Guide de démarrage pour Boa Constructor

<term boa>Boa Constructor, un EDI/RAD pour Python</term> ;#; Kevin Gill
1 novembre 2000
mars 2005 mise à jour de Boa 0.4.0
juillet 2007 mise à jour de Boa 0.6.0
par Werner F. Bruhin ;#;

Une visite de l'environnement de développement Boa Constructor

Installation et démarrage de Boa Constructor

Boa s'exécute sur n'importe quelle plate-forme Windows 32 bits ou sur n'importe quelle plate-forme UNIX avec la boîte à outils de fenêtrage GTK+.

Installation sous Windows

- Instructions détaillées sur wxPython.org
- Installer une version récente de Python. Pour Boa 0.4.0, il faut la version 2.2 ou plus.
- Installer les extensions wxPython. L'installeur standard installer à la fois les bibliothèques wxWidgets et les paquets wxPython. Vérifier que la version de wxPython correspond à celle de Python qui est installée.
- Boa 0.4.0 ou plus nécessite d'utiliser au moins wxPython 2.5.4.1
- Utiliser l'installateur pour Boa (fichiers se terminant par “win32.exe”)
  - Ou créer un répertoire pour Boa, par exemple ..\site-packages\boa
  - Extraire le fichier .zip de Boa constructeur dans ce répertoire. Il peut être nécessaire d'installer un utilitaire comme winzip pour extraire le fichier zip.
  - Créer sur le bureau un raccourci vers le fichier boa.py.

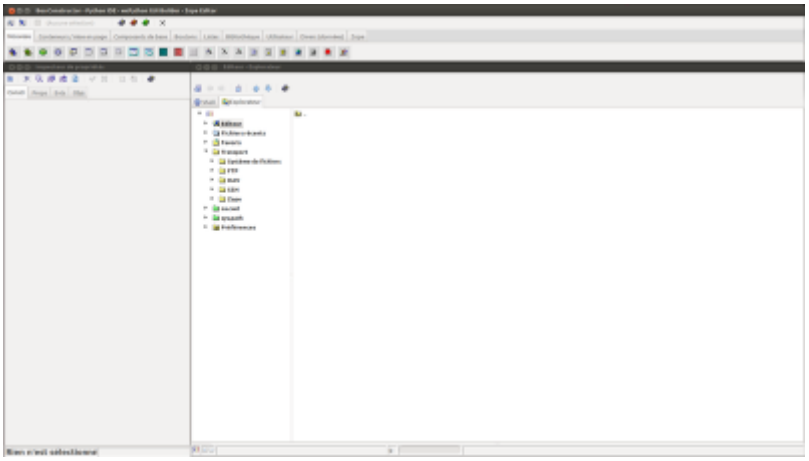
6. Lancer le nouveau raccourci pour exécuter l'environnement de développement Boa.

UNIX Installation

- Detailed instructions from wxPython.org
- These components can also be installed from binary RPMs.
- Install a recent version of Python. As of Boa 0.4.0 you must use version 2.2 or later.
- If GTK+ is not installed, then build and install it. You must use version 1.2 or later.
- If wxWidgets is not installed, then build and install it. Use the version of wxWidgets for GTK+ (there are several versions, e.g. the Microsoft Windows version). You must use version 2.5 or later.
- If wxPython is not installed, then build and install it. The version of wxPython must match the version of wxWidgets, i.e. 2.5.x for wxWidgets 2.5.x.

- Boa 0.4.0 or higher requires to use at least wxPython 2.5.4.1
- Boa constructor requires that the Scintilla editor (stc) and the open GL (ogl) components for wxWidgets are installed. These components are distributed in both the wxPython and the wxWidgets contrib directories. If in doubt, build and install the versions in the wxPython release.
- Create a directory for the Boa Constructor software, e.g. /usr/local/boa. Extract the Boa Constructor package to this directory. The gunzip program may be required to extract the archive
- Create a symbolic link to boa.py on your path, e.g. ln -s /usr/local/boa/boa.py /usr/bin
- In your shell execute boa using the command 'boa.py'

You will now see the tools of the Boa Constructor development environment



This is divided into three sections/areas:

- Palette window on top
  - A tool bar to open/close application windows
  - Multiple tabs
    - New - create new modules, applications etc.
    - Containers/Layout, Basic Controls etc give you access to different components

2. Inspector window on the bottom left
   - This section is also used to display e.g. the Debugger or Class Browser
   - It is used to change properties of components and to define events.
3. Editor/Explorer window on the bottom right
   - This section has multiple tabs
     - Shell - provides access to a Python interpreter, this makes it very easy/confortable to test out small code sections.
     - Explorer - allows to navigate the file system, gives access to the wxPython demo (via a Plug-in), allows to change your preferences for Boa, and it will have additional tabs, one for each file you are working on

Installation from CVS

The Boa installation guide on the wxPython wiki page (http://wiki.wxpython.org/BoaInstallGuide) provides details on how to obtain Boa from CVS

The Palette Window

The Palette window is the main window of the Boa Constructor Program. Closing the Palette window will terminate the program. The Palette contains two sections, a toolbar, and a multi-pane notebook.

The toolbar gives access to the other windows in the Boa Constructor application. You can hold the mouse pointer over the different icons and help tips will appear. These help tips will tell you what the buttons do.

The notebook (or tabbed pane) provides a set of components which you can use while building your applications. The components are grouped for logical access.

The first group of components is called 'New'. This group provides components for creating new “files”, that can be a Python Application, a Python Module, a Setup file, a Boa Application or others.

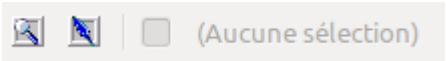
In earlier versions there was also a tab for “Dialogs”, to insert the code for e.g. a file dialog you can now use the new “code template” feature. At the place in your code where you want the dialog press “Alt-t” and select from the drop down list.

The icon shown above is used to create a new Boa Application (its tool tip is wx.App, refering to the base class it uses). The new application consists of the application source file (App1.py) and the initial frame (Frame1.py).

The application file is used as the starting point for the application. The default new application simply loads the form. The form is a blank form. You can add components to this form.

Boa constuctor opens the source code for the new application and the new form in the Editor Window.

Toolbar Icons - Application



Clicking the first one will bring the Inspector window to the foreground.
Clicking the second one will bring the Editor window to the foreground.
The check box will indicate that a compenent is selected and the text “Nothing selected” will be replaced with e.g. wx.Panel

Toolbar Icons - Help

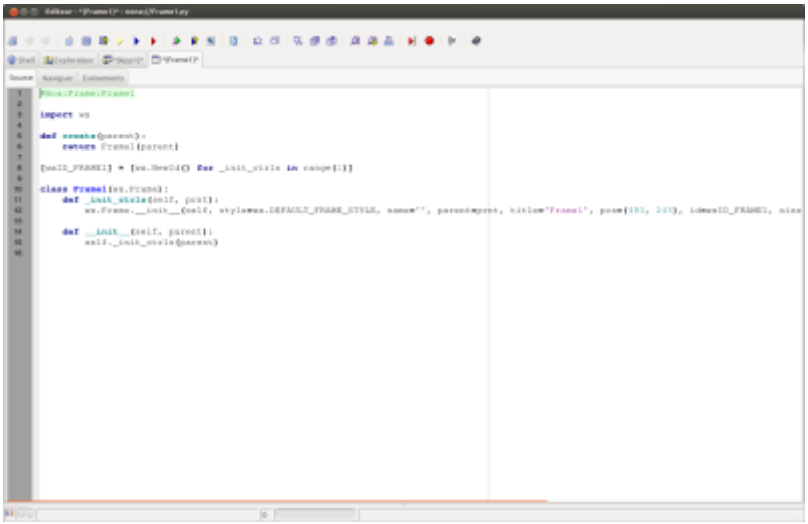
Clicking the first one will bring up Boa help or help for the selected component
Clicking the second one will bring up wxPython help
Clicking the third one will bring up Python help

The Editor Window

The Editor window is one of the three windows which is created when you run Boa Constructor. If you close the Editor Window, you can re-open it from the Palette Window using the  Button.

When it is opened the Editor provides two pages, the Shell page and the Explorer page. You can now open a source file into the Editor. There are several methods for opening a source file in Boa Constructors editor.

- Use the Open Option from the Editor Toolbar or Editor File Menu.
- Use operating system Drag and Drop functionality to drop a file on the editor.
- Use the Explorer Window
- Use an application file
- Start the Boa Constructor application with the source file as a command line parameter.
- Use one of the options from the 'New' page in the Palette.

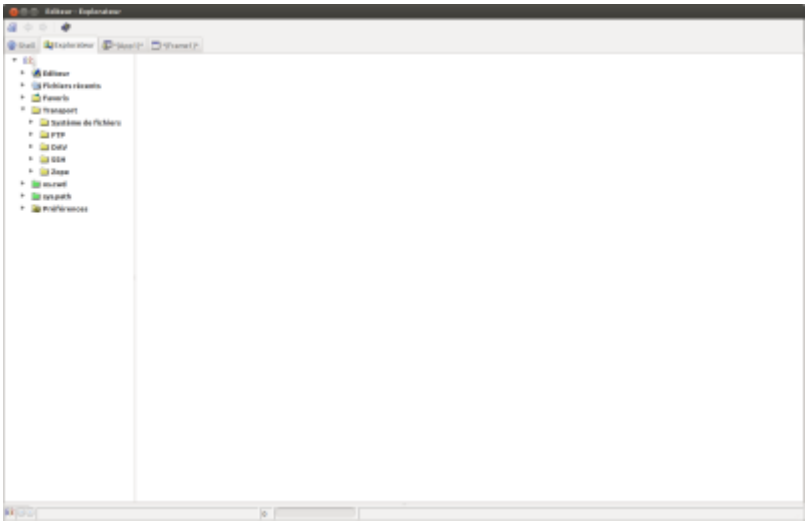


When a source file is opened, the editor creates a new page for the module. Within that page, the editor creates another notebook to provide alternative views of the file, this depends on the type of file. In this case we show an “App1” and “Frame1”, for the Frame1 file three views are shown, 'Source', 'Explore' and 'Events', should you look at the “App1” file you would see 'Application', 'Source' and 'Explore' views.

Note:

The “*” on the “App1” and “Frame1” tab indicate that changes have been made which are not saved.

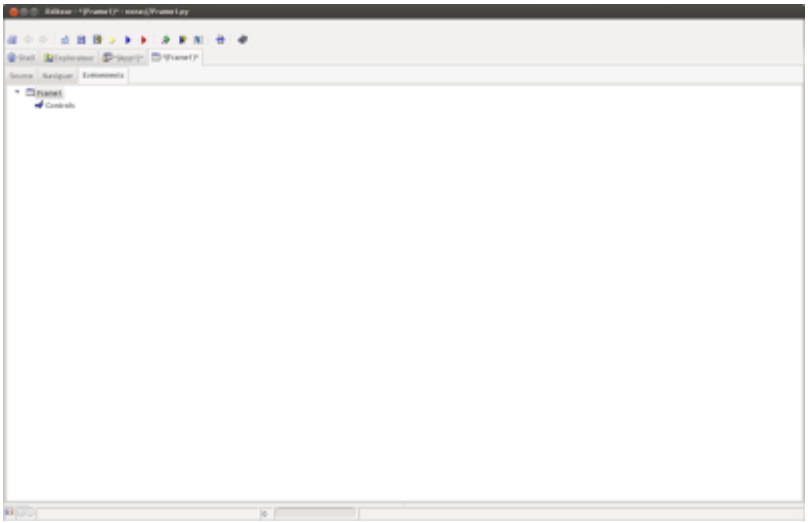
The Source View (seen above) shows the source code and allows you to change the source code, unless it has a “blueish” background which indicates that the Frame Designer is open/active for this file.



The Explore View provides a tree view of the module and the icons shown indicate the following:

- Classes
- Methods
- Functions
- Events
- Global

The 'Explore' Pane shows a Class based view of your source. The methods and attributes of each class in the source are displayed. You can double-click the mouse left button on an attribute or method, and the 'Source' view will display the definition/source code for that item.



The 'Events' Pane shows a Control based view of the Events you defined using Boa's Inspector. Manually added events outside the Boa generated code will not show here.

Additional views can be opened through the Editor/Views menu, this includes such views as “Application ToDo”, “Application Documentation”, “UML” etc, some views will only be available for certain file types.

The 'Views' menu in the editor provides other views on the source. These are Hierarchy, Documentation, ToDo, Application ToDo, Imports, UML, CVS conflicts, Readme.txt, Changes.txt, Todo.txt, Bugs.txt.

- The **Hierarchy View** shows the classes in your source as a hierarchy. The inheritance relationships between classes are clearly visible in this view.
- The **Documentation View** shows documentation, which is automatically produced from your source code. If you use standard python documentation strings for classes and methods (i.e. a string immediately following the declaration), then this information is included in the generated text.
- The **ToDo View** is used for tracking your to-do lists. You add ToDo items to your code as comment followed by three 'X' characters, e.g. “# XXX My todo item”. These are useful to track items that you want to comeback to later.

- The **Application ToDo View** available only for a wx.App type file will show all the files in the project with the number of todo's per file.
- The **UML View** shows the relationships between classes.
- The **Imports View** shows the relationships between modules.
- The **Diff with: View** is created by selecting the menu option File/NDiff files, it will compare the already open file with the one you select.
- The other views should be obvious.

The icons available on the Editor window toolbar for a wxFrame file are:

The icons available on this toolbar change depending on the active file.

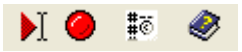
 The icons above allow you to open, close, save, save-as files and Browse forward/backward icons allow you to jump to marked possitions (CTRL-M).

 The icons above let you run the application, run the module or debug the application.

 The icons above let you profile the module, check the source, or start the frame designer (sometimes also refered to as GUI Editor).

 The icons above let you Refresh the screen, Undo, Redo, Cut, Copy and Paste.

 The icons above let you Find/Replace, Find again, and print the source.

 The icons above let you run to the cursor (in debugger), toggle a breakpoint (you can also use F5), insert module information (Author name etc, this can be customized, see Module Info), and help.

Some helpfull short-cuts:

- CTRL+Space will give you code completion, e.g. after entering “wx.F” press CTRL+Space
- CTRL+SHIFT+Space will give me you call tips, e.g. after entering “wx.Frame(” press CTRL+SHIFT+Space
- ALT+O (only available with the ErrOutShortcut plug-in) will show the notebook with the Tracebacks, Output, Errors and Tasks tabs or if it is shown will hide it.

The Frame Designer Windows

You access the Frame Designer Window using the frame designer button  on the Editor Window toolbar. The Frame Designer can only be started if the active file is a Form (wx.Panel, wx.Frame etc) or Dialog type file.

The Designer will display the frame on the screen. The Designer displays the frame as it would appear at runtime. This frame is referred to as the 'Designer'. The Designer also creates two new pages in the Editor. The first one is referred to as the Data View, the second one is the Sizer View.

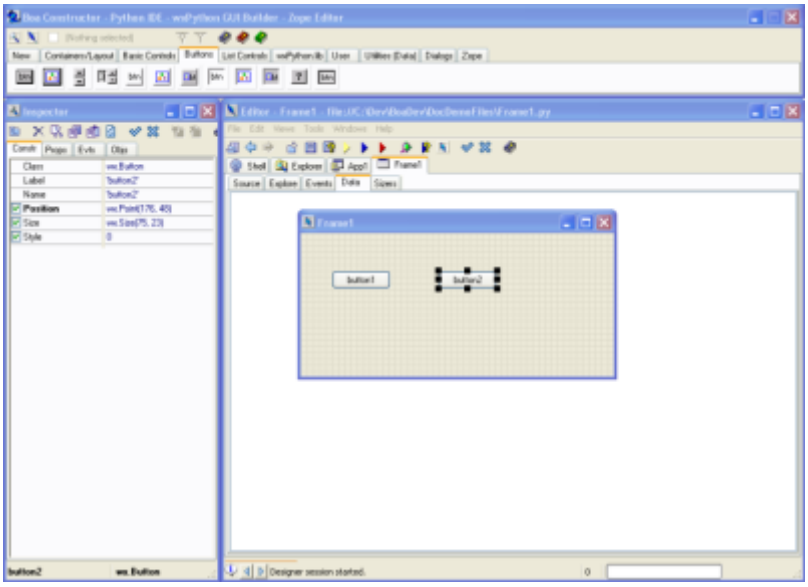
The 'Designer' is used to design the layout of the Frame or Dialog. You can place components in the pane, resize them, move them or delete them. You can also place components within other components.

To create a component, you select the appropriate component from the Palette. Components are grouped for easier access, e.g. the basic controls are together. Once you have selected a control, the status bar in the Palette shows the selected control.

To place the component move the cursor onto the Designer. Click the mouse once in the position where the component is required. Once the component is on the form, it can be moved and resized. You move it by placing the mouse within the component and dragging it. You resize it by dragging one of the eight markers shown on the edge of a selected control.

Changes made through the Designer are saved into the applications source code. Changes are saved when you press the Post button  There are two Post buttons, one on the Editor's toolbar and one on the Inspector's toolbar. Pressing either will close the Designer and the Data View and generate the source code for the changes. Changes can also be posted by closing the Frame.

To Cancel all changes made since opening the Designer, click the Cancel button  on the Editor or Inspector toolbar.

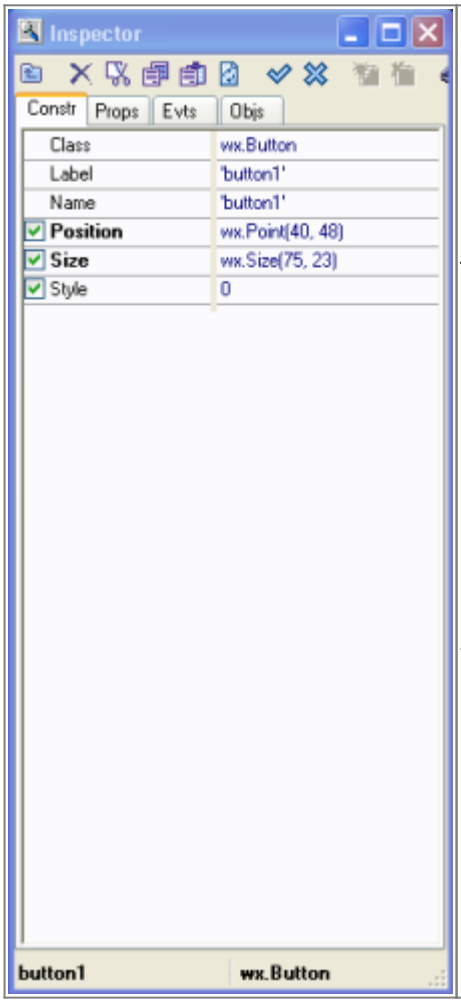


The Frame1 titled window above is the Frame Designer window, a wx.Panel (found on the Containers/Layout palette) has been placed in the wx.Frame and into it two wx.Button controls (found on the Buttons palette) have been placed. The button2 control is currently selected and you can see its properties shown in the Inspector window.

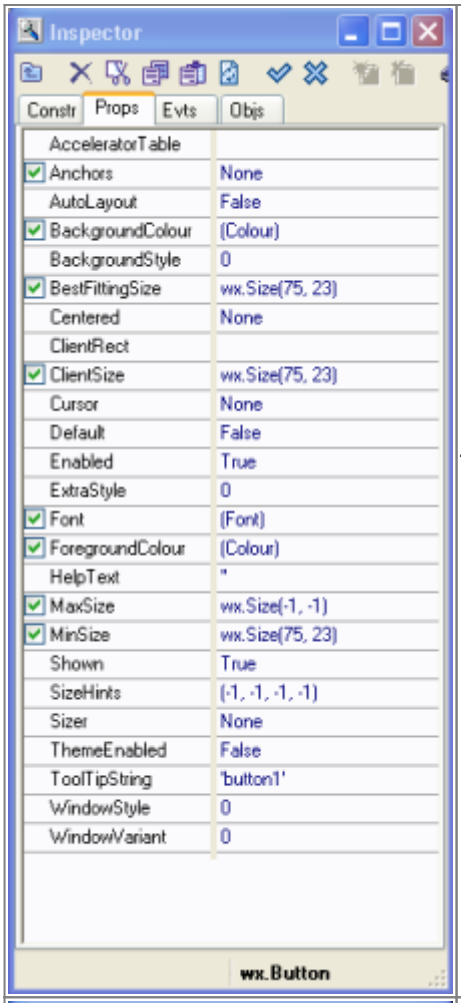
The controls on the Utilities(Data) tab you should drop onto the Data View tab and all sizer type containers (on Containers/Layout tab) you have to drop onto the Sizer View.

The Inspector Window

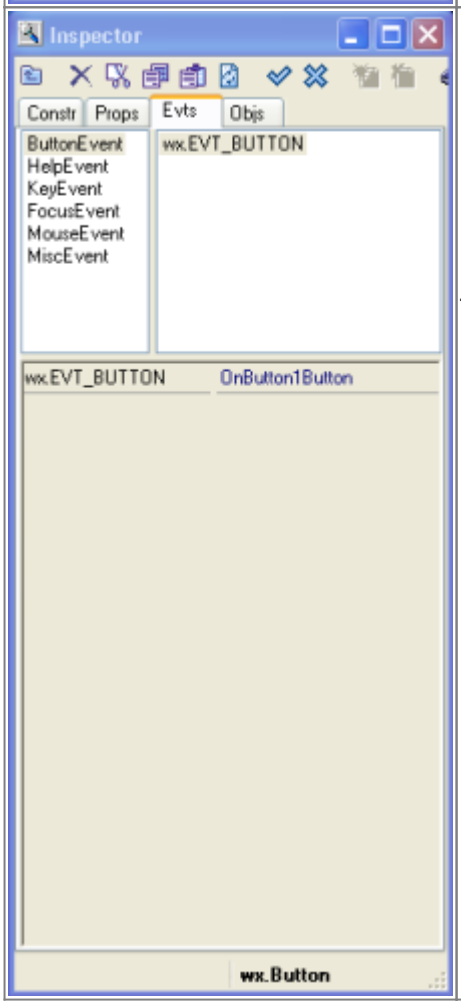
The Inspector always shows the configuration of the currently selected component. The Inspector contains 4 pages, the Constructor page ('Constr'), the Properties page ('Props'), the Events Page ('Evts') and the Objects Page ('Objs').



The Constructor page allows you to edit properties which are required at object construction time, e.g. the name and style of the component.
Look at the wxPython help for the values passed to the constructor for a control. The styles in the manual are of particular importance.
Please note that changes to certain constructor parameters only take effect when the control is created.
You should change the “Name” property to something which will make sense to someone else or to you in 3 month time!



The Properties page gives fine grained control to all properties for this component, e.g. fonts and colours.



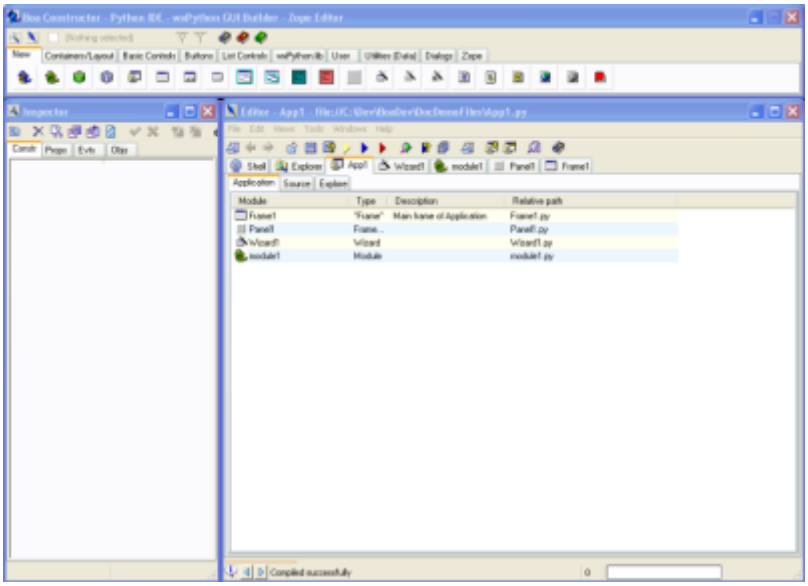
The Events page allows you to select the events which you wish to handle in your code. The events are grouped into logical groups.
Creating an event is easy, just click on e.g. “ButtonEvent” and click on “wx.EVT_BUTTON” and Boa will create an “OnButton” event called “OnControlNameButton”, obviously you will need to add to the generated source what actions you want to take when a user presses this button.

The Objects page allows you to navigate through your components in an alternative format to the Designer pane. This is particularily useful if you have invisible or overlapping components. Also certain components like the wxStatusBar do not process click events (on MSW) so it has to be selected in the Objects page.

Managing the Application

When the module being edited is an application, the Editor provides a special Application View. This Application View allows you to easily track the files in your application, and to add new modules, dialogs and other types of files to your application.

While viewing an Application (any of its views) you may select an item from the 'New' pane in the Palette and it will be added to the Application.



The first of the above icons lets you add an existing file to this application. The second one lets you remove a file from the application.

The Panel1 listed as a module above is NOT the same as the Panel1 contained in Frame1!

Getting around using Explorer

The second page of the Editor notebook contains a page called the Explorer. You use this page to browse various data sources including the filesystem, CVS, Zope, WebDAV, SSH accounts, FTP.

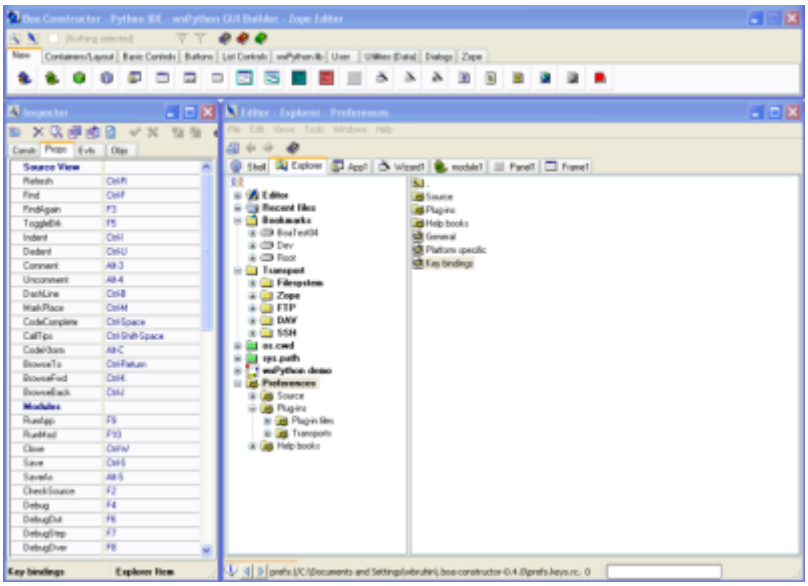
The Explorer shows filesystems and development directories from your operating systems. The first time Boa Constructor is run these filesystems and development directories are added. You can also re-configure where to look see Setting Preferences.

The python directories are taken from the environment variable PYTHONPATH, and the default locations as compiled into your Python runtime system.

Directories that you use a lot can be added to the Bookmarks section by selecting them and clicking on the Bookmark tool  or by using the Edit menu or by right mouse clicking the Directory, note that this item will show next time you start Boa or after you select from the menu Edit/Reload.

If you are using Zope, you can access projects in your Zope Server using the Zope option. The default TCP/IP connection information is configured in the Inspector pane or in your Explorer configuration file, Explorer.msw.cfg on

Windows or Explorer.gtk.cfg on UNIX.



It also allows you to access your Preference settings, above the Key bindings are selected in the Explorer and details are shown in the Inspector.

The wxPython demo shows in the Explorer as it has been activated through the Plug-ins.

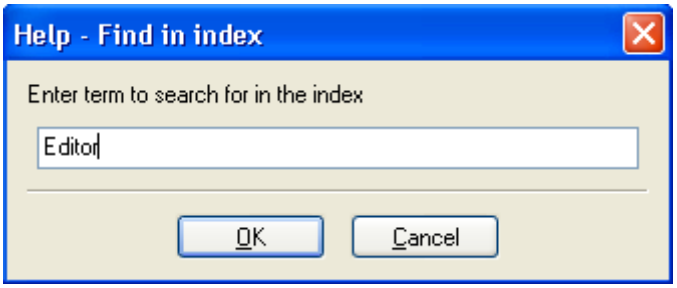
Using Help

The Boa Constructor environment provides links to different help files from your environment.

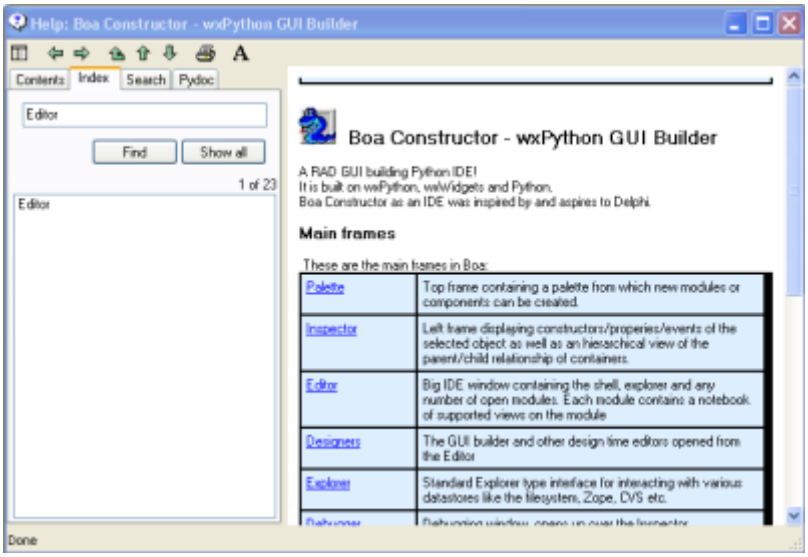
As of Boa 0.6.0 the following help files are included:

- Boa Constructor Help
- Boa Constructor getting Started Guide
- Python 2.5 Documentation
- wxWidgets 2.8.4
- wxPython 2.8.4 API Documenation
- wxStyledTextCtrl Documenation
- Object Graphics Library 3.0

Additional help books can be added (see help books).



The easiest and fastes access to help is just about anywhere in Boa via CTRL-H, then key in what you are looking for and Boa will find all references in ALL the helpbooks it knows about (e.g. wxPython, Python etc) and shows it in a display similar to below.



The help window provides standard HTML based navigation and a search facility.

If the help does not display correctly, you may need to configure your environment.

- The Python or wxPython help may fail to display if the path to the installed Python directory is incorrect. The path to the Python help is in the platform preferences file. This is PrefMSW.py for Windows and PrefGTK.py for UNIX.
- The wxPython help may fail to display if it is not in HTML form. Your wxPython distribution may store their help in Compiled HTML (chm) rather than standard HTML (html/htm). The Boa Contructor tool cannot display compiled HTML. Instead, you should download the HTML version of the documentation from the wxPython site.

Other good places to look for help are:

- The demo files provided with wxPython (depending on the installer you used you might have to download them separately from the wxPython site).

Checkout the Boa plug-in for it, just activate it from Preferences/Plug-ins/Plug-in files/wxPythonDemo and restart Boa

- The wiki pages on wxPython at: http://wiki.wxpython.org/
- There is are a very helpful community out there which will most likely be able to help you.
  - Mail your question to Boa-constructor-users@lists.sourceforge.net if they Boa specific
  - Or mail your question to wxPython-users@lists.wxwidgets.org if they are not Boa specific

Tutoriel - Création de notre première application

Cette section présente un bref tutoriel. Le but de ce tutoriel est de vous familiariser avec l'environnement de développement Boa. Ce tutoriel vous guide étape par étape à travers le processus de construction d'un simple éditeur de texte, appelé Notebook. Après avoir suivi ce tutoriel, vous en saurez pour être efficace avec Boa.

Vous apprendrez à :

- Créer une application.
- Créer des cadres, des menus et des barres d'état.
- Créer des contrôles tels que des boutons, des champs de saisie de texte et des étiquettes.
- Configurer les contrôles selon vos besoins.
- Travailler avec des dialogues classiques.
- Concevoir vos propres boîtes de dialogue.

Création d'une nouvelle application

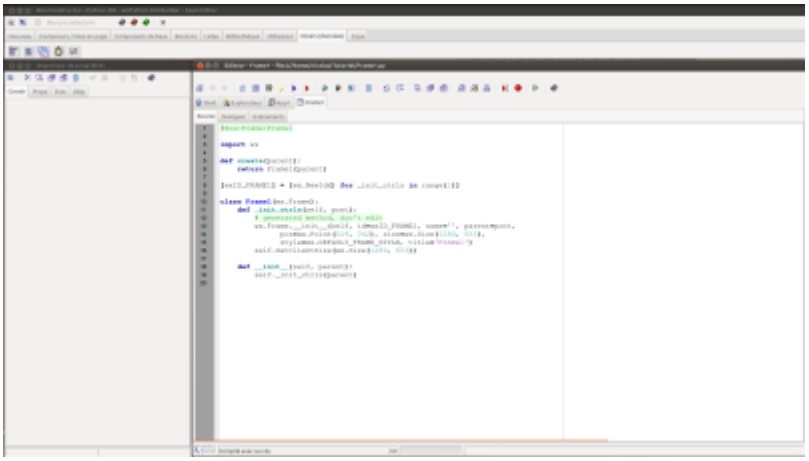
- Choisir un répertoire pour l'application (si nécessaire, le créer)
- Créer une nouvelle application avec le bouton de la palette **Nouvelle l'application** ci-dessous :

 bouton Application - bulle d'aide : **wx.APP**

- Enregistrer les fichiers **App1.py** et **Frame1.py** dans le répertoire créé plus haut. Vous pouvez utiliser le bouton Enregistrer de la barre d'outils de l'éditeur.

Les astérisques (*, indiquant que des modifications du fichier n'ont pas été sauvegardées) disparaissent du nom après l'enregistrement.

- L'application existe, et ne fait qu'afficher un cadre vide. Cliquer sur le bouton Run de la barre d'outils de l'éditeur pour exécuter l'application.

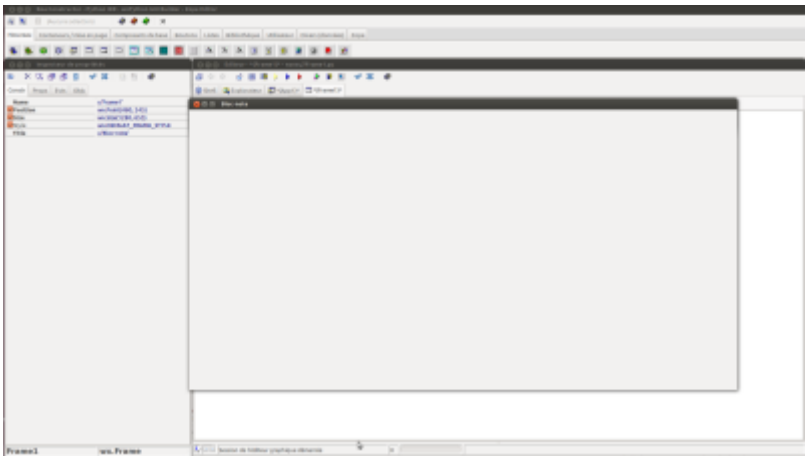


La vue ci-dessus montre dans l'éditeur les deux nouveaux fichiers créés et enregistrés.

 Un clic sur le bouton «Démarrer l'application» (jaune) montre le résultat de la «programmation», c'est à dire juste un cadre vide.

Utilisation de l'éditeur graphique pour définir le titre

- Sélectionnerz l'onglet Frame1 dans l'éditeur pour être sûr d'éditer le cadre.
- Démarrer l'éditeur graphique en cliquant sur le bouton  (éditeur graphique) de la barre d'outils de l'éditeur.



Designer}}

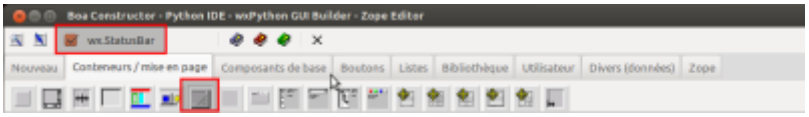
- Le cadre s'affiche comme une zone de dessin (initialement le titre indique «Frame1»
  - Deux nouveaux volets s'affichent dans l'éditeur : le volet «Données» et le volet «Sizers».

2. La fenêtre de l'inspecteur affiche le volet «Constr» (constructeur). Ce volet permet de modifier la taille, la position, le style, un nom de variable et le titre d'un composant. Modifier le champ « Titre ». Donner au cadre le nom « Bloc-Notes » ; quand on appuie sur Retour, le titre de la fenêtre de l'éditeur graphique change.
3. Il faut enregistrer les modifications avec le bouton «Valider» , soit sur la barre d'outils de l'inspecteur ou sur celle de l'éditeur.
4. La fenêtre de l'éditeur graphique se ferme.
5. Noter que le code source a été mis à jour pour refléter le titre.
6. L'éditeur indique que le code source est modifié par une astérisque sur les onglets : il faut donc appuyer sur le bouton Enregistrer.

Ajouter une barre d'état

Le premier composant que nous allons ajouter à l'application sera une barre d'état qui indiquera à l'utilisateur ce qui se passe quand le programme s'exécute, donnera des messages d'aide brefs ou d'autres informations.

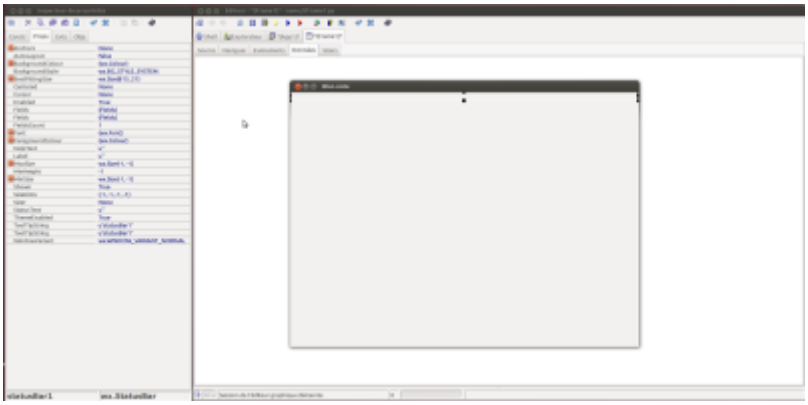
- Sélectionner l'onglet Frame1 dans l'éditeur pour être sûr d'éditer le cadre.
- Démarrer l'éditeur graphique en cliquant sur le bouton éditeur graphique de la barre d'outils de l'éditeur .
- Le cadre s'affiche comme une zone de dessin.
- Dans la palette, sélectionner l'onglet «Conteneurs/mise en page». Cet onglet contient des composants utilisés avec des cadres, composants dont fait partie la barre d'état.
- Survoler les boutons avec la souris. Les bulles d'aide montrent qu'un de ces boutons est le contrôle wx.StatusBar ; c'est celui que nous voulons. Cliquer sur ce bouton.
- Le bouton changer d'apparence pour indiquer qu'il est pressé. Une case à cocher dans la palette indique le type de composant actuellement sélectionné, ici wx.StatusBar.



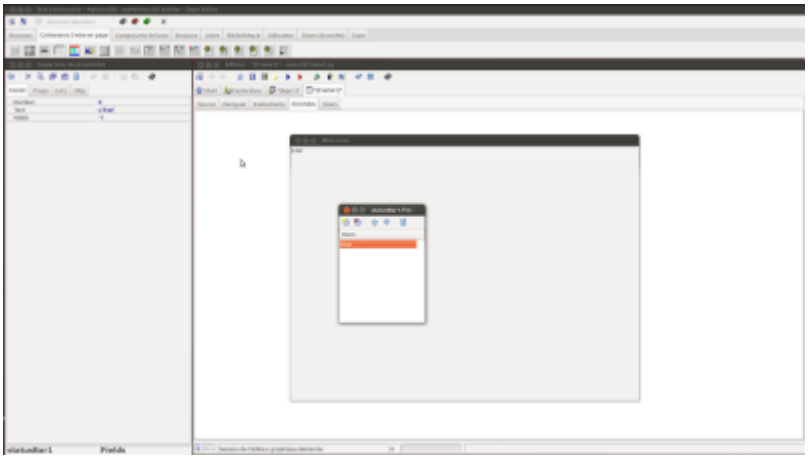
- Déplacer le curseur de la souris sur le cadre de dessin et faire in clic gauche dans la zone de dessin. Cela crée une

barre d'état dans le cadre.

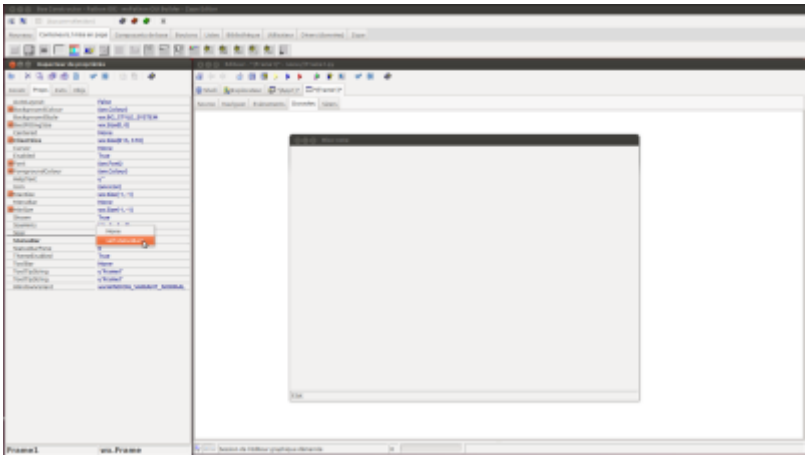
- La barre d'état dans l'inspecteur affiche sur la gauche le nom du widget actuel «statusBar1», et sur la droite, de quelle classe wxWidget elle est dérivée, ici «wx.StatusBar».
- Dans l'inspecteur, sélectionner le volet «Propriétés» qui permet de configurer les propriétés de notre barre d'état.
    - Cliquer sur «Fields» pour modifier sa valeur. Le champ affiche un bouton avec **+++**. Cliquer sur le bouton (s'il n'est pas visible, cliquer sur l'extrémité droite de la case). Cela ouvre l'«éditeur de collection».



- L'outil éditeur de collection permet d'ajouter des sous-composantes à des composants. Nous allons ajouter un champ à la barre d'état [1)].
- Cliquer sur le bouton 'Nouveau'  de l'éditeur de collection. Ceci crée un nouveau champ dans la barre d'état qui devient le champ en cours de l'inspecteur.
- Éditer le champ texte en changeant le nom «Fields0» en «état».
- La barre d'outils de l'éditeur de collection contient un bouton «Rafraîchir»  . Appuyer sur ce bouton pour voir le changement dans la fenêtre de l'éditeur de collection.



- Fermer la fenêtre de l'éditeur de collection. Sélectionner la fenêtre de l'éditeur graphique. Cliquer n'importe où dans la zone de dessin pour faire de cadre (Frame1) le composant courant.
- Sélectionner le volet de propriétés dans l'inspecteur.
- Modifier la propriété 'StatusBar'. Le menu déroulant montre notre nouvelle barre d'état ; la sélectionner pour que le cadre gère la barre d'état, c'est à dire qu'il la positionne en bas de l'écran et ajuste sa taille.



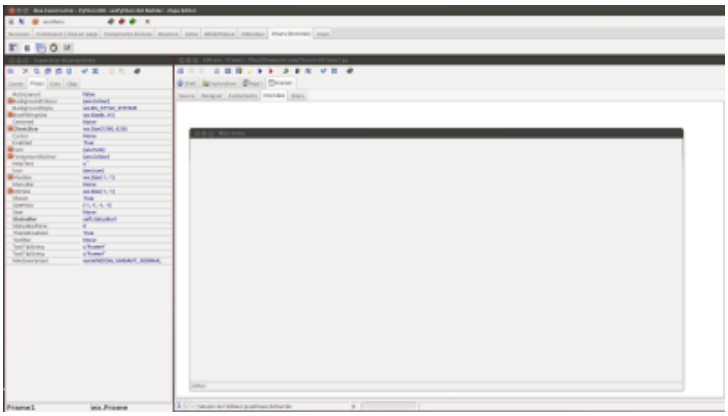
- Mettre à jour les modifications du code source à l'aide des boutons Valider .
- Enregistrer les modifications du code source en utilisant le bouton Enregistrer de la barre d'outils de l'éditeur.

Ajouter une barre de menus

Nous allons maintenant ajouter une barre de menus à l'application. Une barre de menus est un élément courant des programmes Windows. Notre barre de menu contiendra deux entrées, Fichier et Aide. La sélection d'un de ces menus affiche un menu déroulant. L'utilisateur peut sélectionner une option dans le menu déroulant.

- Sélectionner l'onglet Frame1 dans l'éditeur pour être sûr d'éditer le cadre.
- Démarrer l'éditeur graphique en cliquant sur le bouton éditeur graphique sur la barre d'outils de l'éditeur.
    - L'éditeur graphique ajoute deux volets dans la fenêtre de l'Éditeur : «Données» et «Sizer». Dans la palette, sélectionner l'onglet «Divers (données)». Le menu déroulant (wx.Menu) est l'un des composants de cet onglet.

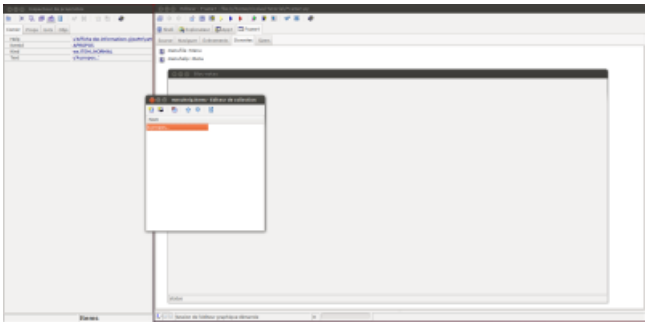
3. Survoler les boutons à la souris. ALes bulles d'aide montrent que l'un de ces boutons représente le contrôle wx.Menu ; c'est celui que nous voulons. Cliquer sur ce bouton.
4. Le bouton changer d'apparence pour indiquer qu'il est pressé. La palette contient une case à cocher qui indique le type de composant actuellement sélectionné, ici wx.Menu.



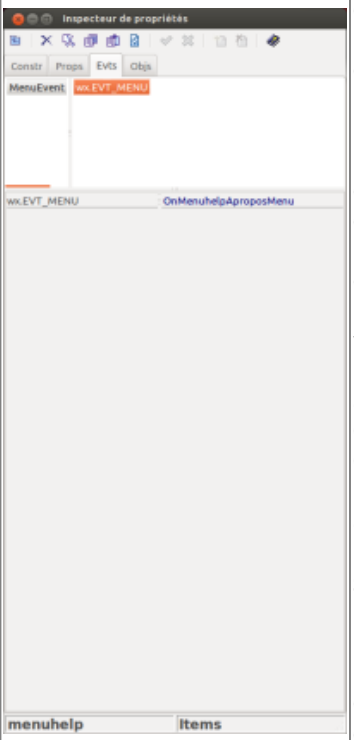
- Faire un clic gauche soit dans l'éditeur, soit dans l'éditeur graphique (attention dans ce cas à cliquer sur la zone Frame1 et non sur la barre d'état).
- Le menu n'est pas visible sur le cadre. Mais il y a désormais une entrée dans l'affichage des données.
- Répéter la procédure pour avoir deux wx.Menus dans la vue de données, appelés menu1 et menu2. Sélectionner menu1 à la souris. L'inspecteur affiche le nom et le titre de menu1.
- Renommer le premier wx.Menu en menufile. Renommer le deuxième wx.Menu en menuHelp. Définir les titres en respectivement Fichier et Aide.
    - Double-cliquer sur l'entrée menuHelp dans la vue de données. Cela ouvre l'éditeur de collection qui va servir à ajouter des éléments à nos menus.

5. Appuyer sur le bouton **Nouveau** de l'éditeur de collection. Cela crée un nouvel élément de menu dans le menu déroulant, qui devient l'élément actif dans l'Inspecteur.
6. Renommer le champ **Label**. de **Items0** en **A propos** ; il est recommandé de renommer ItemId de **ITEMS0** en **A propos**.
7. Renommer le champ **Help String** en **Informations générales sur Bloc-Notes**.

8. La barre d'outils de l'éditeur de collection contient un bouton Refresh. Appuyer sur ce bouton pour voir les changements dans la fenêtre de l'éditeur de collection.



- Dans l'inspecteur, sélectionner le Panneau Événements pour configurer l'action qui se produit lorsque l'élément de menu 'A propos' est sélectionné. Lorsque l'élément de menu 'A propos' est sélectionné, un événement appelé EVT_MENU est généré et envoyé à notre programme. Nous devons ajouter une méthode à notre classe pour gérer cet événement.



- Le côté gauche de la fenêtre des événements montre les groupes d'événements disponibles. Pour menuitem, il n'y a que le groupe 'MenuEvent'. Sélectionner ce groupe.
- Le côté droit de la fenêtre d'événements affiche maintenant les événements dans le groupe sélectionné. Pour le menu, il n'y a qu'un seul événement EVT_MENU dans le groupe «MenuEvent». Double-cliquer sur cet événement.
- Le bas du volet Événements affiche les gestionnaires d'événements dans l'application pour le composant actuel (l'item «A propos» du menu). Il y a maintenant un nouveau gestionnaire appelé OnMenuHelpAboutMenu. C'est le nom de la méthode qui sera appelée lorsque l'option 'A propos' est sélectionnée dans le menu Aide.
- Noter le nommage du gestionnaire d'événements. Boa génère les noms de cette manière. L'événement est la dernière partie (Menu). Le composant est la partie centrale (ici, 'A propos', sous-composante de la composante «menuHelp». Enfin, Boa respecte la convention de préfixer tous les gestionnaires d'événements avec le mot 'On'.
- Fermer l'éditeur de collection.

Maintenant, nous devons répéter le processus pour ajouter des options dans le menu fichier.

- Dans l'affichage données de l'éditeur, double-cliquer sur la rubrique menufile pour ouvrir l'éditeur de collection.
- Ajouter cinq nouveaux éléments.
- Sélectionner chaque élément de menu successivement et les étiqueter 'Open', 'Save', 'Enregistrer sous', 'Close' et 'Exit' ; ici encore, il est recommandé de changer aussi ItemId. Entrer un texte d'aide pour chaque élément de menu. Appuyer sur le bouton de rafraîchissement de l'éditeur de collection pour afficher les nouvelles étiquettes.
- Sélectionner chaque élément de menu à son tour. Pour chaque élément, sélectionner le volet Événements de l'inspecteur, et ajouter un gestionnaire d'événements pour EVT_MENU de chaque élément.
- Close the collection Editor.

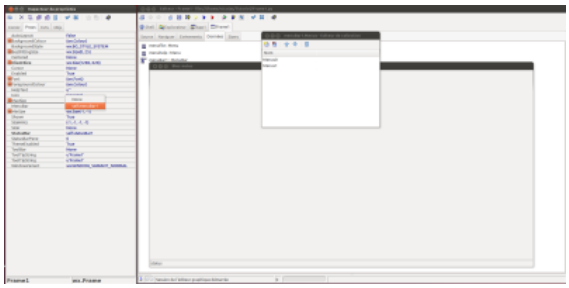
Maintenant, nous allons créer la barre de menus.

- Dans la fenêtre Palette, sélectionner le panneau “Divers (données). Sur ce volet, sélectionner le composant barre de menus.
    - Déplacer le curseur sur la vue Données de l'éditeur. Faire un clic gauche sur ce volet pour ajouter une barre de menu appelé «MenuBar1» à l'application.
    - Faire un double-clic sur l'élément MenuBar1 pour ouvrir son éditeur de collection (comme le montre l'image ci-dessous, on peut en garder plusieurs ouverts).

2. Ajouter deux éléments à la barre de menus à l'aide de l'éditeur de collection. Sélectionner Menus0 dans le volet constructeur de l'inspecteur et modifier le champ 'Menu'. C'est un menu pop-up avec trois éléments, le constructeur wx.Menu() et nos deux menus déroulants. Sélectionner l'élément self.menuFile et définir le titre de 'fichier'. Cela fait du menu menufile le premier menu

déroulant de la barre de menus.

3. Dans l'éditeur de collection, sélectionner le deuxième élément. Répéter le processus pour lier Menus1 au menu d'aide déroulant, avec l'étiquette «Aide».
4. Sélectionner le cadre principal, Frame1 sur le concepteur. Le cadre est maintenant le contrôle actif de l'inspecteur.
5. Sélectionner le volet Propriétés («Props») dans l'Inspecteur. Modifiez le champ 'MenuBar. C'est un menu pop-up. Sélectionner la nouvelle barre de menu self.menuBar1. Cette propriété définit le menuBar à associer au cadre.



- Enregistrer les modifications à l'aide d'un bouton Valider pour fermer le concepteur et laisser Boa générer le code source.
- Enregistrer le code source du votre fichier source généré Frame1.py
- Exécuter le programme.
  - On voit maintenant les menus et la barre d'état (dans ubuntu, les menus sont affichés en haut du bureau).
  - Quand on sélectionne une option de menu, le texte d'aide apparaît dans la barre d'état.

Ajout du contrôle texte

La tâche suivante consiste à ajouter du texte principal contrôle d'édition à notre cadre. Ce contrôle s'appelle wx.TextCtrl.

- Rouvrir l'éditeur graphique pour modifier le cadre, Frame1.py.
- Dans la palette, sélectionner le panneau “Contrôles de base”. Sélectionner wx.TextControl.

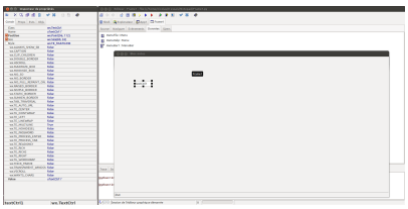
Survoler chaque commande pour connaître son nom.

- Déplacer la souris sur la fenêtre du concepteur, vérifier qu'une bulle d'aide affiche «Frame1», puis faire un clic gauche. Un nouveau contrôle de texte est affiché. Nous n'avons pas à redimensionner le contrôle. Par défaut, il remplit toute la surface disponible, c'est à dire entre la barre d'état et la barre de menus.
- par défaut, wx.TextControl est une entrée à ligne unique. Nous devons indiquer au contrôle que nous voulons que ce soit un contrôle de saisie multi-ligne. Pour cela, éditons la propriété 'style' de l'inspecteur, sur le volet constructeur.
- Modifier le style et le définir à wx.TE_MULTILINE. On peut taper ceci dans le champ de valeur pour le style, ou cliquer sur la case à gauche de style : Boa montre tous les

styles disponibles. Définir les styles à utiliser à «Vrai» en cliquant dessus.

- Le champ style contient du code Python valide. Pour définir deux styles logiques, les séparer par un '|'. On peut voir tous les styles disponibles pour wx.TextControl dans l'aide en ligne wxPython pour wx.TextControl.

Utiliser Ctrl+H et entrer 'textctrl' pour obtenir de la documentation et des descriptions des différents styles, noter que certains peuvent ne pas être montrés pour wx.TextCtrl car ils sont hérités, par exemple wx.Window.



- Renommer le champ texte. Le nom par défaut est «textCtrl1». Le renommer en «TextEditor».
- Dans le volet constructeur, un champ appelé «valeur» contient la valeur par défaut du contrôle.

Mettre ce champ à vide.

- Mettre à jour le code source avec le nouveau contrôle à l'aide d'un des boutons Valider
- Enregistrer les modifications du code source.
- Exécuter l'application.

- Le champ éditeur de texte est automatiquement redimensionné à l'espace disponible.
- Si on redimensionne le cadre, le contrôle se redimensionne.
- Remarquer que wxWidgets offre une barre de défilement. Le champ défile automatiquement si on dépasse le bas. Si on tape une ligne plus longue que la largeur de la fenêtre d'édition, elle se replie
- Les fonctionnalités couper et coller sont également disponibles par défaut, ainsi que le marquage de bloc.

Ajout des fonctionnalités du Menu Fichier

La tâche suivante consiste à

interagir avec l'utilisateur pour implémenter les fonctionnalités des menu. Des boîtes de dialogue sont utilisées pour saisir l'entrée de l'utilisateur. Ces boîtes de dialogue sont modales, c'est à dire qu'on ne pas utiliser les autres fenêtres de l'application en cours jusqu'à ce que le dialogue soit fermé.

- Les boîtes de dialogue sont placées directement dans le code source. Elles ne sont pas placées dans l'éditeur graphique mais avec l'éditeur. Dans le code source Frame1.py, aller dans le gestionnaire d'événements pour l'événement open. Cette méthode est appelée OnMenuFileOpenMenu. Nous allons placer le dialogue «Ouvrir un fichier» dans cette méthode. Placez le curseur directement devant 'event.Skip()'. Nous allons placer ici notre nouveau code.
- Appuyerz sur Alt+T et sélectionner 'wx.FileDialog' dans le menu déroulant et Boa va coller le squelette du code dans la méthode de gestion d'événements.
- Noter le code 'dlg.Destroy()', il est très important que les boîtes de dialogue soient détruites!
- Le code source doit maintenant se

présenter ainsi :

```
def OnMenuFileOpenMenu(self, event):
  dlg = wx.FileDialog(self, "Choose a file", ".", "", "*.*", wx.OPEN)
  try:
    if dlg.ShowModal() == wx.ID_OK:
      filename = dlg.GetPath()
      # Your code
  finally:
    dlg.Destroy()
  event.Skip()
```

- Ce code crée un squelette de dialogue qui interagit avec l'utilisateur. Quand le dialogue est terminé, il est détruit.
- Les mots '# Your code' marquent l'endroit où insérer le code. Ce code est déclenché dans l'état wx.ID_OK, c'est à dire quand l'utilisateur a cliqué sur le bouton «Ouvrir». Nous allons insérer notre code ici. wx.TextCtrl a une méthode que nous allons utiliser pour charger un fichier dans la fenêtre d'édition, la méthode «LoadFile».
- On peut supprimer event.Skip() car aucun autre événement ne devra être appelé dans ce cas. Le nom «event.Skip()» est un peu trompeur, on devrait appeler

«event.Skip()» quand d'autres gestionnaires d'événements doibent aussi être exécutés.
    - Il a été nécessaire lorsque le code a été généré parce que Python signale une erreur s'il ya une méthode sans corps.

4. Parmi les fonctionnalités de notre application, nous devons pouvoir accéder au nom de fichier pour que l'option de menu “Enregistrer” puisse sauvegarder ce fichier, donc nous avons ajouté la ligne 'self.FileName = filename'.
5. La ligne 'self.SetTitle(('Notebook - %s') % filename)' change le titre pour montrer sur quoi on travaille.
6. The listing below shows our new code.

```
def OnMenuFileOpenMenu(self, event):
  dlg = wx.FileDialog(self, "Choose a file", ".", "", "*.*", wx.OPEN)
  try:
    if dlg.ShowModal() == wx.ID_OK:
      filename = dlg.GetPath()
      # Your code
self.textEditor.LoadFile(filename)
```

```
self.FileName=filename
self.SetTitle(('Notebook - %s') % filename)
  finally:
    dlg.Destroy()
```

- Répéter l'exercice pour fournir la fonctionnalité «Enregistrer sous». Insérer une boîte de dialogue de fichier dans le corps de «OnFileSaveasMenu».
- C'est une boîte de dialogue Enregistrer. Changer l'invite, paramètre 2 de wx.FileDialog en «Enregistrer sous». Changer le style, le paramètre 6 en wx.SAVE. La méthode pour enregistrer le fichier s'appelle SaveFile.
- Encore une fois, nous sauvons la valeur de nom de fichier utilisé par l'option de menu “Enregistrer”.
- Le listing ci-dessous montre le code.

```
def OnMenuFileSaveasMenu(self, event):
    dlg = wx.FileDialog(sel
```

```
f, "Save file
as", ".", "",
"*.*", wx.SAVE)
    try:
        if
dlg.ShowModal()
== wx.ID_OK:
filename =
dlg.GetPath()
            #
Your code
self.textEditor.S
aveFile(filename)
self.FileName=fil
ename
self.SetTitle(('N
otebook - %s') %
filename)
    finally:
dlg.Destroy()
```

- Ensuite, nous allons mettre en œuvre l'option de menu 'Fermer'. Dans cette méthode, nous effaçons simplement le contrôle éditeur de texte, la variable membre de nom de fichier et réinitialisons le titre..

```
def
OnMenuFileCloseMe
nu(self, event):
    self.FileName
= None
self.textEditor.C
lear()
self.SetTitle('No
tebook')
```

- Maintenant, mettons en œuvre l'option de menu “Quitter”. Dans cette

méthode, nous devons mettre fin à l'application. Toutes les demandes de wxPython sont terminées par la fermeture de la fenêtre de niveau supérieur. Dans notre cas, nous n'avons que la fenêtre Frame1. Pour mettre fin à l'application, nous invoquons la méthode Close() pour Frame1.

```
def OnMenuFileExitMenu(self, event):
self.Close()
```

- Ensuite, mettons en œuvre l'option de menu “Enregistrer”. Cet élément de menu enregistre le fichier en utilisant le nom actuel, qui est stocké dans la variable de self.FileName.
- Quand il n'y a pas de fichier en cours, la variable self.FileName vaut None. Dans ce cas, l'option de menu «Save» agit comme l'option de menu “Enregistrer sous”.
- La variable FileName doit être créée

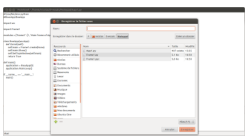
lorsque Frame1 est construit. Nous devons l'ajouter au constructeur. On peut ajouter le code d'application à la fin du constructeur par défaut (init).

```
def __init__(self, parent):
self._init_ctrls(parent)
self.FileName=None
```

- Maintenant, nous pouvons implémenter la fonctionnalité Enregistrer. Nous vérifions s'il y a un nom de fichier courant. S'il existe, nous pouvons enregistrer le contenu de ce fichier. Sinon, il suffit d'appeler la méthode «Enregistrer sous».

```
def OnMenuFileSaveMenu(self, event):
    if self.FileName == None:
        return self.OnFileSaveasMenu(event)
    else:
self.textEditor.SaveFile(self.FileName)
```

- Nous avons maintenant mis en œuvre toutes les fonctionnalités de l'éditeur. Nous pouvons ouvrir des fichiers, les modifier et les sauvegarder.
- L'éditeur doit ressembler à ce qui est montré sur l'image ci-dessous.
    - Le fichier App1.py a été ouvert
    - Puis, l'option de menu “Fichier/Enregistrer sous” a été sélectionnée



Créer une fenêtre de dialogue

Les boîtes de dialogue permettent d'interagir avec l'utilisateur et de récupérer les valeurs entrées. Précédemment, nous avons utilisé la boîte de dialogue pré-construite

wx.FileDialog ; nous allons maintenant développer notre propre boîte de dialogue pour l'option de menu **A propos**.

- Ce nouveau dialogue nécessite une nouvelle fenêtre, qui n'est pas un composant de la fenêtre Frame1 et qui aura donc son propre fichier Python séparé. Dans l'éditeur, sélectionner le module **App1**, volet **Application**.
- Dans la palette, dans le volet **Nouveau**, sélectionner le bouton **wx.Dialog**. Cela crée un nouveau fichier source **Dialog1.py** et ajoute automatiquement ce fichier

source à notre module application.

- Sélectionner le volet Frame1, écrire le code de l'option **A propos** du menu, qui affiche la boîte de dialogue, avec la méthode **OnHelpAboutMenu**. Le code est le suivant :

```
def OnMenuHelpAboutMenu(self, event):
    dlg = Dialog1.Dialog1(self)
    try:
dlg.ShowModal()
finally:
dlg.Destroy()
```

- Ce code fait référence au module Dialog1, qu'il faut donc importer pour que ce code

fonctionne. Par convention, les importations se placent au début du code source. Dans **Frame1.py**, ajouter la ligne

```
import Dialog1
```

après la ligne

```
import wx
```

comme ceci :

```
import wx
import Dialog1
```

- Enregistrer les trois fichiers sources. On peut maintenant lancer l'application : quand on sélectionne l'option **A propos** du menu **Aide**, la nouvelle boîte de dialogue apparaît.

La boîte de

dialogue est modale, c'est à dire qu'il faut la fermer avant de pouvoir accéder à la fenêtre Frame1

Quitter l'application et revenir à Boa.

- Nous allons maintenant ajouter des champs à la boîte de dialogue. Pour cet exercice, nous utilisons un fichier bitmap appelé **Boa.jpg**. On peut aussi en créer un à l'aide d'un utilitaire comme paint. Copier le bitmap dans le répertoire de l'application .
- Sélectionner le volet **Dialog1.py** . Démarrer

l'éditeur graphique en cliquant sur le bouton  .

- Nous allons d'abord ajouter une étiquette à la boîte de dialogue. Dans la palette, volet **Composants de base**, sélectionner le contrôle **wx.StaticText**. Cliquer dans l'éditeur graphique pour créer le contrôle.
- Dans l'inspecteur, mettre la valeur du champ **Label** à “Bloc-Notes - Simple éditeur de texte”. Notez que l'étiquette s'agrandit dans l'éditeur graphique pour accueillir le texte.
- Pour configurer l'alignement du texte dans

l'étiquette, définir la propriété style à **wx.ALIGN_CENTRE** ou la sélectionner après avoir cliqué sur la case à gauche du style.

- Dans l'inspecteur, panneau **Propriétés**, modifier le champ **font** pour définir une assez grande police, par exemple 12 ou 14 points. Noter qu'on peut modifier à la fois la police et la taille en point avec cette propriété.
- Dans la fenêtre de l'éditeur graphique, l'étiquette s'affiche avec huit marqueurs sur les bords. Pour redimensionner la boîte cliquer et

faire glisser l'un de ces marqueurs. Pour déplacer l'étiquette, cliquer au centre de l'étiquette, et glisser pour placer l'étiquette en haut au milieu de la boîte.

- Ajouter une autre étiquette sous la première, avec le texte “Ceci est ma première application avec Boa Constructor ”. Dans l'inspecteur , panneau **Propriétés** , modifier la valeur **BackgroundColour**. Choisir une couleur dans celles proposées et appuyer sur OK. Repositionner et redimensionner l'étiquette jusqu'à ce que l'ensemble soit équilibré.

- Nous allons maintenant ajouter le bitmap. Dans les **Composants de base**, sélectionner le contrôle **wx.StaticBitmap**. Le placer sous la seconde étiquette du dialogue. Dans l'inspecteur, volet **constructeur**, modifier le champ **Bitmap**. Cela ouvre une boîte de dialogue “Ouvrir le fichier” : choisir le bitmap préparé plus tôt. Dans la fenêtre de l'éditeur graphique, le champ **wx.StaticBitmap** change pour accueillir le bitmap. Le déplacer le bitmap jusqu'à ce qu'il soit équilibré sous les

deux étiquettes.

- Enfin, nous allons ajouter un bouton à la boîte de dialogue. Dans la palette, panneau Boutons, sélectionner le type de bouton de base, **wx.Button**. Le placer dans le formulaire sous le bitmap. Dans l'inspecteur, volet constructeur donner à “Label” la valeur “Fermer”. Dans le volet Événements de l'inspecteur, ajouter un gestionnaire pour le type d'événement EVT_BUTTON.

sélectionner d'abord le groupe d'événements, puis l'événemen

t. 

- Ce sont tous les contrôles que nous ajoutons à la boîte de dialogue. Redimensionner de la boîte de dialogue, repositionner et redimensionner les contrôles jusqu'à ce que tout soit bien équilibré.
- Sélectionner Dialog1 dans l'éditeur graphique. Dans le volet constructeur de l'inspecteur, modifier le champ “field et lui donner la valeur “À propos de Bloc-Notes”.
- Appuyez sur un des boutons Valider pour mettre à jour le code

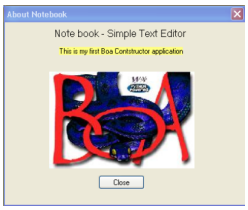
source.

- Il nous reste à mettre en œuvre le gestionnaire d'événements pour le bouton Fermer. Dans l'éditeur, sélectionner le code source de Dialog1. Allez à la méthode 'OnButton1Button'. Nous allons utiliser la même méthode “Close” que nous avons utilisée dans le menu **Quitter**. Noter que cela ferme la fenêtre. Fermer la fenêtre principale sort de l'application. Tandis que fermer une fenêtre enfant revient simplement à la fenêtre parent.

```
def OnButton1Button(
```

```
self, event): self.Close()
```

Lancer l'application. Le nouveau dialogue devrait ressembler à ceci :



Félicitations: Vous avez construit votre première application en utilisant Boa constructeur. Votre éditeur est terminé. Dans ce tutoriel, vous avez utilisé les composants de base de Boa.

Prenez le temps de revoir ce que vous avez fait jusqu'ici. Vous avez appris à :

- Créer une application.
- Créer des cadres, des menus et des barres d'état.
- Créer des contrôles

tels que des boutons, des champs de saisie de texte et des étiquettes.
- Configurer les contrôles selon les besoins.
- Travailler avec des dialogues pré-définis.
- Concevoir vos propres boîtes de dialogue.

Creating an application window using sizers

Sizers are a great way to ensure that your GUI layout is nice and clean. They come in especially handy when you do not know exactly how much space a control needs and/or should be allowed to use, this can be the case when you internationalize your application (I18N) or for such controls as lists or grids where you like to give as much space as possible to them

(or maybe as little as practical).

Please note that the following will just explain how to use sizers in Boa (note that this assumes version 0.6.x of Boa). For more detailed information about sizers you should check the wxPython documentation, the wxPython demo and you might also find the following links helpful (if not a must!) to understand sizers.

- http://wiki.wxpython.org/index.cgi/UsingSizers
- http://wiki.wxpython.org/index.cgi/LearnSizers1
- http://wiki.wxpython.org/index.cgi/wxDesigner_20Sizer_20Tutorial

We will use a wx.Frame and create a screen for address information entry.

- Close all the source files in your editor, so not to add this to the application you created previously.
- On the palette, select the 'New' pane. Select the 'wx.Frame' button. This will create a new source file *(Frame1)*.
- Click on the Save button (or menu File/Save) and save it as AddressEntry.py.
- Select from the menu Edit the option Add module runner. This will add some code to your file so you can run it without having to have a separate wx.App file.
- Save the file and you can run this application, you will see just Frame1

in the title bar and a grey background .

- Select the AddressEntry pane. Start the Designer by clicking on the button  .
- On the palette, select the 'Containers/Layout' pane. Click on the wx.Panel button to select it and click anywhere within the AddressEntry frame. This will drop the panel onto your frame.
- On the same palette pane click on the wx.BoxSizer button to select it and click anywhere on the wx.Panel you just added to your frame. You should see a

yellow line around your panel.

- Post these changes and save the file and re-open the Designer.
- On the sizer pane, click on the boxSizer1 and rename it to e.g. bsMain.
- Bring the designer to the foreground (e.g. just click on the Designer tool bar button).
- Select the wx.ListCtrl control on the “List Controls” pane and drop it onto the Designer, this will automagically add it to the bsMain sizer.
- On the “Containers /Layout” pane select the wx.FlexGridSizer and drop it also onto the wx.Panel, which again

will automagically add it to the bsMain sizer.

- Click on the “Sizers” pane and select the flexGridSizer1 and rename it to e.g. fgsFields.
- In the Inspector change the 'Cols' setting from '0' to '2' and the 'Rows' setting from '1' to '0', as we will have to columns of controls/widgets in this sizer.
- Post the changes, save the file and open the Designer again. I do this quit regurarely to ensure that I don't loose too much of my work if something should go wrong. It is also a good idea to just

run the application to see how it looks.

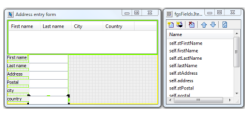
- In the Inspector change the Name from 'Frame1' to 'AddressEntry' and the Title from 'Frame1' to 'Address entry form'.
- Select the wx.ListCtrl in the Designer and change the style from wx.LC_ICON to wx.LC_REPORT and on the “Props” pane click on (Columns) and then on the ”…” to open the Collection Editor for the listctrl. Create the columns “First name, Last name, City and Country”.
- Click on the “Sizers” pane and double click on bsMain to open it's Collection

Editor. Then click on the self.listCtrl1 and change the Border from 0 to 2 (or what you find appropriate for a border around this control) and change Flag from 0 to wx.ALL | wx.EXPAND and change Proportion from 0 to 1. These changes will ensure that you have 2 pixels space around the listctrl and that it will use up as much space as is available. If you run this little application now you will see that the listctrl takes up all the available space.

- On the “Sizers” pane open the Collection

Editor for the fgsFields sizer and add 12 new items, when you now look at the Designer it will show these items in red.

- From the “Basic Controls” Palette pane select the wx.StaticText control and drop it onto the top left red area and to the right of it drop a wx.TextCtrl and then repeat this until your Designer screen and the fgsFields collection editor look something along these lines.



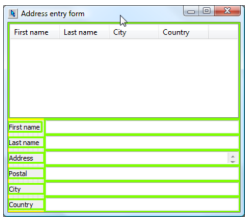
- Make sure to rename the controls to names which make sense (i.e.

firstName, lastName, address, postalCode, city and country).

- Now we need to set the Border, Flag and Proportion for each of these controls.
- For wx.StaticText I suggest: 2, wx.ALL | wx.ALIGN_CENTER_VERTICAL and 0
- For wx.TextCtrl I suggest: 2, wx.ALL | wx.EXPAND and 0
- On the “Sizers” pane you need to select the fgsFields sizer and make the second column growable which you can do from the Inspector “Props” pane by clicking on the “…” next to “(Growable

s)".

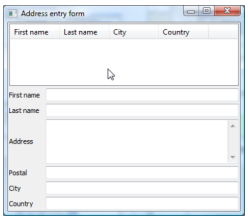
- And for this to take any effect you need to change the Flag for the fgsFields sizer in the bsMain sizer from 0 to wx.EXPAND .
- So, now you should see something like this in the Designer.



- If you run it at this point and resize the window you can see the sizers at work.
- You might also notice that you see scrollbars on the Address field and it is larger then other fields. For this you need to change it's style from 0

to wx.TE_MULTILINE and in the Designer you enlarge it to the hight you want to allocate for it.

- When you run the application you should see something along these lines.

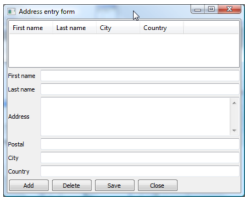


- We will also need some buttons for this, so we can add, delete, save and close this form.
- For this open the Designer again and drop another flexGridSizer (I will name it fgsButtons) onto the “Sizers” tab and then add it to

the bsMain sizer.

- Then add four items to the fgsButtons sizer and then drop wx.Button controls onto the red squares on the designer.
- In the sizer Collection Editor change the Border to 2, the Flag to wx.ALL for all these buttons.
- Then select the first button by double clicking its entry in the Collection Editor and in the Inspector “Constr” pane change the label from button1 to “” (blank) and the name from button1 to “add” and the Id to wx.ID_ADD.
- Repeat this for the others but name them delete,

save and close and use the appropriate wx.ID_ entries (having access to the stock button ID's is new in Boa 0.6.0, it will only work if you blank the label.)



- You should now see something like the above when you run it.
- Obviously you only have the GUI code at this point and one would have to flesh all this with code for each of the buttons, but for the moment this goes beyond this tutorial.

Please note that the file generated

during this example is also available in the directory “Examples\guide” under your Boa installation directory.

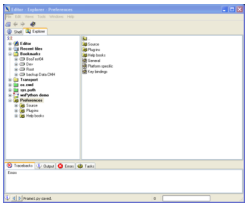
For coding guide lines you might also want to consult the wxPython style guide http://wiki.wxpython.org/index.cgi/wxPython_Style_Guide.

Other Useful Items

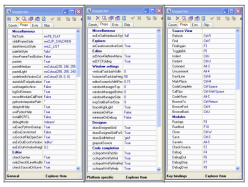
Setting Preferences

The Boa Constructor tool provides a number of features which can be customised by you.

Most of the customization settings can be set by using the Editor Explorer View. Click on Preferences and it shows you something similar to the image below.



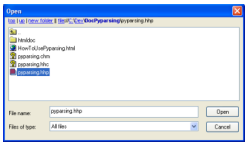
To change settings double click on either 'General', 'Platform specific' or 'Key bindings' and the properties of each will be shown in the Inspector as shown below:



Help books

Boa by default includs its help books and the ones for wxPython and Python.

However if you like to add others you can do so by selecting Preferences/Help Books and right mouse click in the right pane of the Explorer. Select 'Add new Item' and browse to find the '.hpp' file.

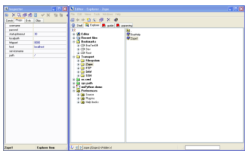


Bookmarks

If you like to add additional bookmarks just right mouse click on the folder you like to add within the Explorer.

Transport

Adding additional transports works similar to the bookmarks, select the transport type on the left hand side of the Explorer view, e.g. 'Zope' and then right mouse click in the right hand side and select 'New' and then complete the information in the Inspector.



Module Info

You might want to change the following section in the file 'prefs.rc' stored in your user preference

directory, on a Windows system this is by default in 'driveletter:\Documents and Settings\username\.boa-constructor'.

```
# Info that will be filled into the comment block. (Edit->Add module info)
# Also used by setup.py
staticInfoPrefs = {
'Purpose': '',
'Author': '<your name>',
'Copyright': '(c) 2004',
'Licence': '<your licence>',
'Email': '<your email>',
}
```

- Replace '<your name>' with guess what your name.
- Change the '© 2004'
- Replace '<your license>' with something

along these lines 'Shareware - see license.txt for details'

- Replace '<your email>' with the email you want to use

If you like to edit the 'CustomModuleInfo.plug-in' file right mouse clicking the corresponding entry in 'Preferences/Plug-ins/Plug-in files' and select 'Open Plug-in file'.

If you don't want to loose your changes when you upgrade Boa you might want to copy the 'CustomModuleInfo.plug-in' file and call it 'MyCustomModuleInfo.plug-in' in the Plug-ins directory. After restarting Boa you should disable the standard one by right mouse clicking and selecting the appropriate option.

I changed it as follows:

```
"""
Demonstrates how to change system constants as a plug-in
"""

import sourceconst
# The order of (Name)s may change and lines may also be removed
sourceconst.defInfoBlock = '''# -*- coding: iso-8859-1 -*-#
#-----------------------------------------------------------------------
# Name: %(Name)s
# Purpose: %(Purpose)s
#
# Author: %(Author)s
#
# Created: %(Created)s
# RCS-ID: %(RCS-ID)s
# Copyright: %(Copyright)s
# Licence: %(Licence)s
#-----------------------------------------------------------
```

```
------------
------------
------------
------------
------
'''

import
Preferences
# (Name)s
not in the
original
dictionary
needs to be
added
# New field:
%(NewField)s
#Preferences
.staticInfoP
refs['NewFie
ld'] =
'Whatever'
```

- Added the '# -*- coding: iso-8859-1 -*-#'
- Moved the 'New field:'
- Commented the 'Preferences.staticInfoPrefs' line as I don't need it

After restarting Boa and clicking on the button  Boa will insert the following into the selected file (in this case Frame1.py).

```
# -*-
coding:
iso-8859-1 -
```

```
*-#
#----------
----------
----------
----------
----------
----------
------
# Name:
Frame1.py
# Purpose:
#
# Author:
Werner F.
Bruhin
#
# Created:
2005/12/03
# RCS-ID:
$Id:
node31.html,
v 1.1.2.1
2005/03/14
09:23:09
wbruhin Exp
$
# Copyright:
(c) 2004 -
2005
# Licence:
Shareware,
see
license.txt
for details
#----------
----------
----------
----------
----------
----------
------
```

1)

On peut ajouter plusieurs champs