

Introduction à la programmation des interfaces graphiques

Améliorer sa programmation sous MATLAB



par Jérôme Briot ([Dut sur developpez.com](#))

Date de publication : 01/06/2007

Dernière mise à jour : 17/03/2009

Cet article est une introduction au développement des interfaces graphiques sous MATLAB.

Contenu : l'article comporte :

- une description succincte des objets graphiques (hiérarchie, identifiant, propriétés)
- une présentation de l'outil GUIDE pour le développement des interfaces graphiques
- une évaluation du GUIDE par rapport au codage classique "à la main"

Public visé : cet article est destiné aux débutants ayant déjà quelques notions de MATLAB et surtout aux utilisateurs avancés qui seront incités à ne pas utiliser l'outil GUIDE pour développer les interfaces graphiques.



Votre avis et vos suggestions sur cet article m'intéressent ! Alors après votre lecture, n'hésitez pas :

Avant-propos.....	3
1 - Les objets graphiques et leur fonctionnement.....	4
Les objets graphiques.....	4
Les identifiants des objets graphiques.....	5
Les propriétés des objets graphiques.....	6
2 - Avec GUIDE ou bien en solo ?.....	7
Présentation.....	7
Exemple à partir d'un GUI simple.....	7
Conclusion.....	11
Pour résumer.....	11
Remarque.....	11
Remerciements.....	12

Avant-propos

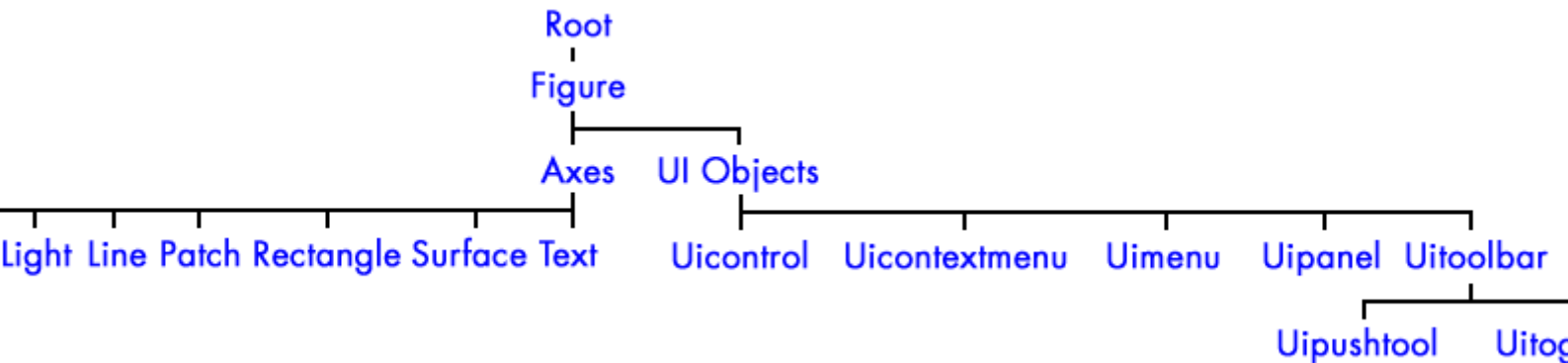
Les **IHM** (Interfaces **H**omme **M**achine), sont appelées **GUI** (**G**raphical **U**ser Interfaces) dans MATLAB. Elles permettent à l'utilisateur, grâce à des objets graphiques (boutons, menus, cases à cocher, ...) d'interagir avec un programme informatique.

Du fait du nombre important d'objets et surtout du nombre encore plus élevé des paramètres associés, leur programmation "à la main" déroute généralement le débutant. Depuis la version 5.0 (1997), MATLAB possède un outil IDE dédié à la création des interfaces graphiques. Cet outil, appelé **GUIDE** (**G**raphical **U**ser Interface **D**evelopment **E**nvironment), permet de concevoir intuitivement ces interfaces graphiques.

1 - Les objets graphiques et leur fonctionnement

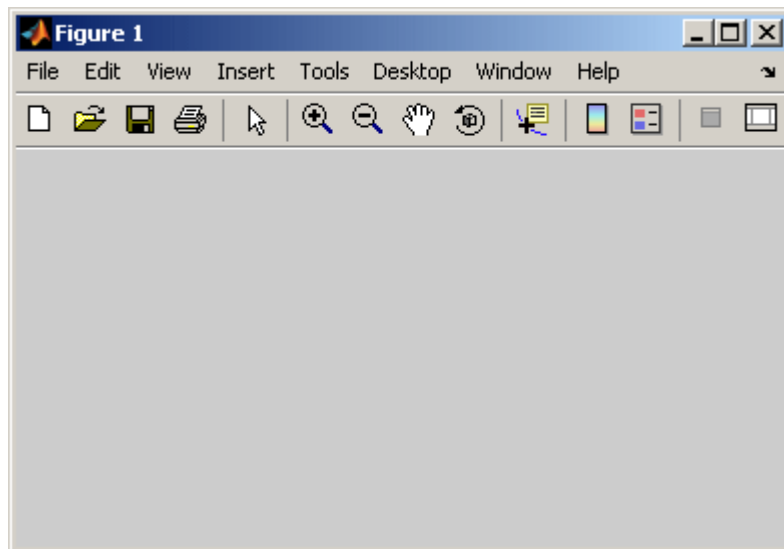
Les objets graphiques

Sous Matlab, les objets graphiques sont disposés selon une hiérarchie pyramidale parent-enfant :

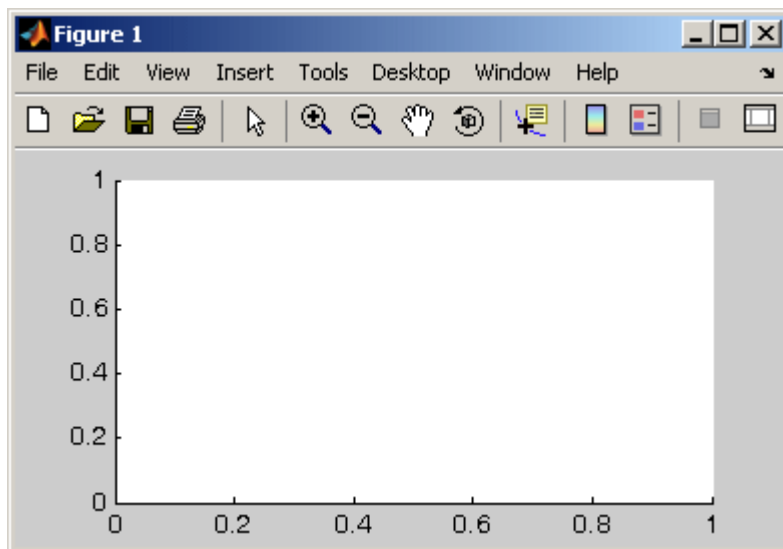


Au sommet de la hiérarchie se trouve l'objet **Root**. Cet objet est invisible (on peut se le représenter comme étant l'écran de l'ordinateur). L'utilisateur n'interagit que très rarement avec cet objet.

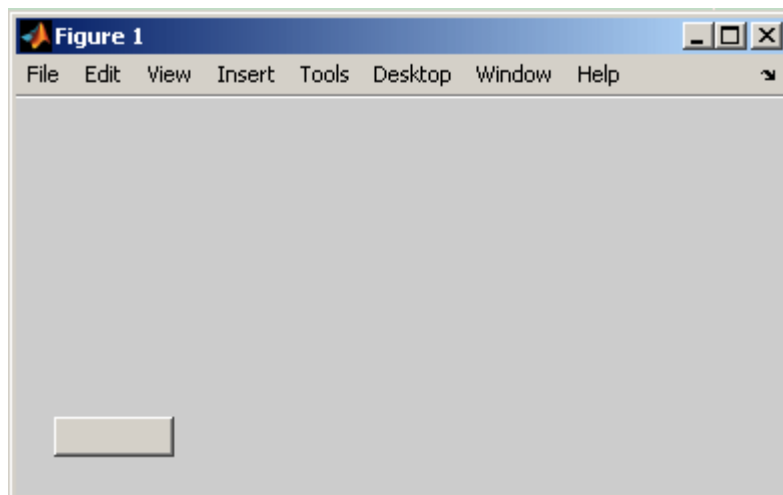
Ensuite, on trouve les objets de type **Figure**. Ce sont les conteneurs visibles où sont disposés tous les autres objets enfants. Plusieurs objets Figure peuvent être ouverts simultanément et peuvent éventuellement communiquer entre eux.



Viennent ensuite les objets de type **Axes** qui sont les zones de traçage des graphiques (2D ou 3D). Ces objets ont pour enfants, tous les objets représentant des résultats mathématiques (courbes, surfaces, images, maillages, etc.). Un objet Figure peut contenir plusieurs objets Axes simultanément.



On trouve également au même niveau, les objets **UI (User Interface)** tels que des boutons, des menus, des cases à cocher, ... Ces objets permettent à l'utilisateur d'interagir dynamiquement à la souris avec le GUI.



Les identifiants des objets graphiques

A la création d'un objet, MATLAB lui attribue automatiquement un identifiant (handle), sous la forme d'un nombre réel unique qui peut être stocké dans une variable. Ceci permet de retrouver à tout moment un objet graphique au cours du fonctionnement d'une interface. Cet identifiant existe tant que l'objet existe. Dès que l'objet est détruit, cet identifiant disparaît.

Par exemple, à la création d'un objet Figure :

Création d'un objet et récupération de son identifiant

```
h=figure
h =
    1
```

L'identifiant est ici un nombre entier (et non pas un réel). C'est un cas particulier, les objets Figure étant par défaut identifiés par des entiers.

Le programmeur gère les identifiants, soit avec la fonction GUIHANDLES, soit avec les fonctions FINDOBJ/FINDALL. Quelques identifiants particuliers peuvent être gérés avec les fonctions suivantes :

- GCA qui récupère l'identifiant de l'objet Axes courant
- GCBF qui récupère l'identifiant de l'objet Figure où se trouve l'objet graphique dont l'action est en cours
- GCBO qui récupère l'identifiant de l'objet graphique dont l'action est en cours
- GCF qui récupère l'identifiant de l'objet Figure courant
- GCO qui récupère l'identifiant de l'objet graphique courant

Les propriétés des objets graphiques

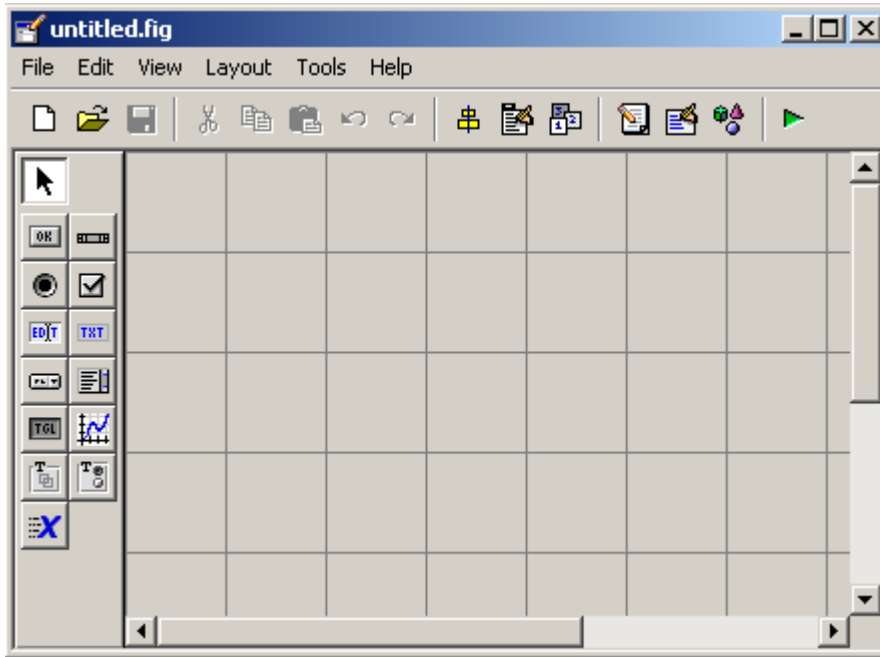
Chaque objet graphique possède des propriétés (position, couleur, action, etc.) qui sont définies à sa création et qui peuvent être modifiées dynamiquement au cours du fonctionnement du GUI. Ces propriétés peuvent être récupérées et modifiées en utilisant l'identifiant de l'objet et les fonctions **GET** et **SET**. La difficulté consiste, bien entendu, à apprendre et à maîtriser ces nombreuses propriétés :

- les propriétés de l'objet **Root**
- les propriétés de l'objet **Figure**
- les propriétés de l'objet **Axes**
- les propriétés des objets **Uicontrol**

2 - Avec GUIDE ou bien en solo ?

Présentation

Le GUIDE est un outil graphique qui regroupe tout ce dont le programmeur à besoin pour créer une interface graphique de façon intuitive.



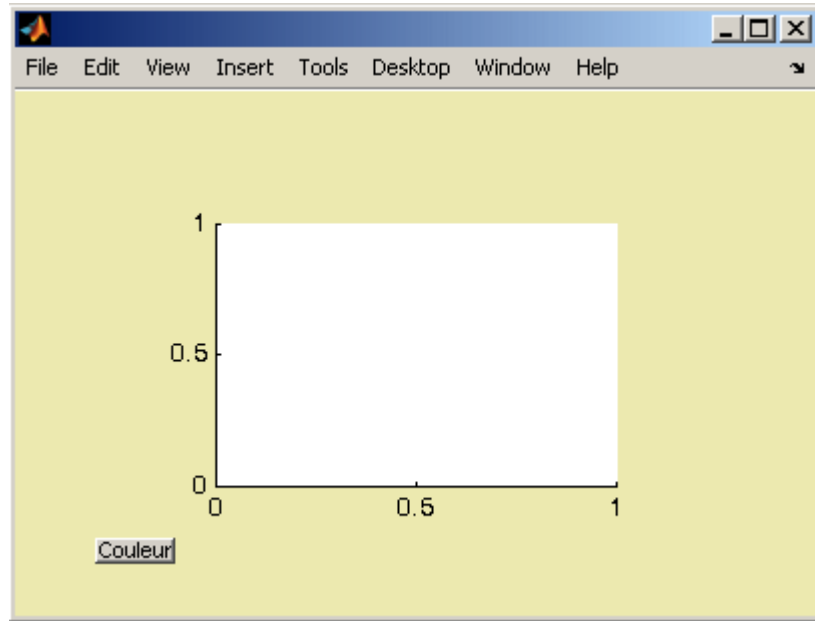
Le placement des objets est réalisé par sélection dans la boîte à outils, mise en place et mise à dimension à la souris. Un double-clic sur chaque objet permet de faire apparaître un menu avec les propriétés de cet objet. Leur modification et l'aperçu de ces modifications sont immédiats. Au final, le code est généré automatiquement et l'interface est enregistrée sous deux fichiers portant le même nom mais dont les deux extensions sont *.fig* et *.m*. Le premier contient la définition des objets graphiques. Le second contient les lignes de code qui assurent le fonctionnement de l'interface graphique.

L'utilisation du GUIDE semble donc être LA méthode de programmation des GUI sous MATLAB. Mais, comparons cette méthode à la programmation des GUI "à la main" à l'aide d'un exemple simple.

Exemple à partir d'un GUI simple

- **But** : comparer la programmation des GUI, à l'aide du GUIDE et "à la main".

On se propose de créer une interface graphique simple, composée d'une figure contenant un objet Axes et un objet Uicontrol de type Pushbutton. Lorsque l'on clique sur l'objet Pushbutton, l'objet Axes change de couleur de façon aléatoire.



- **Avec le GUIDE :**

Après avoir mis en place tous les objets et ajusté toutes les propriétés, le GUIDE génère deux fichiers. Un fichier *.fig* (non éditable) contenant les objets graphiques (Figure, Axes et Pushbutton) et un fichier *.m* contenant les lignes de code suivantes :

Code généré automatiquement par le GUIDE

```
function varargout = gui(varargin)
% GUI Application M-file for gui.fig
%   FIG = GUI launch gui GUI.
%   GUI('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 02-May-2007 18:24:02

if nargin == 0 % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Generate a structure of handles to pass to callbacks, and store it.
    handles = guihandles(fig);
    guidata(fig, handles);

    if nargin > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        if (nargout)
            [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
        else
            feval(varargin{:}); % FEVAL switchyard
        end
    catch
        disp(lasterr);
    end

end

%| ABOUT CALLBACKS:
%| GUIDE automatically appends subfunction prototypes to this file, and
```

Code généré automatiquement par le GUIDE

```

%| sets objects' callback properties to call them through the FEVAL
%| switchyard above. This comment describes that mechanism.
%|
%| Each callback subfunction declaration has the following form:
%| <SUBFUNCTION_NAME>(H, EVENTDATA, HANDLES, VARARGIN)
%|
%| The subfunction name is composed using the object's Tag and the
%| callback type separated by '_', e.g. 'slider2_Callback',
%| 'figure1_CloseRequestFcn', 'axis1_ButtondownFcn'.
%|
%| H is the callback object's handle (obtained using GCBO).
%|
%| EVENTDATA is empty, but reserved for future use.
%|
%| HANDLES is a structure containing handles of components in GUI using
%| tags as fieldnames, e.g. handles.figure1, handles.slider2. This
%| structure is created at GUI startup using GUIHANDLES and stored in
%| the figure's application data using GUIDATA. A copy of the structure
%| is passed to each callback. You can store additional information in
%| this structure at GUI startup, and you can change the structure
%| during callbacks. Call guidata(h, handles) after changing your
%| copy to replace the stored original so that subsequent callbacks see
%| the updates. Type "help guihandles" and "help guidata" for more
%| information.
%|
%| VARARGIN contains any extra arguments you have passed to the
%| callback. Specify the extra arguments by editing the callback
%| property in the inspector. By default, GUIDE sets the property to:
%| <MFILENAME>('<SUBFUNCTION_NAME>', gcbo, [], guidata(gcbo))
%| Add any extra arguments after the last argument, before the final
%| closing parenthesis.

% -----
function varargout = pushbutton1_Callback(h, eventdata, handles, varargin)

set(gca, 'color', rand(1,3));
    
```

De toute évidence, ce code est très difficile à lire et à exploiter même pour un programmeur averti. Son évolution et sa maintenance sont donc également très difficiles (qui plus est sans l'utilisation du GUIDE). Mais la principale limitation vient du fait que la programmation des objets graphiques n'apparaît nulle part (fichier .fig crypté).

- **Sans le GUIDE :**

La même interface graphique programmée "à la main" peut être écrite dans un seul fichier .m:

Code écrit à la main

```

function gui2

% Création de l'objet Figure
figure('units','pixels',...
    'position',[250 250 500 500],...
    'color',[0.925 0.913 0.687],...
    'numbertitle','off',...
    'name','Exemple sans le GUIDE');

% Création de l'objet Axes
axes('units','normalized',...
    'position',[0.25 0.25 0.5 0.5]);

% Création de l'objet Uicontrol Pushbutton
uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.1 0.1 0.1 0.05],...
    'string','Couleur',...
    'callback','set(gca, 'color', rand(1,3));');
    
```

Ce code est relativement simple et, mis à part les propriétés spécifiques à chaque objet, il est relativement lisible. Un programmeur pourra aisément faire évoluer ce code quelque soit la version de MATLAB utilisée.

- **Au final :**

Bien que la génération automatique du code permette d'éviter les erreurs de syntaxe (généralement périlleuses à corriger pour le débutant), celle-ci peut vite devenir une limitation pour les utilisateurs plus avertis. Elle pénalise grandement l'évolutivité ultérieure des GUIs en masquant une grande partie du code. Au final, cela peut même entraîner des problèmes de compatibilité entre des codes générés sous différentes versions de MATLAB.

Le GUIDE est donc un outil efficace pour le programmeur débutant car la gestion des objets y est très intuitive. Il peut également être utile dans la conception des interfaces graphiques complexes pour gérer la mise en position des objets. Ces deux cas mis à part, le programmeur veillera le plus tôt possible à programmer ses interfaces graphiques à la main.

Conclusion

Pour résumer

- Les interfaces graphiques se nomment **GUI** sous MATLAB
- Les objets graphiques sont hiérarchisés (parent-enfant) : **Root => Figure => Axes et Uicontrol**
- Chacun des objets graphiques possèdent de nombreuses **propriétés** que le programmeur doit apprendre
- Il existe deux techniques de programmation : à l'aide de l'outil **GUIDE** ou "**à la main**"
- Le **GUIDE** convient bien à l'apprentissage mais devient vite **inefficace** (génération automatique du code)
- La programmation "**à la main**" reste la technique la plus **efficace** à long terme pour programmer les GUI

Remarque

L'auteur de ce tutoriel invite à créer les interfaces graphiques (GUI) sous MATLAB "à la main". Sa conviction repose sur sa propre expérience et surtout sur les discussions qu'il a eu avec d'autres programmeurs MATLAB. On notera que cette opinion est généralement admise dans la communauté internationale MATLAB (1).

Remerciements

L'auteur tient à remercier **Fleur-Anne.Blain** pour la correction orthographique de cet article et **Caro-Line** pour son aide pendant la rédaction de ce même article.

1 : GUIs: GUIDE vs Code Only