        gtk_container_add (GTK_CONTAINER (thanks_hbuttonbox), thanks_button);
        GTK_WIDGET_SET_FLAGS (thanks_button, GTK_CAN_DEFAULT);

        {
            thanks_button_align = gtk_alignment_new (0.5, 0.5, 0, 0);
            gtk_widget_show (thanks_button_align);
            gtk_container_add (GTK_CONTAINER (thanks_button),
                               thanks_button_align);

            thanks_hbox = gtk_hbox_new (FALSE, 2);
            gtk_widget_show (thanks_hbox);
```

```
            gtk_container_add (GTK_CONTAINER (thanks_button_align),
                               thanks_hbox);

            thanks_button_img = gtk_image_new_from_stock
                           ("gtk-close", GTK_ICON_SIZE_BUTTON);
            gtk_widget_show (thanks_button_img);
            gtk_box_pack_start (GTK_BOX (thanks_hbox), thanks_button_img,
                                FALSE, FALSE, 0);

            thanks_button_label = gtk_label_new_with_mnemonic ("_Close");
            gtk_widget_show (thanks_button_label);
            gtk_box_pack_start (GTK_BOX (thanks_hbox), thanks_button_label,
                                FALSE, FALSE, 0);
        }
    }
}
```

1.36 - Thanks.h

```
/*****************************************************************************/
/* Function prototypes.                                                      */
/*****************************************************************************/

#ifndef THANKS_H
#define THANKS_H

    void thanks_window_showhide  (void);

    void create_thanks_window    (void);

# endif
```

1.37 - Icon.rc

```
1 ICON "logo\\32x32.ico"
```

SEZIONE 2

GSesame 1.0 Compilazione ed installazione

2.1 - Makefile

```
CC = gcc
LD = gcc

GTK_CFLAGS = `pkg-config --cflags gtk+-2.0`
GTK_LIBS = `pkg-config --libs gtk+-2.0`
C_FLAGS = -Wall -g

RM = rm -f
CP = cp

BIN_DIR = ../../bin

OS = $(shell uname -s | tr A-Z a-z)

################################################################################
#  Flags definition for platform dependent compile.                            #
################################################################################

ifeq ($(findstring linux,$(OS)),linux)
       PROG = gsesame
       PLATDEP_DIR = linux
endif

ifeq ($(findstring mingw,$(OS)),mingw)
       WIN_FLAGS = -mwindows
       PLATDEP_DIR = windows
       PROG = gsesame.exe
       ICON = icon.o
endif

ifeq ($(findstring cygwin,$(OS)),cygwin)
       WIN_FLAGS = -mwindows -mno-cygwin
       PLATDEP_DIR = windows
       PROG = gsesame.exe
       ICON = icon.o
endif

################################################################################
#  Source and object files                                                     #
################################################################################

SRC = gsesame.c menu.c toolbar.c list.c project.c saf.c \
      winselcfg.c hvproccfg.c mainproc.c $(PLATDEP_DIR)/platdep.c \
      gsescfg.c primitives.c plotch.c plothv.c warning.c \
      about.c thanks.c

OBJS = $(SRC:.c=.o)

################################################################################
#  Compiling the source file.                                                  #
################################################################################

all: $(PROG)

$(PROG): $(OBJS)
       $(LD) $(WIN_FLAGS) -o $(PROG) $(OBJS) $(ICON) $(GTK_LIBS)

.c.o:
       $(CC) $(C_FLAGS) -c $< -o $@ $(GTK_CFLAGS)
```

```
##############################################################################
#  Compiling the windows icon.                                               #
##############################################################################

icon:
       windres icon.rc $(ICON)

##############################################################################
#  Installing the program.                                                   #
##############################################################################

install:
       $(CP) $(PROG) $(BIN_DIR)

##############################################################################
#  Cleaning everything that can be automatically recreated with "make".      #
##############################################################################

clean:
       $(RM) $(PROG) $(OBJS) $(ICON)
```

2.1 - NSIS install script

```
;--------------------------------
; GSesame configuration script
; for NSIS installer
;--------------------------------
; Include Modern UI

  !include "MUI.nsh"

;--------------------------------
; General

  ; Name and file
  Name "GSesame v.1.0"
  OutFile "GSesame-1.0-Setup.exe"

  ; Default installation folder
  InstallDir "$PROGRAMFILES\GSesame1.0"

;--------------------------------
; Variables

  Var STARTMENU_FOLDER

;--------------------------------
; Interface Settings

  !define MUI_ABORTWARNING

;--------------------------------
; Pages

  !insertmacro MUI_PAGE_WELCOME
  !insertmacro MUI_PAGE_LICENSE "License.txt"
  !insertmacro MUI_PAGE_COMPONENTS
  !insertmacro MUI_PAGE_DIRECTORY
  !insertmacro MUI_PAGE_STARTMENU Application $STARTMENU_FOLDER
  !insertmacro MUI_PAGE_INSTFILES
  !insertmacro MUI_PAGE_FINISH
```

```
  !insertmacro MUI_UNPAGE_WELCOME
  !insertmacro MUI_UNPAGE_CONFIRM
  !insertmacro MUI_UNPAGE_INSTFILES
  !insertmacro MUI_UNPAGE_FINISH

;--------------------------------
; Languages

  !insertmacro MUI_LANGUAGE "English"
  ; !insertmacro MUI_LANGUAGE "Italian"

;--------------------------------
; Installer Sections

Section "GSesame (required)" SecGSesame

  SetOutPath "$INSTDIR\bin"
  File "gsesame\bin\*.*"

  SetOutPath "$INSTDIR\src"
  SetOutPath "$INSTDIR\prj"
  SetOutPath "$INSTDIR\saf"
  SetOutPath "$INSTDIR\out"
  SetOutPath "$INSTDIR\tmp"
  SetOutPath "$INSTDIR\img"

  SetOutPath "$INSTDIR\license"
  File "gsesame\license\*.*"

  ; Create uninstaller
  WriteUninstaller "$INSTDIR\Uninstall.exe"

  !insertmacro MUI_STARTMENU_WRITE_BEGIN Application

    ; create desktop shortcut
    CreateShortCut "$DESKTOP\GSesame.lnk" "$INSTDIR\bin\gsesame.exe" ""

    ; Create shortcuts
    CreateDirectory "$SMPROGRAMS\GSesame"
    CreateShortCut "$SMPROGRAMS\GSesame\GSesame.lnk" "$INSTDIR\bin\GSesame.exe"
    CreateShortCut "$SMPROGRAMS\GSesame\Uninstall.lnk" "$INSTDIR\Uninstall.exe"

  !insertmacro MUI_STARTMENU_WRITE_END

SectionEnd

Section "Source Code" SecSource

  SetOutPath "$INSTDIR\src"
  File /r "gsesame\src\*.*"

SectionEnd

Section "Examples" SecExample

  SetOutPath "$INSTDIR\prj"
  File "gsesame\prj\*.*"

  SetOutPath "$INSTDIR\saf"
  File "gsesame\saf\*.*"

  SetOutPath "$INSTDIR\out"
```

```
  File "gsesame\out\*.*"

  SetOutPath "$INSTDIR\img"
  File "gsesame\img\*.*"

SectionEnd

;--------------------------------
; Descriptions

  ; Language strings
  LangString DESC_SecGSesame ${LANG_ENGLISH} "Install the GSesame main program
and the Sesame modules"
  LangString DESC_SecSource ${LANG_ENGLISH} "Install the source code"
  LangString DESC_SecExample ${LANG_ENGLISH} "Install a project demo and some
examples"

  ; Assign language strings to sections
  !insertmacro MUI_FUNCTION_DESCRIPTION_BEGIN
    !insertmacro MUI_DESCRIPTION_TEXT ${SecGSesame} $(DESC_SecGSesame)
    !insertmacro MUI_DESCRIPTION_TEXT ${SecSource} $(DESC_SecSource)
    !insertmacro MUI_DESCRIPTION_TEXT ${SecExample} $(DESC_SecExample)
  !insertmacro MUI_FUNCTION_DESCRIPTION_END

;--------------------------------
; Uninstaller Section

Section "Uninstall"

 ; Delete Files
   RMDir /r "$INSTDIR\*.*"

 ; Remove the installation directory
   RMDir "$INSTDIR"

 ; Delete Start Menu Shortcuts
   Delete "$DESKTOP\GSesame.lnk"
   Delete "$SMPROGRAMS\GSesame\*.*"
   RmDir  "$SMPROGRAMS\GSesame"

SectionEnd
```

SEZIONE 3

ICmix
Codice sorgente

3.1 - ICmix.c

```
/*****************************************************************************/
/*  ICMIX: Inter Components Mixer for SAF File                               */
/*                                                                           */
/*  Copyright (c) 2005 Poggi Valerio                                         */
/*  Email: valerio.poggi@idpa.cnr.it                                         */
/*****************************************************************************/
/*  This program is free software; you can redistribute it and/or modify     */
/*  it under the terms of the GNU General Public License as published by     */
/*  the Free Software Foundation; either version 2 of the License, or        */
/*  (at your option) any later version.                                      */
/*                                                                           */
/*  This program is distributed in the hope that it will be useful,          */
/*  but WITHOUT ANY WARRANTY; without even the implied warranty of           */
/*  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the            */
/*  GNU General Public License for more details.                             */
/*                                                                           */
/*  You should have received a copy of the GNU General Public License        */
/*  along with this program; if not, write to the Free Software              */
/*  Foundation, Inc., 59 Temple Place, Suite 330, Boston,                    */
/*  MA  02111-1307  USA                                                      */
/*****************************************************************************/

# include <stdio.h>

struct FILESAF
{
    char code   [100];
    char time   [100];
    char freq   [100];
    char ndat   [100];
    char ch0id  [100];
    char ch1id  [100];
    char ch2id  [100];
    char units  [100];
    double data [3];
};

struct FILESAF read_saf (FILE *fp, struct FILESAF file);
char *string_extract (FILE *fp, char *string);
void write_saf (FILE *fp, struct FILESAF file);

/*****************************************************************************/
/*  MAIN                                                                     */
/*****************************************************************************/

int main (int argc, char *argv[])
{
    /*************************************************************************/
    /* Variables declaration                                                 */
    /*************************************************************************/

    FILE *fpref, *fpsaf, *fpch0, *fpch1, *fpch2;
    char filech0 [100], filech1 [100], filech2 [100];
    struct FILESAF strref, strsaf, strch0, strch1, strch2;
    int safndat, refndat, maxndat;
    int year, month, day, hour, minute;
    float second;
    char time[100];
    float saftstart, reftstart;
    float saffreq, reffreq;
```

```
int count;
char string[100];
int i;

/*************************************************************************/
/* Controls                                                              */
/*************************************************************************/

if ( argc < 5 )
{
    printf ("Usage: icmix [-s syncronized] [-u unsyncronized]\n");
    printf ("             target(.saf) reference(.saf) out(.saf)\n\n");
    return 1;
}

if ((strstr(argv[1],"-s") != argv[1]) &&
    (strstr(argv[1],"-u") != argv[1]))
{
    printf ("Usage: icmix [-s syncronized] [-u unsyncronized]\n");
    printf ("             target(.saf) reference(.saf) out(.saf)\n\n");
    return 1;
}

if ((fpsaf = fopen (argv[2],"r")) != NULL)
{
    if ((fpref = fopen (argv[3],"r")) != NULL)
    {

        /*****************************************************************/
        /* Setting output files                                          */
        /*****************************************************************/

        sprintf (filech0, "IC.ch0.%s", argv[4]);
        sprintf (filech1, "IC.ch1.%s", argv[4]);
        sprintf (filech2, "IC.ch2.%s", argv[4]);

        fpch0 = fopen (filech0, "w");
        fpch1 = fopen (filech1, "w");
        fpch2 = fopen (filech2, "w");

        /*****************************************************************/
        /* Reading informations from imput SAF                           */
        /*****************************************************************/

        strsaf = read_saf (fpsaf, strsaf);
        strref = read_saf (fpref, strref);

        sscanf (strsaf.freq, "%f", &saffreq);
        sscanf (strsaf.ndat, "%d", &safndat);
        sscanf (strsaf.time, "%d%d%d%d%d%f",
                &year, &month, &day, &hour, &minute, &second);

        reftstart = ((float)hour * 3600) +
                    ((float)minute * 60) + second;

        sscanf (strref.freq, "%f", &reffreq);
        sscanf (strref.ndat, "%d", &refndat);
        sscanf (strref.time, "%d%d%d%d%d%f",
                &year, &month, &day, &hour, &minute, &second);

        saftstart = ((float)hour * 3600) +
                    ((float)minute * 60) + second;
```

```
/*****************************************************************/
/* Syncronization                                                */
/*****************************************************************/

if (strstr(argv[1],"-s") == argv[1])
{
    if (saftstart > reftstart)
    {
        sprintf (time, "%s", strsaf.time);
        count = (int)((saftstart - reftstart) * saffreq);

        if (count < refndat)
        {
            for (i = 0; i < count; i++)
            {
                fgets (string,100,fpref);
            }
            refndat = refndat - count;
        }
        else
        {
            printf ("Can't syncronize files!\n");
            return 1;
        }
    }
    else
    {
        sprintf (time, "%s", strref.time);
        count = (int)((reftstart - saftstart) * saffreq);

        if (count < safndat)
        {
            for (i = 0; i < count; i++)
            {
                fgets (string,100,fpsaf);
            }
            safndat = safndat - count;
        }
        else
        {
            printf ("Can't syncronize files!\n");
            return 1;
        }
    }
}
else
{
    sprintf (time, "%s", strsaf.time);
}

if (refndat < safndat)
{
    maxndat = refndat;
}
else
{
    maxndat = safndat;
}

/*****************************************************************/
/* Setting header informations for outputs                       */
/*****************************************************************/
```

```
sprintf (strch0.code, "IC STATION (CH0)");
sprintf (strch0.time, "%s", time);
sprintf (strch0.freq, "%s", strsaf.freq);
sprintf (strch0.ndat, "%d", maxndat);
sprintf (strch0.ch0id, "CH0 (%s)", argv[3]);
sprintf (strch0.ch1id, "CH0 (%s)", argv[2]);
sprintf (strch0.ch2id, "CH0 (%s)", argv[2]);
sprintf (strch0.units, "%s", strsaf.units);

sprintf (strch1.code, "IC STATION (CH1)");
sprintf (strch1.time, "%s", time);
sprintf (strch1.freq, "%s", strsaf.freq);
sprintf (strch1.ndat, "%d", maxndat);
sprintf (strch1.ch0id, "CH1 (%s)", argv[3]);
sprintf (strch1.ch1id, "CH1 (%s)", argv[2]);
sprintf (strch1.ch2id, "CH1 (%s)", argv[2]);
sprintf (strch1.units, "%s", strsaf.units);

sprintf (strch2.code, "IC STATION (CH2)");
sprintf (strch2.time, "%s", time);
sprintf (strch2.freq, "%s", strsaf.freq);
sprintf (strch2.ndat, "%d", maxndat);
sprintf (strch2.ch0id, "CH2 (%s)", argv[3]);
sprintf (strch2.ch1id, "CH2 (%s)", argv[2]);
sprintf (strch2.ch2id, "CH2 (%s)", argv[2]);
sprintf (strch2.units, "%s", strsaf.units);

/*****************************************************************/
/* SAF data demultiplexing and writing                           */
/*****************************************************************/

write_saf (fpch0, strch0);
write_saf (fpch1, strch1);
write_saf (fpch2, strch2);

for (i = 0; i < maxndat ; i ++)
{
    fscanf (fpsaf, "%lf", &(strsaf.data[0]));
    fscanf (fpsaf, "%lf", &(strsaf.data[1]));
    fscanf (fpsaf, "%lf", &(strsaf.data[2]));

    fscanf (fpref, "%lf", &(strref.data[0]));
    fscanf (fpref, "%lf", &(strref.data[1]));
    fscanf (fpref, "%lf", &(strref.data[2]));

    fprintf(fpch0, " %lf %lf %lf\n",
            strref.data[0],
            strsaf.data[0],
            strsaf.data[0]);

    fprintf(fpch1, " %lf %lf %lf\n",
            strref.data[1],
            strsaf.data[1],
            strsaf.data[1]);

    fprintf(fpch2, " %lf %lf %lf\n",
            strref.data[2],
            strsaf.data[2],
            strsaf.data[2]);
    }

fclose (fpch0);
fclose (fpch1);
```

```
            fclose (fpch2);
            fclose (fpref);
        }
        else
        {
            printf ("Warning!\n");
            printf ("Unable to open file: %s\n", argv[3]);
            return 1;
        }
        fclose (fpsaf);
    }
    else
    {
        printf ("Warning!\n");
        printf ("Unable to open file: %s\n", argv[2]);
        return 1;
    }

    return 0;
}

/*****************************************************************************/
/*  READ_SAF (Read Header)                                                   */
/*****************************************************************************/

struct FILESAF read_saf (FILE *fp, struct FILESAF file)
{
    char string [100];
    char *extract;

    do
    {
        fgets (string,100,fp);

        extract = string_extract (fp, string);

        /* Saving information in the structure */
        {
            if (strstr(string,"STA_CODE") == string)
            {
                strcpy(file.code,extract);
            }
            if (strstr(string,"START_TIME") == string)
            {
                strcpy(file.time,extract);
            }
            if (strstr(string,"SAMP_FREQ") == string)
            {
                strcpy(file.freq,extract);
            }
            if (strstr(string,"NDAT") == string)
            {
                strcpy(file.ndat,extract);
            }
            if (strstr(string,"CH0_ID") == string)
            {
                strcpy(file.ch0id,extract);
            }
            if (strstr(string,"CH1_ID") == string)
            {
                strcpy(file.ch1id,extract);
            }
            if (strstr(string,"CH2_ID") == string)
```

```
            {
                strcpy(file.ch2id,extract);
            }
            if (strstr(string,"UNITS") == string)
            {
                strcpy(file.units,extract);
            }
        }
    }
    while(strstr(string,"####----") != string);

    return file;
}

/*****************************************************************************/
/*  STRING_EXTRACT                                                           */
/*****************************************************************************/

char *string_extract (FILE *fp, char *string)
{
    char *newline_ptr;
    char *extract = NULL;

    newline_ptr = strchr (string,'\n');
    if (newline_ptr != NULL)
    {
        newline_ptr[0] = '\0';
    }

    newline_ptr = strchr (string,'\r');
    if (newline_ptr != NULL)
    {
        newline_ptr[0] = '\0';
    }

    extract = strchr(string, '=');

    if (extract != NULL)
    {
        extract = extract + sizeof(char);

        while ((extract[0] == ' ') ||
               (extract[0] == '\0'))
        {
            extract = extract + sizeof(char);
        }
    }
    else
    {
        extract = string;
    }

    return extract;
}

/*****************************************************************************/
/*  WRITE_SAF (Write header)                                                 */
/*****************************************************************************/

void write_saf (FILE *fp, struct FILESAF file)
{
    fprintf (fp, "SESAME ASCII data format (saf) v. 1 \n");
    fprintf (fp, "STA_CODE = %s \n",   file.code);
```

```
    fprintf (fp, "START_TIME = %s \n", file.time);
    fprintf (fp, "SAMP_FREQ = %s \n",  file.freq);
    fprintf (fp, "NDAT = %s \n",       file.ndat);
    fprintf (fp, "CH0_ID = %s \n",     file.ch0id);
    fprintf (fp, "CH1_ID = %s \n",     file.ch1id);
    fprintf (fp, "CH2_ID = %s \n",     file.ch2id);
    fprintf (fp, "UNITS = %s \n",      file.units);
    fprintf (fp, "####---------------------------------------\n");
}
```