Eurogam project **EDOC116**

Developpement logiciel

DEVGUIDE: a graphical interface editor

---

edition : 1.0
date : February1991
auteurs: L. Michel G. Zehnacker

Centre Recherches Nucleaires Strasbourg
CNRS-IN2P3 France

Abstract

This document describe the abilities of the OpenWindow Developper's Guide (*DevGuide*).

The first section describes the abilities of *DevGuide* , the second section describes how generate C code from a *DevGuide* file. The third section gives some suggestions for an easy use of Devguide in the Eurogam project.

1 The usage of DevGuide

1.1 What does make DevGuide ?

DevGuide is used to create user interfaces that comform to the *OPENLOOK* User Interface as it is implemented in OpenWindow. *DevGuide* is a Graphical User Interface Editor.

The graphical interface is composed with the standard *XView* objects set. The objects are manipulated (size, position, ...) directly by the mouse from the *DevGuide* window to the screen model. After to be put on the screen model, the objects parameters are set by specific dialog boxes (event filter, graphicals options, callback routine name,...).

The screen model is stored as an ASCII file (*.G* extension) describing each *XView* object. The file format is specific to *DevGuide* .

The screen model can be dynamically tested under the control of *DevGuide* (open menus, edit text,....). !some objects can be connected to a callback routine which name is user's defined.

It's possible to associate with the model an help text, edited under the control of *DevGuide* , what will be called automatically by the "help key" when the application will be runing.

1.2 What doesn't make DevGuide ?

DevGuide doesn't generate C or C++ code. The parameters defining the objects are static, except the label of certains objects class which can be set from a file.

DevGuide doesn't create new object class. It doesn't manipulate user's defined objects. May be user's defined objects can be inserted in the storage file initially created by *DevGuide* . We haven't verified this ability.

1.3 The ".G" file

The window created by *DevGuide* are stored in an ASCII file named *xxxx.G* where *xxxx* is the application name.

In this file, each object is described by its name, its geometric characteristics, its callback routine name and some configuration flags. The file can be easily modified by a text editor.

The file is written by *DevGuide* for store an interface, it's read by it for edit an old interface.

The C code is created by a compiler (*gxv*) from the *.G* file. The following example show the definition of a button in a *.G* file.

```
(
:type            :button
:name            Bouton1
:owner           Controle-area1
:help            ””
:x               170
:y               62
:constant-width nil
:button-type     :normal
:width           58
:heigth          20
:foreground-color”darkblue”
:label           ”quitte”
:label-type      :string
:menu            nil
:notify-handler quittecallbackroutine
:event-handler  nil
:events          ()
:user-data       ()
)
```

2 The ”gxv” compiler: the C code generation

2.1 what does make the ”gxv” compiler ?

The *gxv* compiler creates C files from a *.G* file created by *DevGuide* . The generated source code uses the *XView* toolkit. They make the creation of the graphics objects, the configuration of the XEvent main loop and the entry point of the callback routines. A makefile is also created to compile the set of source files.

An application is described by one *appl.G* file from which the *gxv* compiler writes four source files:

- *appl_ui.h*	definition of the specifics data structures.
- *appl_ui.c*	creation of the graphics objects.
- *appl_stubs.c*	mainloop and callback routines.
- *Makefile*	file used for manage the compilation.

A version of *gxv* (*gxv* +) create C++ file, but we don't u se it actually.

2.2 The "appl_ui.h" file

The *appl_ui.h* file is a classical header file in C langage. It contains the prototypes of the functions which create the Xviews objects and the definition of a data structure pointing on this objects.

2.3 The "appl_ui.c"file

The *appl_ui.c* file regroups all the functions which create and set up the *XView* objects. This file depends just on the screen model edited by *DevGuide* . It must not be modified while t he *appl.G* is not modified.

2.4 The "appl_stubs.c" file

The *appl_stubs.c* file contains the main function and the callback routines. The main function calls the routines of creation and initialization of the objects and starts the main loop. The callback routines have their names defined in the *appl.G* file. They just send a message on the screen, on "stderr stream", when are called. This message writes the name of the routine and the statement of the control object if necessary. This is very useful to test the graphical interface before write the user's code.

The user's code must be inserted in these routines. Each modification of the interface using the *appl.G* file requires a new compilation by *gxv* , that creates a new set of source files and t hen deletes the user's code in *appl_stubs.c* . We propose a method to avoid this restriction of the use of *DevGuide* in the section 3.

2.5 The "Makefile"

The compilation of the *application uses the make* utility. *Gxv* create a make file named *Makefile* .

3 How make a modifiable interface by "DevGuide" ?

3.1 Classical cycle for the creation of an application.

The standard cycle to create an *application with an graphical interface by DevGuide* is the following:

- Edit the graphical interface by *DevGuide*

- Test the graphical interface under control of *DevGuide* .

- Create a *appl.G* file describing the interface, where *appl* is the name of the application.
- Compile the *appl.G* file to write the set of source code files.
- Make the executable file of the interface.
- Test the interface, and make a new cycle if some modifications are necessary.
- Insert the user code in the *appl_stubs.c* file.
- Compile the entire application.

This way, it's difficult to modify the graphical interface after the insertion of the user code in the *appl_stubs.c* file, because the use of *gxv* erases it. Then the programmer must rewrite his routines in the *appl_stubs.c* file.

3.2 Cycle for the creation of an adaptable application.

The following method results of the personnal test of devGuide done in Strabourg.

After a modification of the *appl.G* file, *gxv* doesn't write a new makefile. So it's possible to modify it by inserting sources filename containing the code of the user's routines. This routines are called from the modified callback routines created by *gxv* .

The graphical objects are defined in the *appl_ui.c* . So after a modification of the *appl.G* file, just *appl_ui.c* will be changed. If new callback routines are created, they must be added to the *appl_stubs.c* .

Then, for keep the user's code in the *appl_stubs.c* file it's possible to change its name (example: *mv appl_stubs.c appl_kern.c*) with the appropriarte modification in *Makefile* , so the new *appl_stubs.c* will be ignored at the compilation. In this case, the new callback routines written by it gxv (empty routines) must be added to the *appl_kern.c* file.

The new cycle to create an application with an graphical interface by *DevGuide* is the following:

- Edit the graphical interface by *DevGuide*
- Test the graphical interface under control of *DevGuide* .
- Create a *appl.G* file describing the interface, where *appl* ,is the name of the application.
- Compile the *appl.G* file to write the set of source code files.

- Make the executable file of the interface.
- Test the interface, and make a new cycle if some modifications are necessary.
- Insert the user code in the *appl_stubs.c* and change its name.
- Create if necessary the user's source files.
- Modify the *Makefile* .
- Compile the entire *application.*
- *Insert in the appl_kern.c* (former *appl_stubs.c*) the new callback routine if necessary.

3.3 Modification of the graphical objects

Each object is created (call *xv_create*) with its attributs set by *DevGuide* . The *XView* toolkit permits to change dynamically the values of the attributs (call *xv_set*). Then it's possible to change the control labels or the menu item list without use *DevGuide* . It's also possible to create dynamically new control objects if they handler routines exist. But, in this case, the new object will be not included in the data structure created by *gxv* .

3.4 Name of the variables describing the graphical objects

Gxv creates a data structure describing the graphical object. Each field of this stucture pointes an object out. The name of each field is the same as the name of the object. Then, , it's easy to point an object out in the user's code without read the data structure. The programmer needs just the name of the objects defined by *DevGuide* .

Example: How point a button out ?

name of the application: *Appl*
name of the base window*Base*
name of the button: *Quitte*
name of the button identifier*Appl_Base-¿Quitte*

3.5 Proposition for the syntax

It can be interesting to use an homogeneous syntax for name the graphic objects. We Propose 2 rules for name it:

- Each object must have a name corresponding to its function. This name is written with lowercase characters except for the 1st letter of each word. Example: name of a control area which command a zoom: *ZoomControl* . This syntax is homogeneous with the *atoms* of X11.

- The name of the objets must not use the "_" character, because it's used by *gxv* to merge the variable name.

Objects must not have long name because *gxv* merge they for name the variables.