Compréhension de programmes : Analyse des mécanismes de GIMP (2)

Groupe 1

Vendredi 20 Novembre 2018

Table des matières

1 Introduction

Pour réaliser notre projet GIMP nous avons formé un groupe de 5 membres qui sont :
- PANCHALINGAMOORTHY Gajenthran
- LOKO Loïc
- MOTAMED SALEHI Amin
- BOUCHIHA Abdelrahim
- GHOUIBI Ghassen.

Ce devoir sur GIMP était réparti en 4 étapes : la création d'un élément dans le menu, l'affichage d'un message lorsqu'on accède à un élément du menu, l'ouverture d'une boîte lorsqu'on accède à un élément du menu et enfin l'ajout d'une fonctionnalité.
Nous tenons à préciser avant de passer aux sections suivantes que les commandes seront en gras (`commande`) et que le nom des fichiers et des dossiers sera encadré (*fichier*)

2 La création d'un élément dans le menu

Il existe plusieurs manières de créer/implémenter un élément dans le menu. Ceci s'explique par le fait que les éléments du menus sont "catégorisés". En effet, comme nous l'avons vu dans le dernier devoir, il existe des éléments qui vont permettre d'ouvrir une boîte de dialogue ou des éléments qui vont réaliser une simple action sans ouvrir de boîte de dialogue (le zoom). De plus, un élément dans le menu peut être soit un élément en plus dans la barre de menu ou être un sous-élément d'un élément. Nous allons donc vous décrire les différentes possibilités qu'il existe afin de mieux comprendre les mécanismes du menu (contrairement au précédent devoir, nous allons essayer de détailler le plus possible les lignes de code qui permettent les modifications).

2.1 créer un élément "boîte de dialogue"

Pour créer un élément du menu générant une boîte, il faudra se diriger vers le dossier *app/actions* dans un premier temps (comme nous l'avons vu dans le

dernier devoir) et modifier le fichier *dialogs-actions.c* qui regroupe l'ensemble des éléments "boîte de dialogue" et de ses caractéristiques. Puis dans un second temps, il faudra se concentrer sur le dossier *app/dialogs*, où les fichiers commençant par *[a-zA-z]-dialog.c* correspondent aux différents éléments du menu qui génèrent une boîte de dialogue et s'occupent de la création et la gestion des fonctionnalités de l'élément en question (nous verrons leurs contenus plus tard), le fichier *dialogs-constructors.c* permet la construction d'une boîte de dialogue et le fichier *dialogs.c* qui regroupe l'ensemble des valeurs des éléments "boîte de dialogue" et applique leur fonctionnalité en fonction de la valeur de l'entrée (donnée dans *dialogs-actions.c*)

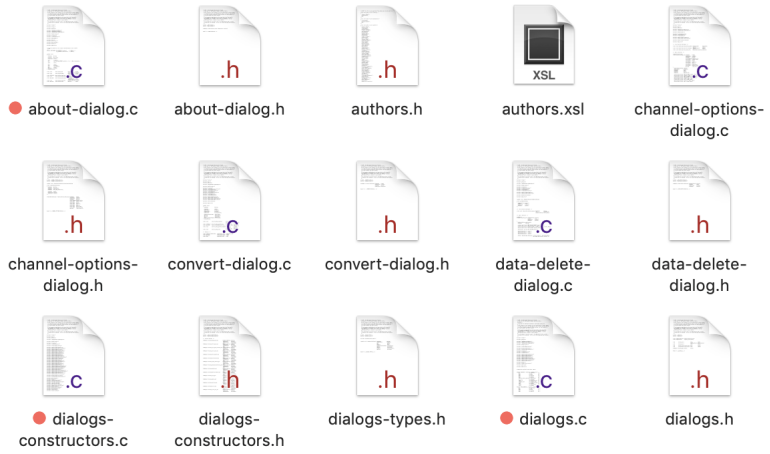

FIGURE 1 – Dossier *app/dialogs*

Pour cette section, nous allons seulement créer un élément et ignorer sa fonctionnalité (nous nous occuperons de cela lors des prochaines sections). Pour cela, nous allons nous focaliser sur le fichier *dialogs-actions.c* qui regroupe l'ensemble des éléments "boîte de dialogue" dans des tableaux de type `GimpActionEntry` ou de ses variantes (par exemple : les entrées du menu sont stockées dans le tableau de type `GimpStringActionEntry`). Nous avons parlé des structures `GimpActionEntry` et de ses variantes lors du précédent rapport, elle permettent de représenter les éléments (identifiant, caractéristiques et fonctionnalité à appliquer). Donc pour créer une nouvelle entrée au menu, nous allons donc insérer un nouvel élément sur le tableau de type `GimpStringActionEntry` qui s'appelle `dialogs_toplevel_actions`.
L'élément rajouté de type `GimpStringActionEntry` aura (voir figure suivante) :
- `dialogs-exemple01` comme champ `name` qui correspond au nom de l'élément et qui va pouvoir être reconnu dans les différents fichiers du programme

(notamment *app/menus/image-menu.xml et app/menus/image-menu.xml.in*)
- `NULL` comme champ `stock_id` Notre entrée n'aura pas de `stock_id`, tout simplement car nous nous ne soucions pas de cela dans cet exercice (voir les sections pour plus d'informations)
- `NC_("dialogs-action", "Exemple-BdDialogue")` comme champ `label` qui correspond au nom de l'élément affiché dans le menu du logiciel (pour l'utilisateur)
- `NULL` comme champ `accelerator` qui correspond au raccourci clavier pour faciliter l'accès au menu. Ici la valeur `NULL` permet d'expliquer qu'il n'y a pas de raccourci pour cette élément. La majorité des entrées "boîte de dialogue" ne possède pas de raccourci clavier
- `NC_("dialogs-actions", "La description de l'entrée Exemple-BdDialogue")` comme champ `tooltip` qui correspond à la description de l'élément lorsque le curseur se trouve sur l'entrée du menu en question.
- `gimp-exemple01-dialog` comme champ `value` qui correspond à la valeur de l'élément, c'est-à-dire à la fonctionnalité qu'il va avoir. Ce champ sera reconnu par le fichier *app/dialogs/dialogs.c* qui va pouvoir ensuite réaliser une action en conséquence (nous verrons cela dans les prochaines sections)
- `GIMP_HELP_EXEMPLE01_DIALOG` comme champ `help_id` qui est l'identifiant de l'élément

```c
struct _GimpStringActionEntry
{
  const gchar *name;
  const gchar *stock_id;
  const gchar *label;
  const gchar *accelerator;
  const gchar *tooltip;
  const gchar *value;
  const gchar *help_id;
};
```

```c
static const GimpStringActionEntry dialogs_toplevel_actions[] =

{ "dialogs-exemple01", NULL,
  NC_("dialogs-action", "Exemple-BdDialogue"), NULL,
  NC_("dialogs-action", "La description de l'entrée Exemple-BdDialogue"),
  "gimp-exemple01-dialog",
  GIMP_HELP_EXEMPLE01_DIALOG },
```

FIGURE 2 – Structure `GimpActionEntry` et ses variantes créées dans *app/widgets/gimpactiongroup.h* et utilisées dans dialogs

Cette ajout là va avoir des répercussions sur d'autres fichiers. Nous allons donc manipuler ces fichiers afin de créer notre élément.

2.1.1 dossier *app/menus*

L'ajout du champ `name dialogs-exemple01` va avoir un impact sur les fichiers *menus/image-menu.xml* et *menus/image-menu.xml.in*. En effet, ces fichiers s'occupent de l'organisation des entrées du menu, il est donc indis-

pensable d'ajouter le champ `name` dans les fichiers .xml. Les fichiers comportent des balises `<menu>` imbriquées dans des balises `<menuitem>`. La balise `<menu>` correspondra à l'élément "principal" du menu qui regroupera des sous-éléments dans des balises `<menuitem>`. Parfois, il se peut qu'il y ait des séparations (représentées par des fins traits horizontaux dans le logiciel) entre les sous-éléments d'un élément. Ces séparations sont causées par la balise `<separator>` afin de mieux hiérarchiser les entrées. De plus, l'ordre des balises `<menuitem>` est important car cela correspond à l'ordre des éléments dans le logiciel.
Pour ajouter une entrée dans le menu "Help", Il faut donc ajouter le champ `name dialogs-exemple01` en tant que `<menuitem>` dans la balise `<menu>` représentant la section "Help" (où nous pouvons trouver "Tips of the Day" ou "About"). Pour trouver la balise `<menu>`, on recherche l'attribut `name` dont le nom est `"Help"`.

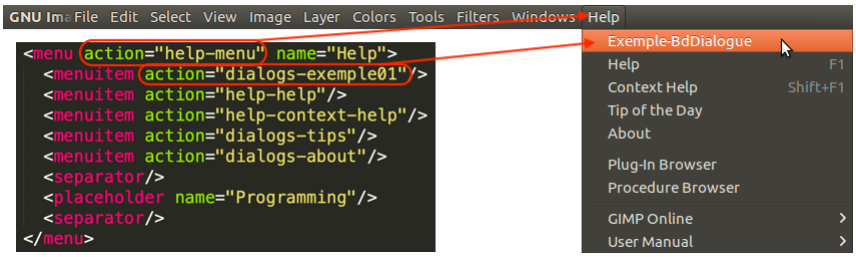

FIGURE 3 – fichier *menus/image-menu.xml*

2.1.2 dossier *libgimpwidgets*

Pour le champ `stock_id` `NULL`, nous aurions pu également créer un nouvel `stock_id`. Si nous avions fait cela, il aurait fallu rajouter cet identifiant dans le fichier *libgimpwidgets/gimpstock.h*. Le fichier *libgimpwidgets/gimpstock.h* contient l'ensemble des `stock_id`, il s'agit de macros qui contiennent le nom des `stock_id`. Ce fichier correspond aux icônes représentant les différents éléments du menu. Nous pouvons déduire cela à partir des noms de fonctions `icon_set_from_inline` et `gtk_icon_source_new`. Par exemple, elle s'occupera de générer une icône représentant une loupe pour `GIMP_STOCK_TOOL_ZOOM` ou encore une ampoule pour `GIMP_STOCK_INFO`. Elle nécessite en grande partie les fonctions du fichier `gtk/gtk.h` pour pouvoir fonctionner.
L'ensemble des icônes des entrées se trouve dans le dossier *themes/Default/images*. Les fichiers images commencent tous par *stock-* afin de mieux se repérer.
Remarque : Nous tenons à préciser qu'au départ, nous pensions que ce champ permettant de classifier les éléments du menu qui possède plus ou moins le même rôle. Au final ce n'était absolument pas cela.

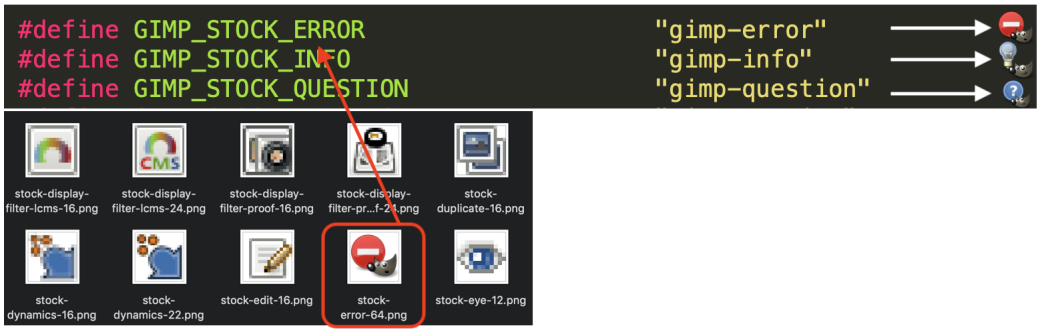

FIGURE 4 – fichier *libgimpwidgets/gimpstock.h* et les icônes dans le dossier *themes/Default/images*

2.1.3 dossier *app/widgets*

L'ajout du champ `help_id GIMP_HELP_EXEMPLE01_DIALOG` va avoir un impact sur le fichier *app/widgets/gimphelp-ids.h*. En effet, ce fichier regroupe l'ensemble des identifiants des éléments du menu. Comme le `stock_id`, il s'agit de macros qui contiennent le nom des `help_id`. Il faut donc penser à rajouter un nouvel `help_id` si on souhaite créer une entrée. Nous remarquons également que les macros commencent tous par `GIMP_HELP` et se terminent par `DIALOG` pour les éléments "boîte de dialogue". Ainsi, nous allons donc créer un `help_id` qui aura la forme suivante : `#define GIMP_HELP_[nom]_DIALOG "gimp-[nom]-dialog"` où `[nom]` correspond au nom de l'élément.

```
#define GIMP_HELP_EXEMPLE01_DIALOG          "gimp-exemple01-dialog"
#define GIMP_HELP_ABOUT_DIALOG              "gimp-about-dialog"
#define GIMP_HELP_COLOR_DIALOG              "gimp-color-dialog"
```

FIGURE 5 – fichier *app/widgets/gimphelp-ids.h*

2.1.4 dossier *app/dialogs/dialogs.c*

L'ajout du champ `name dialogs-exemple01` va avoir un impact sur le fichier *dialogs.c*. Le nom d'un élément du menu doit être spécifié dans le fichier *dialogs.c* afin qu'il appelle pour chaque nom, la fonctionnalité concernée. De plus, le fichier *app/dialogs/dialogs.c* découpe les noms des éléments en différentes catégories : selon son utilité et son emplacement dans le logiciel afin de pouvoir mieux se situer (`DOCKABLE, FOREIGN, DOCK, LISTGRID`...). Dans l'exemple de la figure 6, le `TOPLEVEL` correspond à l'emplacement de l'élément "boîte de dialogue", `"gimp-exemple01-dialog"` correspond au champ `name` et `dialogs_about_get` correspond à la boite de dialogue de l'élément.

Concernant la fonctionnalité, nous nous sommes contentés de mettre la même fonctionnalité que l'entrée "About" vu que cela n'est pas nécessaire pour cette section là.

```
{ "dialogs-exemple01", NULL,
  NC_("dialogs-action", "Exemple-BdDialogue"), NULL,
  NC_("dialogs-action", "La description de l'entrée Exemple-BdDialogue"),
  "gimp-exemple01-dialog",
  GIMP_HELP_EXEMPLE01_DIALOG },
```

```
TOPLEVEL ("gimp-exemple01-dialog",
          dialogs_about_get,                TRUE, FALSE, FALSE),
```

FIGURE 6 – fichier *app/dialogs/dialogs.c* et sa relation avec l'élément créé

2.1.5 conclusion : élément "boîte de dialogue"

Ainsi, l'ajout d'un élément "boîte de dialogue" va avoir des conséquences sur plusieurs fichiers et il ne suffit pas seulement de modifier un seul fichier pour avoir un élément totalement fonctionnel (pas seulement visuel en changeant les chaîne de caractères). Voici un schéma des différents fichiers mis en relation.

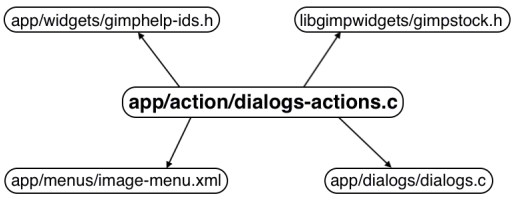

FIGURE 7 – relations entre les différents fichiers impliquant la création d'une entrée "boîte de dialogue"

2.2 créer un élément "actions"

Pour créer un élément "actions", nous allons rester dans le dossier *app/actions*. Contrairement aux éléments "boîte de dialogue" qui sont concentrés sur un seul fichier (en l'occurence *dialog-actions.c*), les éléments "actions" sont répartis dans des fichiers dont le nom est de la forme : *[a-zA-Z]-actions.c*. Un fichier *[a-zA-Z]-actions.c* va contenir un ensemble d'éléments d'une seule "catégorie". Par exemple, *help-actions.c* va contenir l'ensemble des éléments qui vont pouvoir aider l'utilisateur (guides, documentation, astuces...).

Dans chaque fichier *[a-zA-Z]-actions.c*, nous aurons bien évidemment un tableau de structure `GimpActionEntry` qui regroupera l'ensemble des entrées. Cependant, contrairement aux entrées "boîte de dialogue", ces fichiers là auront également des éléments qui vont contenir des sous-éléments (on peut les qualifier de conteneurs). Ils n'auront pas de fonctionnalité particulière mais ils pourront donner accès à d'autres éléments (ils ont souvent comme suffixe `-menu` sur le champ `name` de `GimpActionEntry`). Par exemple, les éléments visibles dans la barre de menus sont des conteneurs vu qu'ils donnent accès à plusieurs sous-éléments.

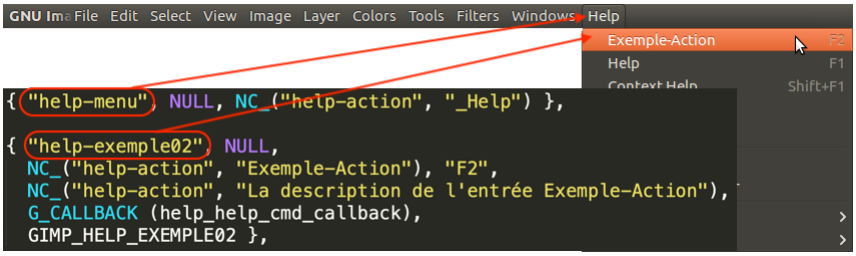

FIGURE 8 – Relation entre le fichier *help-actions.c* et le logiciel Gimp

Dans l'exemple ci-dessus, nous avons rajouté une entrée dans le conteneur "Help" qui correspond au nom `help-menu` dans notre code. Pour l'argument de la `Callback`, nous avons envoyé une fonctionnalité déjà existante vu que ce n'est pas le but de cet exercice. De plus, pour le champ `name help-exemple02`, nous avons respecté la convention de GIMP, à savoir : `"[nomDuConteneur]-[nomDeLentrée]"` où `nomDuConteneur` correspond au nom du conteneur (ici il s'agit de "Help") et `nomDeLentrée` au nom du sous-élément (ici "Exemple-Action"). A noter que pour cette entrée, nous avons rajouté un raccourci clavier (le champ `accelerator` qui vaut `F2`).

Outre la structure des conteneurs (qui, nous le rappelons, n'ont pas besoin d'avoir de fonctionnalité particulière si ce n'est l'accès à d'autres éléments), les éléments sont organisées de la même manière que les éléments "boîte de dialogue" (ce qui est évident vu qu'il s'agit de plus ou moins la même structure) à un détail près : le champ `value` qui est remplacé par le champ `callback` de type `GCallback`.
Nous rappelons que le champ `value` pour la structure `GimpActionEntry` permettait au fichier *app/dialogs/dialogs.c* d'appeler la bonne fonctionnalité de l'élément selon la `value` récupérée. Or, avec la structure `GimpActionEntry`, nous avons un champ de type `GCallback` qui va pouvoir appeler directement la fonctionnalité.

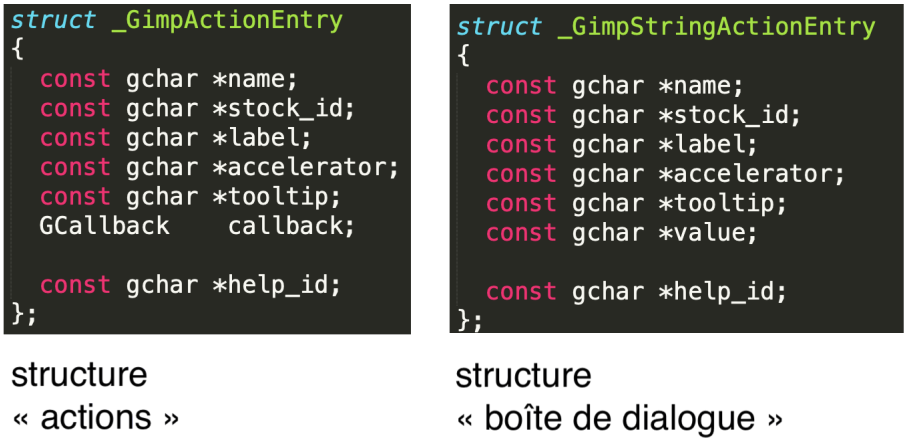

FIGURE 9 – Différence entre la structure `GimpActionEntry` et `GimpStringActionEntry`

Ainsi le champ `callback` va avoir directement accès à la fonctionnalité contrairement à l'élément "boîte de dialogue" qui est obligé de stocker un champ `value` de type `gchar`. Ici nous avons utilisé une fonctionnalité qui existait déjà car l'objectif était surtout de savoir la manière dont on crée une entrée.

2.2.1 conclusion : élément "actions"

En règle générale, les éléments "actions" ressemblent beaucoup aux éléments "boîte de dialogue" : les étapes pour les modifications seront quasiment les mêmes. Les seuls différences observables, ce sont l'emplacement de ces derniers et le champ `GCallback` et `value`. Mais vu que le champ des deux structures est légèrement différent, cela revient à dire que les éléments ne solliciteront pas les mêmes fichiers. En effet, le champ `GCallback` fait appel à une fonction contenu dans un fichier *[a-zA-Z-commands.c]* (nous verrons cela dans une prochaine section) et le champ `value` est utilisé par le fichier *app/dialog/dialogs.c*

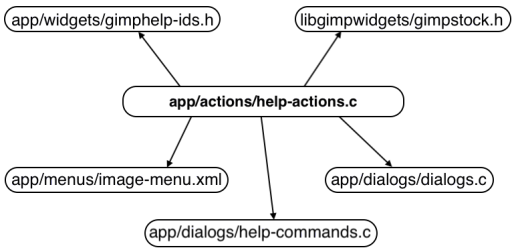

FIGURE 10 – relations entre les différents fichiers impliquant la création d'une entrée "action" pour la section "Help"

2.3 créer un élément via les plug-ins

Il existe également une autre manière d'ajouter un élément du menu, (que cela soit une boîte de dialogue ou non) à l'aide de plug-ins qui se trouve dans le dossier *plug-ins*. L'implémentation des éléments devient donc différente de ce qu'on a vu dans les sections précédentes. En effet, nous n'utilisons pas de structure `GimpActionEntry` ou de ses variantes pour représenter un élément. Les éléments sont directement créés dans le fichier en question, ils sont très peu dispachés comme les éléments "boîte de dialogue" et "actions". Comme plug-ins, nous avons la capture d'écran, les vidéos, les redirections vers des liens hypertextes, des outils s'occupant de traiter l'image...

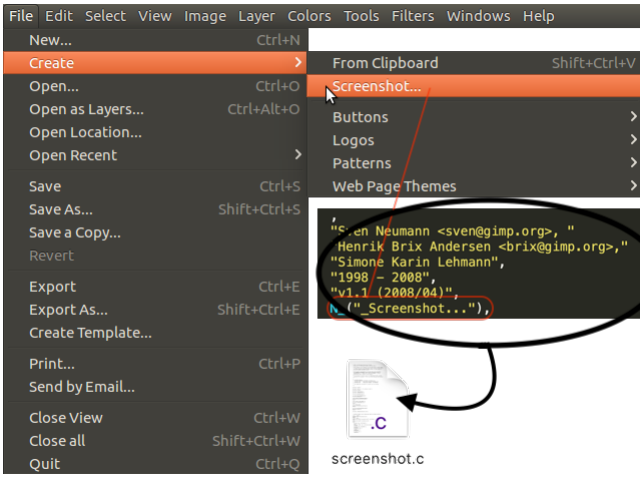

FIGURE 11 – Relation entre l'élément "Screenshot..." et le fichier *screenshot.c*

Par ailleurs, on remarque plusieurs similitudes entre les fichiers du dossier *plug-ins*. Tout d'abord, la fonction *query* qui permet d'initialiser la procédure (la fonctionnalité) avec la fonction `gimp_install_procedure` et de la placer dans le menu à l'aide de la fonction `gimp_plugin_menu_register`, et la

fonction *run* qui va exécuter la fonctionnalité de l'élément.
La fonction `gimp_install_procedure` prend en paramètre :
- le nom de la procédure
- la description de la procédure
- l'auteur
- les copyrights
- la version du plug-in
- le nom affiché sur le logiciel
- les arguments du plug-ins
- la/les valeur(s) de retour

Tandis que la fonction `gimp_install_menu_register` prend en argument le nom de la procédure et son emplacement sur le logiciel si il en a un (en effet, les plug-ins ne sont pas tous des éléments et ne contiennent donc pas la fonction `gimp_install_menu_register`).
Remarque : Certains éléments du menu possèdent la fonction `gimp_plugin_icon_register` qui sert à représenter l'élément sous forme d'icône.

```
gimp_install_procedure (PLUG_IN_PROC,
                        N_("Add a canvas texture to the image"),
                        "This function applies a canvas texture map to the drawable.",
                        "Karl-Johan Andersson", /* Author */
                        "Karl-Johan Andersson", /* Copyright */
                        "1997",
                        N_("_Apply Canvas..."),
                        "RGB*, GRAY*",
                        GIMP_PLUGIN,
                        G_N_ELEMENTS (args), 0,
                        args, NULL);

gimp_plugin_menu_register (PLUG_IN_PROC, "<Image>/Filters/Artistic");
```

FIGURE 12 – Une partie du corps de la fonction `query` du fichier *apply-canvas.c*

Nous verrons l'implémentation d'une entrée via les plug-ins dans la section "Ajout d'une fonctionnalité".

3 L'affichage d'un message lorsqu'on accède à un élément du menu

Si on souhaite afficher un message lorsqu'on accède à un élément du menu, il faut donc s'occuper de la fonctionnalité de l'élément ou plutôt l'emplacement de celui-ci (nous avions rapidement vu cela dans la section précédente). Nous allons donc voir 3 façons d'écrire un message : dans un premier temps

avec une entrée "boîte de dialogue", puis avec une entrée "actions" et enfin une entrée via plug-in.

3.1 message avec un élément "boîte de dialogue"

On sait que le fichier *app/dialog/dialogs.c* reconnaît les entrées "boîte de dialogue" grâce au champ `value` de la structure `GimpActionEntry`. En connaissant cela, le fichier va pouvoir appeler pour chaque `name`, une fonction correspondante à l'entrée selectionnée.

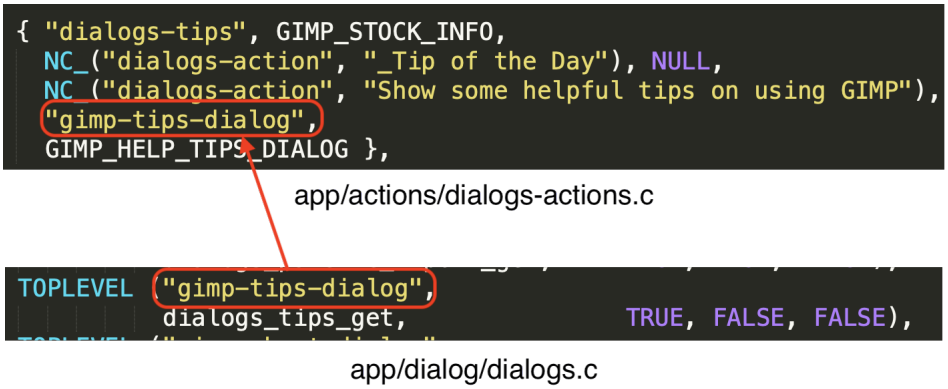

FIGURE 13 – Relation entre le fichier *app/dialog/dialogs.c* et *app/action/dialogs-actions.c*

Dans l'exemple suivant, dans le fichier *app/dialog/dialogs.c*, une fois qu'on a identifié le champ `value` de "Tip of the Day", on peut appeler la fonction `dialogs_tips_get`. Si on cherche le fichier déclarant cette fonction à l'aide de la commande `grep`, on observe qu'il s'agit du fichier *dialogs-constructors.c* (en réalité, il n'y avait pas besoin de faire de commande car on savait déjà (depuis le dernier devoir) que ce fichier rassemble l'ensemble des fonctionnalités de nos entrées "boîte de dialogue" d'où son nom "dialogs-constructor"). Chacune des fonctions de *dialogs-constructors.c* va retourner une boîte de dialogue (d'ailleurs toutes les fonctions retourne un `GtkWidget`, ce qui est évident vu qu'ils sont censés tous retourner une boîte de dialogue).

```
GtkWidget *
dialogs_tips_get (GimpDialogFactory *factory,
                  GimpContext       *context,
                  GimpUIManager     *ui_manager,
                  gint               view_size)
{
  return tips_dialog_create (context->gimp);
}
```

FIGURE 14 – La fonction qui se déclenche pour l'entrée "Tip of the Day"

Nous pouvons donc afficher un message avant de retourner la boîte de dialogue ou alors afficher un message dans la fonction qui traite la boîte de dialogue (à savoir `tips_dialog_create` dans l'exemple ci-dessus). Les fonctions renvoyées par les fonctions du fichier *dialogs_ constructors.c* (dans notre exemple `tips_dialog_create`) se trouve dans le fichier *[nomDeLelement]-dialog.c* où *nomDeLelement* correspond plus ou moins au nom de l'entrée sélectionnée (en règle générale). Par exemple, pour "Tip of the Day", nous chercherons dans le fichier `tips-dialogs.c`. Une fois le fichier trouvée, on vérifie si la fonction existe et si c'est le cas, on peut afficher un message dès le début du corps de la fonction.

```
GtkWidget *
tips_dialog_create (Gimp *gimp)
{
  printf("On entre dans la boîte de dialogue \"Tip of the Day\"");
  GimpGuiConfig *config;
  GtkWidget     *vbox;
  GtkWidget     *hbox;
  GtkWidget     *button;
  GtkWidget     *image;
  gint           tips_count;

  g_return_val_if_fail (GIMP_IS_GIMP (gimp), NULL);
```

FIGURE 15 – Message lors de l'accès à la boîte de dialogue

3.2 message avec un élément "actions"

Pour créer un message avec un bouton "action", on devra donc se diriger vers l'entrée "action" en question comme nous l'avons vu dans la section précédente. Par exemple si on souhaite rajouter un message pour un sous-élément de "Help", nous allons devoir choisir les fichiers *help-actions.c* et

help-commands.c. En effet, le fichier *help-actions.c* dont nous avons vu l'utilité précédemment regroupera l'ensemble des éléments du conteneur "Help" tandis que le fichier *help-commands.c* (sollicité par *help-actions.c* pour le champ `GCallback` de la structure `GimpActionEntry`) va nous servir à initialiser l'ensemble des fonctionnalités des éléments du conteneur "Help". Par conséquent, nous aurons, pour chaque conteneur de sous-éléments, les fichiers *[a-zA-Z]-actions.c* et *[a-zA-Z]-commands.c*. D'un côté, le fichier *[a-zA-Z]-actions.c* va nous permettre de représenter les éléments (description, ID, fonctionnalité appelée), et de l'autre côté le fichier *[a-zA-Z]-commands.c* va nous indiquer les fonctionnalités à appliquer pour nos entrées.

```
{ "help-menu", NULL, NC_("help-action", "_Help") },

{ "help-exemple02", NULL,
  NC_("help-action", "Exemple-Action"), "F2",
  NC_("help-action", "La description de l'entrée Exemple-Action"),
  G_CALLBACK (help_help_cmd_callback),
  GIMP_HELP_EXEMPLE02 },
```

FIGURE 16 – Le champ `GCallback` exécutant la fonctionnalité de l'entrée "Exemple-Action"

De plus, nous remarquons que chaque fonction du fichier *help-commands.c* suivent une convention à savoir : `[nomDeLElement]_cmd_callback` où [nomDeLElement] correspond au nom de l'entrée qui utilise cette fonctionnalité. Par exemple pour l'entrée Help (help-help) qui est contenu dans Help (help-menu), sa fonctionnalité aura comme nom : `help_help_cmd_callback`.
Ainsi, si nous souhaitons placer un `printf` lorsque on exécute une entrée, nous pouvons la placer dans les fonctions présentes dans le champ `GCallback`, c'est-à-dire `help_help_cmd_callback` dans notre exemple.

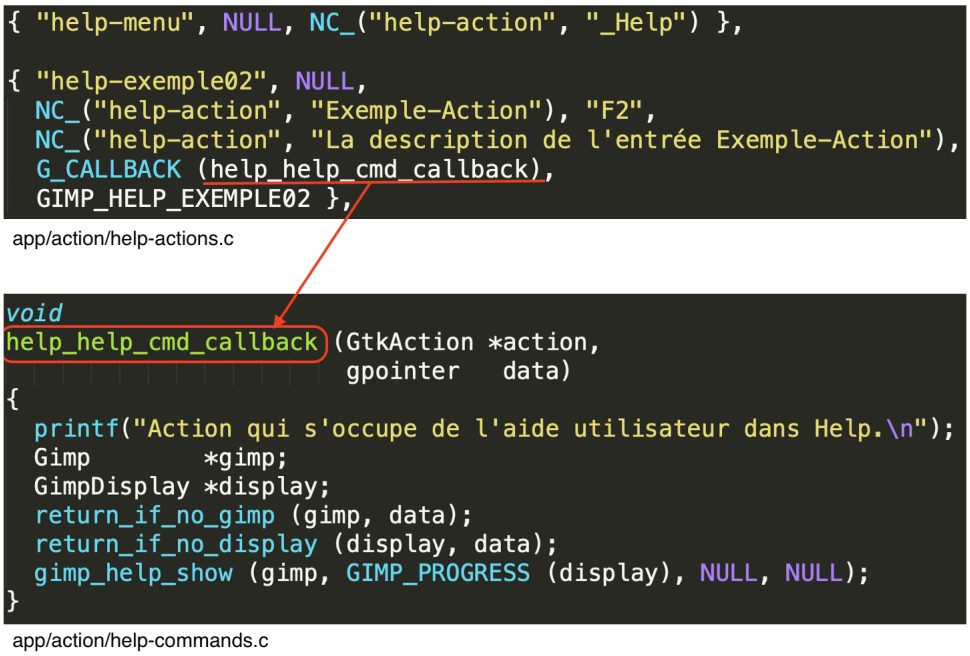

FIGURE 17 – Relation entre le fichier *help-actions.c* et *help-commands.c*

3.3 message avec un élément via plug-in

Dans ce cas, il suffit seulement de trouver le fichier correspondant à l'entrée pour ajouter un message. La commande `grep` pourra nous être d'une grande utilité car il va nous aider à retrouver les fichiers. Par exemple, pour réaliser une capture d'écran, nous devons aller dans : *File* > *Create* > *Screenshot...*. On va donc rechercher la chaîne "screenshot" dans le dossier plug-ins avec `grep -r "Screenshot... plug-ins"`.
Remarque : Parfois, il ne faudra pas écrire tout le texte car certains caractères sont sépares par des "_", sûrement pour les dossier *po* (traductions). Une fois le `grep` exécuté, on aperçoit que le fichier concerné est *plug-ins/common/screenshot.c*.

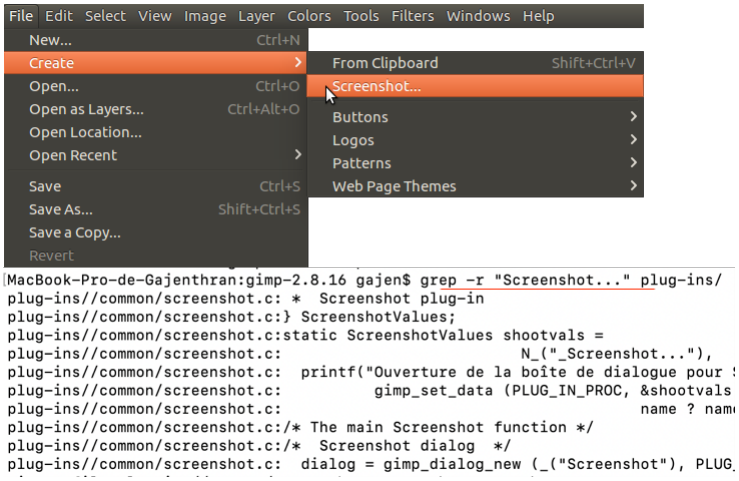

FIGURE 18 – Méthode pour retrouver le fichier traitant l'entrée "Screenshot..."

Néanmoins, il est possible que cette méthode ne fonctionne pas tout le temps ou qu'elle retourne plusieurs fichiers. Dans ce cas, il ne faut pas oublier que pour créer une entrée via un plug-in, nous avons besoin des fonctions `gimp_install_procedure` et `gimp_plugin_menu_register` (vu dans les sections ci-dessus).
Une fois le fichier récupéré, nous pourrons mettre un `printf` dans la fonction `run` qui permet de lancer la procédure dès que l'entrée est sélectionnée.

```
static void
run (const gchar      *name,
     gint              nparams,
     const GimpParam  *param,
     gint             *nreturn_vals,
     GimpParam       **return_vals)
{
  printf("Ouverture de la boîte de dialogue pour Screenshot...\n");
  GimpRunMode        run_mode = param[0].data.d_int32;
  GimpPDBStatusType  status   = GIMP_PDB_SUCCESS;
  GdkScreen         *screen   = NULL;
  gint32             image_ID;
```

FIGURE 19 – Ajout du message dans la fonction `run` du fichier *plug-ins/common/screenshot.c*

Bonus : commande `grep` pour le logiciel en français
Les valeurs de retour de la commande `grep` seront légèrement différentes pour les utilisateurs ayant un logiciel en français et qui souhaite mettre en

argument le nom de l'entrée en français. En effet, car ces dernières retourneront des fichiers *.po*. Il faut donc dans ce cas là, aller dans les fichiers *.po*. Les fichiers *.po* possèdent le nom et le répertoire du fichier traduit.

```
MacBook-Pro-de-Gajenthran:gimp-2.8.16 gajen$ grep -r "Capture d'" po-plug-ins/
po-plug-ins//fr.po:msgstr "_Capture d'écran..."
po-plug-ins//fr.po:msgstr "Capture d'écran"
po-plug-ins//fr.po:msgstr "_Capture d'écran..."
```

```
#: ../plug-ins/common/screenshot.c:253
msgid "_Screenshot..."
msgstr "_Capture d'écran..."
```

FIGURE 20 – Méthode pour retrouver le fichier traitant l'entrée "Screenshot" dans un logiciel en français

4 Ajout d'une boîte de dialogue à un élément

Nous avons déjà vu la création d'une entrée "boîte de dialogue". Désormais il nous reste à ajouter la boîte de dialogue. Pour cela, nous devons nous diriger vers le fichier *dialogs/dialogs.c*. Cette fois-ci au lieu d'utiliser la boîte de dialogue de "About" en appelant la fonction `dialogs_about_get`, nous devons créer notre propre boîte de dialogue.

```
TOPLEVEL ("gimp-exemple01-dialog",
          dialogs_about_get,                TRUE, FALSE, FALSE),
```

FIGURE 21 – Fichier *dialogs/dialogs.c* et boîte de dialogue appelée par notre entrée

Pour créer notre boîte de dialogue, il va falloir créer une nouvelle fonction dans *app/dialogs/dialogs-constructors.c*. Le fichier *app/dialogs/dialogs-constructors.c* s'occupe de construire l'ensemble des boîtes de dialogue de GIMP (en réalité il s'occupe surtout de récupérer les boîtes de dialogue, car la création des boîte de dialogue se passe dans les fichiers *[nomDeLelement]-dialogs.c*), sans compter les plug-ins.

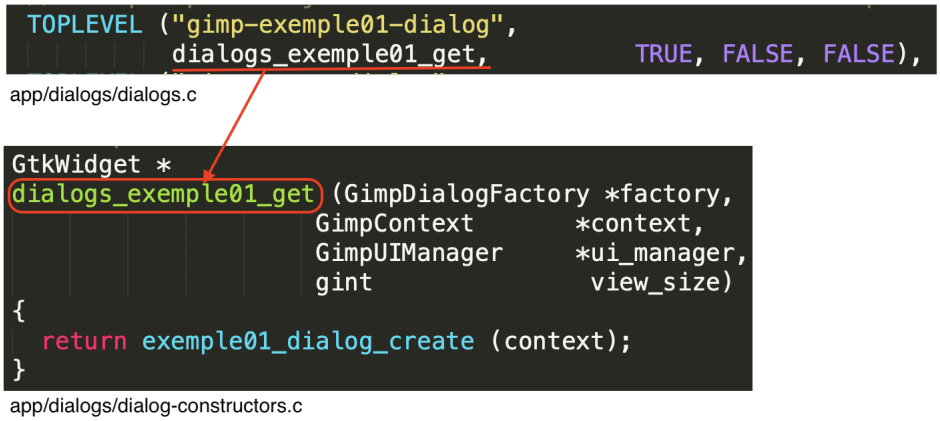

FIGURE 22 – Relation entre les fichiers *app/dialogs/dialogs-constructors.c* et *app/dialogs/dialogs.c*

Nous allons donc créer une fonction `dialogs_exemple01_get` afin de respecter les conventions, de type `GtkWidget`, pour créer une boîte de dialogue. La fonction retournera une fonction présente dans un fichier que nous allons créer. Ce fichier se nommera *exemple01-dialog.c* (on aura donc un fichier header *exemple01-dialog.h*) et s'occupera de la gestion de notre boîte de dialogue. Il ne faut surtout pas oublier d'inclure le fichier header *exemple01-dialog.h* dans le fichier *app/dialogs/dialogs-constructors.c*.
Pour cette exercice, le contenu de notre fichier *exemple01-dialog.c* ressemblera à celui de *about-dialog.c* car on ne s'occupera pas de la fonctionnalité de la boîte de dialogue. Nous nous occupons seulement de créer une nouvelle boîte de dialogue.

```c
GtkWidget *
dialogs_exemple01_get (GimpDialogFactory *factory,
                       GimpContext       *context,
                       GimpUIManager     *ui_manager,
                       gint               view_size)
{
  return exemple01_dialog_create (context);
}
```

app/dialogs/dialogs-constructors.c

```c
GtkWidget *
exemple01_dialog_create (GimpContext *context)
{
  printf("Ouverture de la boîte de dialogue exemple01.\n");
  static GimpAboutDialog dialog;

  g_return_val_if_fail (GIMP_IS_CONTEXT (context), NULL);

  if (! dialog.dialog)
    {
      GtkWidget *widget;
      GtkWidget *container;
      GdkPixbuf *pixbuf;
      GList     *children;
      gchar     *copyright;

      dialog.n_authors = G_N_ELEMENTS (authors) - 1;
```

app/dialogs/exemple01-dialog.c

FIGURE 23 – Relation entre les fichiers *app/dialogs/dialogs-constructors.c* et *app/dialogs/exemple01-dialog.c*

Remarque : Il faut toujours penser à rajouter les fichiers créés dans les *Makefile* au risque de ne pas pouvoir compiler correctement.

Ainsi nous obtenu le schéma suivant :

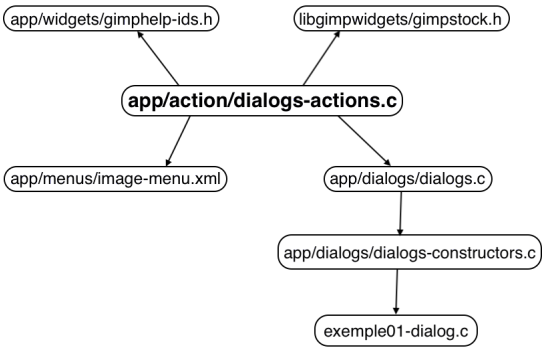

FIGURE 24 – Relations entre les différents fichiers impliquant l'ajout d'une boîte de dialogue

5 Ajout d'une fonctionnalité

Nous avons décidé d'ajouter trois fonctionnalités au logiciel GIMP qui sont : une boîte de dialogue Twitter, un plug-in Twitter et un plug-in Reverse-RGB. Malheureusement nous ne traiterons pas les éléments "actions" dans cette section, les fichiers *.po* (pour la traductions des entrées) par manque de temps.

5.1 boîte de dialogue Twitter

La boîte de dialogue Twitter permet de nous rediriger vers le compte Twitter officiel de GIMP `https://twitter.com/GIMP_Official`. Elle comporte le logo de Twitter, une description du contenu proposé par GIMP sur leur compte, un lien hypertexte vers leur compte et enfin un bouton pour quitter la boîte.

FIGURE 25 – Boîte de dialogue Twitter

Nous avons déjà expliqué la création d'une entrée et l'insertion d'une boîte de dialogue donc nous allons survoler cette étape et nous focaliser sur le code.
Dans un premier temps, nous allons créer une entrée "boîte de dialogue". Pour cela, il faut aller dans le fichier *app/actions/dialog-actions.c* et créer un élément `GimpStringActionEntry` dans le tableau `dialogs_toplevel_actions`. Voici la forme de notre entrée :

```
{ "dialogs-twitter", NULL,
  NC_("dialogs-action", "Twitter"), NULL,
  NC_("dialogs-action", "Redirect to Twitter"),
  "gimp-twitter-dialog",
   GIMP_HELP_TWITTER_DIALOG },
```

FIGURE 26 – Entrée (boîte de dialogue) pour Twitter

Une fois créé, nous devons également modifier les fichiers suivants :
- le champ `name dialogs-twitter` dans *menus/image-menu.xml et menus/image-menu.xml.in*

```
#define GIMP_HELP_TWITTER_DIALOG                    "gimp-twitter-dialog"
```

FIGURE 27 – *menus/image-menu.xml*

- le champ `help_id GIMP_HELP_TWITTER_DIALOG` dans *app/widgets/gimphelp-ids.h*

```
<menu action="help-menu" name="Help">
  <menuitem action="dialogs-twitter"/>
  <menuitem action="help-help"/>
  <menuitem action="help-context-help"/>
  <menuitem action="dialogs-tips"/>
  <menuitem action="dialogs-about"/>
  <separator/>
    <placeholder name="Programming"/>
  <separator/>
</menu>
```

FIGURE 28 – *app/widgets/gimphelp-ids.h*

- le champ `value gimp-twitter-dialog` dans *app/dialogs/dialogs.c*

```
TOPLEVEL ("gimp-twitter-dialog",
          dialogs_twitter_get,           TRUE, FALSE, FALSE),
```

FIGURE 29 – *app/dialogs/dialogs.c*

On remarque, sur la figure ci-dessus, l'ajout également de la fonction `dialogs_twitter_get` qui permettant de construire notre boîte de dialogue dans le fichier *app/dialogs/dialogs-constructors.c*.

```
GtkWidget *
dialogs_twitter_get (GimpDialogFactory *factory,
                     GimpContext       *context,
                     GimpUIManager     *ui_manager,
                     gint               view_size)
{
  return twitter_dialog_create (context);
}
```

FIGURE 30 – *app/dialogs/dialogs-constructors.c*

Cette dernière retourne la fonction `twitter_dialog_create` qui s'occupe de la création de notre boîte de dialogue (en réalité, comme nous l'avons dit précédemment, le fichier *dialogs-constructors.c* se contente seulement de récupérer (d'où le nom `get`) et retourner une fonction de type `GtkWidget`, il ne s'occupe pas de la création de la boîte de dialogue). La création et la gestion de la boîte de dialogue "Twitter" se feront en réalité dans `twitter-dialog.c` et `twitter-dialog.h`. En effet, le fichier *twitter-dialog.c* contiendra les fonctions suivantes : `twitter_dialog_create` qui va créer la boîte de dialogue. et `twitter_dialog_load_logo` qui va chercher le logo correspondant à la boîte de dialogue.

La fonction `twitter_dialog_load_logo` retourne un `GdkPixbuf` représentant une image. Elle se charge dans un premier temps de récupérer le répertoire et le nom du fichier concernés grâce à la fonction `g_build_filename` qui prend en argument le répertoire de l'image (*(data/)images*) et le nom de l'image (*twitter-logo.png*). Puis, on pourra générer et retourner une image à l'aide du nom de fichier récupéré avec la fonction `gdk_pixbuf_new_from_file`.

```
static GdkPixbuf *
twitter_dialog_load_logo (void)
{
  GdkPixbuf *pixbuf;
  gchar     *filename;

  filename = g_build_filename (gimp_data_directory (),
                               "images",
                               "twitter-logo.png",
                               NULL);

  pixbuf = gdk_pixbuf_new_from_file (filename, NULL);
  g_free (filename);

  return pixbuf;
}
```

FIGURE 31 – Fonction `twitter_dialog_load_logo`

La fonction `twitter_dialog_create` va créer la boîte de dialogue comme nous l'avions dit. Voici les différentes étapes de la fonction :
- Elle initialise une variable de type `GimpTwitterDialog`. La structure `GimpTwitterDialog` prend en arguments un `GtkWidget` qui va permettre de stocker une boîte de dialogue et un `int` qui va valoir soit 1 ou soit 0 selon si la création de la boîte de dialogue Twitter s'est bien déroulée ou pas (0 pour une erreur et 1 pour un bon déroulement).

```
typedef struct
{
  GtkWidget  *dialog;
  int         state;
} GimpTwitterDialog;
```

FIGURE 32 – Structure `GimpTwitterDialog`

- Elle charge l'image de la boîte de dialogue à l'aide de la fonction `gdk_pixbuf_new_from_file` vu précédemment (le logo Twitter en l'occurence) qui va être stocker dans la variable `pixbuf`

- On créer un objet qui représentera notre boîte de dialogue avec la fonction `g_object_new` qui sera stocké dans `widget`. Cette dernière prend en argument le nom de code de l'entrée (`gimp-twitter`), la position de notre boîte (`GTK_WIN_POS_CENTER`) pour indiquer qu'elle doit être au centre), le titre de notre boîte (`_("Social Network")`), le nom affiché sur la boîte (`TWITTER_ACRONYM`), la description du contenu proposé par GIMP sur leur compte Twitter (`TWITTER_DESCRIPTION`), le logo Twitter (`pixbuf` evidemment car c'est la variable qui stocke notre image), la redirection vers le site Web Twitter de GIMP (`TWITTER_WEBSITE`) et le label du site Web (`TWITTER_WEBSITE_LABEL`). `TWITTER_WEBSITE_LABEL`, `TWITTER_ACRONYM`, `TWITTER_DESCRIPTION` et `TWITTER_WEBSITE` sont des macros représentants en réalité des chaînes de caractères qui se situe dans le fichier *app/twitter.h* car nous trouvons que c'est beaucoup lisible que de mettre le texte en brut.

```
#define TWITTER_WEBSITE_LABEL \
  _("Visit the official Twitter of GIMP")

#define TWITTER_WEBSITE \
  "https://twitter.com/gimp_official"

#define TWITTER_ACRONYM \
  _("TWITTER")

#define TWITTER_NAME \
  _("Official Twitter of GIMP")

#define TWITTER_DESCRIPTION \
  _("GIMP uses Twitter to share their contents, expand their community and " \
    "inform the public of new features. It's also a way to communicate "     \
    "with the public about improvements and bug in GIMP.")                   \
```

FIGURE 33 – Fichier *app/twitter.h*

- Ensuite on place notre variable `widget` qui contient notre boîte de dialogue désormais dans notre variable de type `GimpTwitterDialog` initialisée au départ. Puis nous libérons la mémoire de `widget` à l'aide de `g_signal_connect`.
- Et enfin on renvoie notre variable de type `GimpTwitterDialog` ou plus précisément le premier champ de `GimpTwitterDialog` (`dialog` qui est un `GtkWidget`).

```
GtkWidget *
twitter_dialog_create (GimpContext *context)
{
  static GimpTwitterDialog dialog;
  dialog.state = 0;

  g_return_val_if_fail (GIMP_IS_CONTEXT (context), NULL);

    if(!dialog.dialog) {
        GtkWidget *widget;
        GdkPixbuf *pixbuf;

        pixbuf = twitter_dialog_load_logo ();

        widget = g_object_new (GTK_TYPE_ABOUT_DIALOG,
                               "role",               "gimp-twitter",
                               "window-position",    GTK_WIN_POS_CENTER,
                               "title",              _("Social Network"),
                               "program-name",       TWITTER_ACRONYM,
                               "comments",           TWITTER_DESCRIPTION,
                               "logo",               pixbuf,
                               "website",            TWITTER_WEBSITE,
                               "website-label",      TWITTER_WEBSITE_LABEL,
                               NULL);

        if (pixbuf)
            g_object_unref (pixbuf);

        dialog.dialog = widget;

        g_signal_connect (widget, "response",
                          G_CALLBACK (gtk_widget_destroy),
                          NULL);

        dialog.state = 1;
    }
  gtk_window_present (GTK_WINDOW (dialog.dialog));

  return dialog.dialog;
}
```

FIGURE 34 – Fonction `twitter_dialog_create`

5.2 plug-in Twitter

Le plug-in Twitter ne contient pas de boîte de dialogue, il réalise la même action que la boîte de dialogue Twitter, il va nous permettre de nous rediriger vers le site Web Twitter de GIMP. Cependant, il est un peu simple à réaliser que la première entrée Twitter. En effet, nous n'avons pas besoin de gérer plusieurs fichiers ici pour les modifications, un seul suffit. L'élément "GIMP Online" de GIMP possède des sous-éléments qui permettent de nous rediriger vers le site Web officiel de GIMP. Nous nous sommes donc demandés la manière dont ils fonctionnaient.

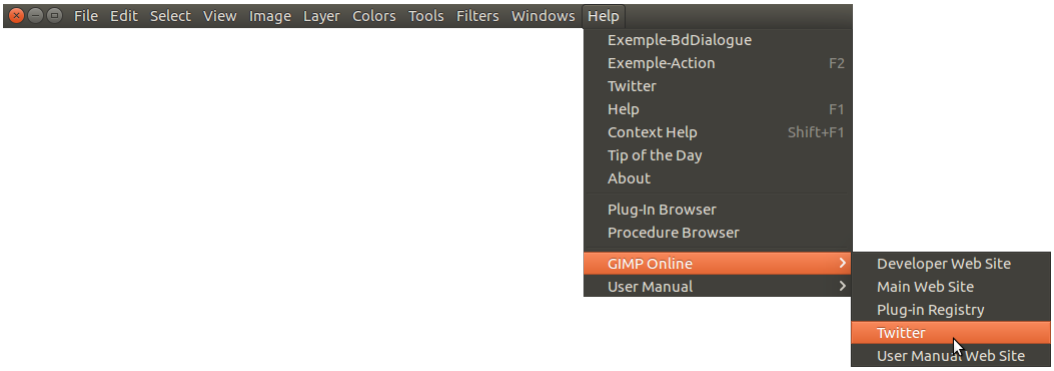

FIGURE 35 – Elément/Conteneur GIMP Online

Ces sous-éléments sont contenus dans `plug-ins/script-fu/scripts/gimp-online.scm`. On va implémenter du code Scheme. Mais en réalité, le code n'est pas difficile à comprendre, il suffit simplement de l'analyser puis l'interpréter afin de comprendre son fonctionnement. Pour rajouter une entrée ou plus précisément une entrée réalisant une redirection vers un lien, il suffit de suivre ces trois étapes :
- déclaration d'une variable qui va stocker notre URL. Nous avons déclaré une variable `gimp-online-twitter-web-site` qui va stocker `"https://twitter.com/gimp_officia`
- représentation et enregistrement de notre entrée dans le logiciel GIMP à l'aide de `script-fu-register` qui prend notre variable déclarée juste avant. Notre entrée aura donc un nom (`"Twitter"`), une description (`"Bookmark to the official Twitter account of GIMP"`), un auteur (`"Henrik Brix Andersen <brix@gimp.org>"`), un copyright (`"Henrik Brix Andersen <brix@gimp.org>"`) et la date de la dernière version (`"2018"`).
- l'emplacement de notre entrée dans le logiciel GIMP à l'aide de `script-fu-menu-register` qui prend notre variable déclarée juste avant et qui lui indique son emplacement (`<Image>/Help/GIMP Online`).
Remarque : Les sous-éléments de GIMP Online sont tous placés dans l'ordre alphabétique.

```
;; Création d'une nouvelle redirection vers un lien,
;; en l'occurence vers le twitter de GIMP
(define (gimp-online-twitter-web-site)
  (plug-in-web-browser "https://twitter.com/gimp_official")
)

;; Informations concernant l'entrée Twitter (nom, description,
;; auteur, date)
(script-fu-register "gimp-online-twitter-web-site"
    _"Twitter"
    _"Bookmark to the official Twitter account of GIMP"
     "Henrik Brix Andersen <brix@gimp.org>"
     "Henrik Brix Andersen <brix@gimp.org>"
     "2018"
     ""
)

;; Emplacement de l'entrée Twitter dans GIMP
(script-fu-menu-register "gimp-online-twitter-web-site"
                         "<Image>/Help/GIMP Online")
```

FIGURE 36 – Fichier *gimp-online.scm*

5.3 plug-in Reverse-RGB

Le plug-in Reverse-RGB consiste cette fois-ci à traiter l'image. En effet, on récupère la couleur primaire la plus représentée sur l'image et fait en sorte qu'elle soit la moins représentée (en mettant sa valeur à 0) et et en plus de ça, on place les deux autres valeurs au maximum. Par exemple si nous avons le logo "Twitter" qui est principalement représenté par du bleu. Nous aurons donc la valeur B de RGB à 0 et RG à la valeur maximale de B au départ. Nous obtiendrons ainsi du jaune. Néanmoins, nous avons également rajouter un "range slider" qui permet d'ajuster la valeur de la couleur la plus représentée.

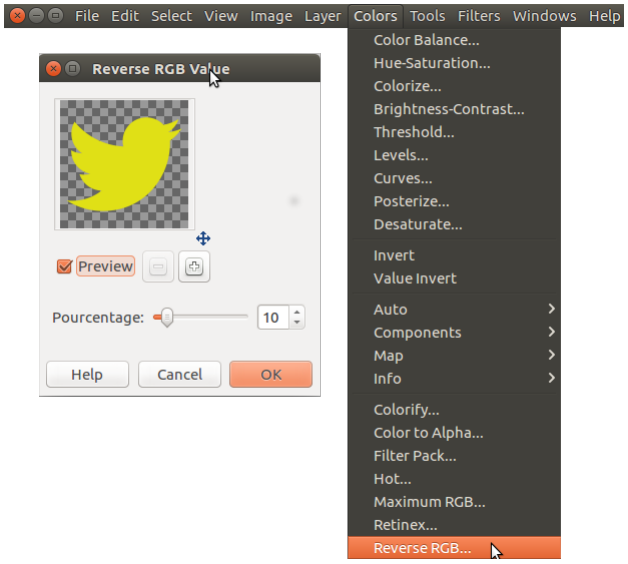

FIGURE 37 – Entrée Reverse RGB... et sa boîte de dialogue

Pour réaliser cela, nous nous sommes inspirés du plug-in Max-RGB. Nous avons surtout laissé la fonction `run` et `main_function` du fichier *max-rgb.c* vu qu'il s'agit toujours de la même chose et modifié la fonction `query`, `reverse_rgb_dialog` qui s'occupe de la création de la boîte de dialogue pour notre Reverse-RGB et `reverse_rgb_func` qui correspond à l'algorithme réalisant le traitement de l'image.

5.3.1 query

Comme nous l'avons vu précédemment, la fonction `query` est consacrée à l'initialisation de notre entrée. Nous appelons la fonction `gimp_install_procedure` qui s'occupe de créer l'entrée, elle prend en paramètre :
- le nom de code du plug-in : `PLUG-IN-PROC` (macro qui correspond à `"plug-in-reverse-rgb"`)
- la description de l'entrée : `N_("Reduce image to pure red, green, and blue")`
- aide pour l'entrée : `"There's no help yet."`
- auteur(s) : `"Panchalingamoorthy Gajenthran (p.gajenthran@gmail.com)"`
- copyright : `"Panchalingamoorthy Gajenthran (p.gajenthran@gmail.com)"`
- date de dernière version : `"December 2018"`
- nom de l'entrée affiché dans le logiciel GIMP : `N_("Reverse RGB...")`

```
gimp_install_procedure (PLUG_IN_PROC,
                        N_("Reduce image to pure red, green, and blue"),
                        "There's no help yet.",
                        "Panchalingamoorthy Gajenthran (p.gajenthran@gmail.com)",
                        "Panchalingamoorthy Gajenthran",
                        "December 2018",
                        N_("Reverse RGB..."),
                        "RGB*",
                        GIMP_PLUGIN,
                        G_N_ELEMENTS (args), 0,
                        args, NULL);
```

FIGURE 38 – Fonction `gimp_install_procedure` du fichier *reverse-rgb.c*

Puis nous appelons la fonction `gimp_plugin_menu_register` qui permet de placer une entrée dans le menu selon l'argument qu'on lui donne. Ici les arguments données sont :
- l'élément qu'on souhaite placer : `PLUG_IN_PROC`
- l'emplacement : `"<Image>/Colors/Modify"` donc dans le conteneur "Color".

Voici la structure du menu "Color" dans le fichier *menus/image-menu.xml*.

FIGURE 39 – Relation entre le fichier *image-menu.xml* et la fonction `gimp_plugin_menu_register`

5.3.2 reverse_rgb_dialog

Dans cette fonction, nous allons nous occuper de la création et gestion de la boîte de dialogue. Pour créer une boîte de dialogue, nous utiliserons la fonction `gimp_dialog_new` où on lui indiquera le nom de la boîte de dialogue (`_("Reverse RGB Value")`), le nom de code de la boîte de dialogue (`PLUG_IN_ROLE`). La fonction retourne un `GtkWidget` et sera retournée dans la variable `dialog`. Une fois la boîte de dialogue créée, on pourra y insérer des éléments `gtk` comme des checkbox, radios, des slides ranger, des tables...

```
dialog = gimp_dialog_new (_("Reverse RGB Value"), PLUG_IN_ROLE,
                          NULL, 0,
                          gimp_standard_help_func, PLUG_IN_PROC,
                          GTK_STOCK_CANCEL, GTK_RESPONSE_CANCEL,
                          GTK_STOCK_OK,     GTK_RESPONSE_OK,
                          NULL);
```

FIGURE 40 – Fonction `gimp_dialog_new`

Tout d'abord, nous allons créer un conteneur dans notre boite de dialogue qui va, comme son l'indique, contenir l'ensemble des éléments. On va devoir utiliser la fonction `gtk_box_new` qui va nous permettre de créer un conteneur (une boîte où on pourra y insérer des éléments), la boîte sera stockée dans la variable `main_vbox` de type `GtkWidget`. Une fois le conteneur créé, il va falloir le placer dans la boîte de dialogue. Pour cela, on utilisera la fonction `gtk_box_pack_start` qui va se charger de prendre en argument le conteneur (ici il s'agit de rajouter dans la boîte de dialogue `dialog`) et l'élément qu'on souhaite rajouter (`main_vbox`).

```
main_vbox = gtk_box_new (GTK_ORIENTATION_VERTICAL, 12);
gtk_container_set_border_width (GTK_CONTAINER (main_vbox), 12);
gtk_box_pack_start (GTK_BOX (gtk_dialog_get_content_area (GTK_DIALOG (dialog))),
                    main_vbox, TRUE, TRUE, 0);
gtk_widget_show (main_vbox);
```

FIGURE 41 – Fonction `gtk_box_new`

Puis nous pouvons continuer à rajouter des éléments dans le conteneur : - `gimp_zoom_preview_new` qui est stockée dans la variable `preview` et qui prend en argument l'image sélectionnée `drawable`. Cette fonction se contente d'afficher l'image dans la boîte de dialogue et nous pourrons utiliser `preview` pour visualiser les modifications sans les valider à l'aide de la fonction `g_signal_connect_swapped` qui prendra en paramètre `preview`. Et on oublie surtout pas d'ajouter l'élément au conteneur (`main_vbox`) avec la fonction `gtk_box_pack_start`.

```
preview = gimp_zoom_preview_new (drawable);
gtk_box_pack_start (GTK_BOX (main_vbox), preview, TRUE, TRUE, 0);
gtk_widget_show (preview);

g_signal_connect_swapped (preview, "invalidated",
                          G_CALLBACK (main_function),
                          drawable);
```

FIGURE 42 – Fonction `gimp_zoom_preview_new`

- il nous manque le "ranger-slider" pour ajuster la couleur. Pour cela, nous allons appeler la fonction `gimp_scale_entry_new` stockée dans `adj` (pour adjustement qui veut dire ajustement). Elle comporte de nombreux arguments car elle a besoin de connaître le nom du "range-slider", les extrémités, la valeur de départ (pas forcément à 0), et surtout pour construire l'élément, nous avons besoin d'un `GTK_TABLE`. On va donc devoir créer avant ça une table avec la fonction `gtk_table_new` et spécifier l'espace nécessaire pour cette table (avec `gtk_table_set_col_spacings` et `gtk_table_set_row_spacings`). Et on l'ajoute dans le conteneur. Cependant, il faut exploiter les valeurs obtenues par le "range-slider", on va donc utiliser `g_signal_connect`. Elle a besoin de savoir sur quel élément appliqué le signal, le nom du signal (une chaîne de caractère, ici `"value-changed"` pour signaler un changement de valeur), la fonction à appliquer lorsqu'il y a un évènement qui se produit sur le "range-slider" `adj` (ici on appelle la fonction `gimp_int_adjustment_update` qui renvoie la valeur du "range-slider" et qui sera stocké dans `reservergb.pourcent`)
Remarque : `reservergb` est de type `ReverseParams` qui possède un champ `pourcent` récupérant la valeur du "range-slider" et un champ `state` pour savoir si le plug-in s'est bien déroulé.

```
table = gtk_table_new (4, 3, FALSE);
gtk_table_set_col_spacings (GTK_TABLE (table), 6);
gtk_table_set_row_spacings (GTK_TABLE (table), 6);
gtk_box_pack_start (GTK_BOX (main_vbox), table, FALSE, FALSE, 0);
gtk_widget_show (table);

adj = gimp_scale_entry_new (GTK_TABLE (table), 0, 1,
                            _("Pourcentage:"), 100, 50,
                            reversergb.pourcent,
                            0, 100, 1, 1, 0,
                            TRUE, 0, 0, NULL, NULL);

g_signal_connect (adj, "value-changed",
                  G_CALLBACK (gimp_int_adjustment_update),
                  &reversergb.pourcent);
```

FIGURE 43 – Fonction `gimp_zoom_preview_new`

En plus de ne pas oublier de placer les éléments créés dans un conteneur avec `gtk_box_pack_start`, il faut penser à l'afficher à chaque fois à l'appel de `gtk_widget_show`.

5.3.3 reverse_rgb_func

La fonction `reverse_rgb_func` prend en argument les données de l'image (`data`). A l'aide d'une boucle `for` qui parcourt la couleur rouge, verte et

bleue, on recherche la valeur maximale des 3 et on place cette valeur dans `max`. Ensuite à l'aide d'une seconde boucle, nous cherchons la couleur ayant la valeur la plus importante. A l'issue de ces deux boucles, nous aurons la valeur maximale entre les 3 couleurs et la couleur correspondante.

```
for (ch = 0; ch < 3; ch++)
  if (param->flag * max <= param->flag * (tmp_value = (*src++))) {
    if (max == tmp_value)
      max_ch += 1 << ch;
    else {
      max_ch = 1 << ch;
      max = tmp_value;
    }
  }

  for(i = 0; i < 3; i++) {
    if(max_ch & (1 << i))
      value = i;
}
```

FIGURE 44 – Recherche de la couleur ayant la valeur maximale

Après avoir trouvé la valeur maximale et sa couleur, on met cette couleur à 0 mais on pourra néanmoins l'ajuster avec le "range-slider" donc en réalité la couleur vaut `reversergb.pourcent * max / 100` où `reversergb.pourcent` est la valeur du "range-slider", `max` est la valeur maximale obtenue. Les deux autres couleurs vont désormais correspondre à la valeur maximale récupérée à savoir `max`.

```
dest[value] = (reversergb.pourcent * max) / 100;
for(i = 0; i < 3; i++) {
  if(value != i) {
    dest[i] = max;
  }
}
```

FIGURE 45 – Changement des couleurs RGB

6 Conclusion

Pour conclure, la compréhension du programme GIMP nous a permis d'acquérir de nombreuses connaissances. En effet, ils nous a permis de traiter un projet de (très) grande envergure, à essayer de s'organiser (dans la construction de nos idées). Heureusement que le cours a été réparti en deux, dans un premier temps l'analyse de l'architecture puis l'analyse des mécanismes pour ne pas trop se perdre.