GIM

A LaTeX like language for building Graphical Interactive Manuals

– User's guide and reference manual –

– Release 1.0 –

Serge CHAUMETTE

Abstract

GIM (**G**raphical **I**nteractive **M**anual) is a language plus a set of tools that make easier the task of building on-line manuals for graphical applications. Graphical interactive manuals are described using a LaTeX like language that enables to produce text, icons, pieces of interfaces, etc. The resulting manuals can be installed as stand-alone commands or integrated within **X Toolkit** or **Wcl/Motif** applications.

This document is designed to be both a user's guide and a reference manual for **GIM**. It also provides many illustrative examples.

Résumé

GIM (**G**raphical **I**nteractive **M**anual) est un langage et un ensemble d'outils qui facilitent la construction de manuels graphiques et interactifs. Ces manuels graphiques interactifs sont décrits à l'aide d'un langage à la LaTeX qui permet de produire du texte, des icônes, des éléments d'interfaces, etc. Les manuels résultants peuvent être installés en tant que commandes, ou intégrés dans des applications **X Toolkit** ou **Wcl/Motif**.

Ce document est conçu à la fois comme un manuel utilisateur et comme un manuel de référence pour **GIM**. Il propose aussi de nombreux exemples.

Contents

List of Figures

List of Tables

List of Programs

GIM

1 Introduction

When building the graphical user interface of an application, one must also build some sort of on-line manual. It is often the case that this manual is of poor quality compared to the interface of the application itself.

The reason for that is that it is as difficult and time consuming to develop a good quality manual as it is to write a real interface.

GIM (**G**raphical **I**nteractive **M**anual) is a step towards solving this problem. It provides a LaTeX[2] like language, which is used to describe a graphical interactive manual. This graphical manual is composed of pages which can contain icons, text, and sensitive items such as links between pages, pieces of graphical interfaces, etc.

The resulting graphical interactive manuals can be integrated within **X Toolkit**[3] or **Wcl/Motif**[1, 4] applications. They can also be installed as stand alone manuals.

More than a language, **GIM** is a package composed of a set of tools, comprising:

- a LaTeX like language, called **GIM**;
- a compiler for this language (`gim`);
- a tool to test, i.e. to see, the manuals being developed (`gimtest`);
- a library for integration of a **GIM** manual within applications (`libxgim.a`).

2 A first look at a GIM manual

Figure 1 presents a simple manual generated by **GIM**. Moving between pages and closing the manual is automatically handled (note the arrows and the "close" button on top of each manual page). The **GIM** source of this manual is given below (see program 1).

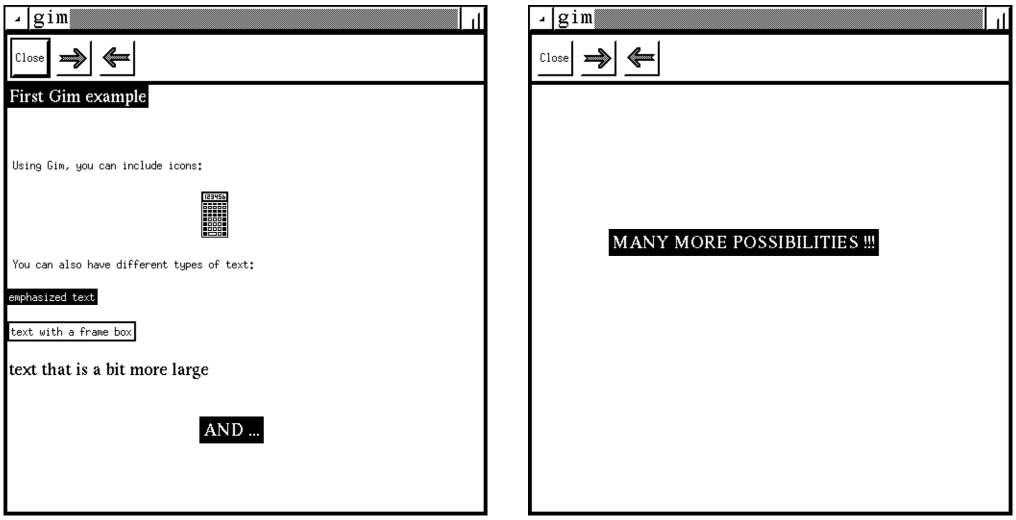


Figure 1: A simple **GIM** manual

Program 1: Source file of a simple GIM manual

```
# This is a sample gim source file

\pagesize{500}{500}

\begin{manual}

\em\large
First Gim example

\newline\newline\newline\newline

# Show an icon

\icon{arrow.bm} Using Gim, you can include icons:

\newline\newline

\hspace{200}\icon{calculator}

\newline\newline
```

```
# Show different kinds of text

\icon{arrow.bm} You can also have different types of text:

\newline\newline
\em emphasized text

\newline\newline

\fbox text with a frame box
\newline\newline

\large text that is a bit more large

\newline\newline

# What is not shown in this example

\vspace{20}\hspace{200}\em\large\fbox AND ...

\newpage

\vspace{150}\hspace{80}\em\large\fbox MANY MORE POSSIBILITIES !!!

\end{manual}
```

3 What you must know about your system before beginning

Read this section and proceed as indicated. If needed, require assistance from your system administrator.

$\boxed{1.}$ Answer the following questions:

1. In which directory are **GIM** commands ?
   ....
2. Which is the **GIM** library directory ?
   ....
3. Which is the local binaries directory ?
   ....

4. Which are the local and user specific applications defaults (app-defaults) directories ?
   ....
   ....
   ....
   ....

5. Which are the local and user specific bitmaps directories ?
   ....
   ....
   ....
   ....

2. Report the above answers in the following table:

1. gim-bin-dir	
2. gim-lib-dir	
3. local-bin-dir	
4. app-default-dir	
5. bitmaps-dir	

Table 1: Summary of system configuration

Throughout this document, we will refer to these directories using the symbolic names that appear in the first column of table 1. This means that `<bitmaps-dir>`, for instance, will stand for one of the names of the directories that appear in the corresponding row(s).

4 The GIM language

The overall structure of a **GIM** program is presented figure 2. It is composed of a set of initializations that consist in setting defaults, followed by the body of the manual. This body is composed of pages themselves composed of line. Each line is in turn composed of line elements.

defaults setting

manual body

`\begin{manual}`

page 1

line 1
`\newline`
line 2
`\newline`
...
`\newline`
line n

`\newpage`

page 2

line 1
`\newline`
line 2
`\newline`
...
`\newline`
line n

`\newpage`

page n

line 1
`\newline`
line 2
`\newline`
...
`\newline`
line n

`\end{manual}`

Figure 2: Overall structure of a **GIM** manual

4.1 Setting defaults

Defaults are parameters that will be applied to all pages of the manual being described.

Setting page size

syntax: \pagesize{<width>}{<height>}

result: all pages will be <width> pixels wide and <height> pixels heigh.

4.2 Structuring a manual

Structuring a manual consists in putting delimiters around its body and breaking it into pages and lines.

Beginning the body of a manual

syntax: \begin{manual}

result: starts up the body of the manual.

Finishing the body of a manual

syntax: \end{manual}

result: finishes the current page and line and terminates the body of the manual.

Breaking pages

syntax: \newpage

result: finishes the current line and page.

Breaking lines

syntax: \newline

result: finishes the current line.

4.3 Producing line elements

There are several types of line elements that can be produced: icons, text, and pieces of interfaces.

Text

syntax: <text>

result: produces <text>

example:

```
This is a sample text on
two lines.
```

result:

This is a sample text
on two lines.

† Please note that the text is not formatted. It is displayed as is, and line breaks are respected. Two different line elements of type text are separated by a blank line or a line element of another type than text.

Icons

syntax: \icon{<bitmap-file-name>}

result: produces the icon which bitmap description is stored in <bitmap-file-name>.

example: \icon{dragon.bm}

result: 

Non popup pieces of interfaces

Non popup pieces of interfaces are a combination of **Motif** widgets[1] that do not require creation of a new window to show up. This combination is expressed using the extensions of **Wcl** for resources specifications. Please refer to **Wcl** documentation[4] for full details.

syntax: \widget{<widget_name>}{<widget_description>}

result: the described piece of interface is inserted.

example:

```
This a menu : \widget{manual_menu}{
*manual_menu.wcConstructor:                XmCreateMenuBar
*manual_menu.wcChildren:                   menu_pull, menu_button
*manual_menu*menu_button.wcConstructor: XmCreateCascadeButton
*manual_menu*menu_button.labelString:   A menu with 2 entries
*manual_menu*menu_button.subMenuId:     *menu_pull
*manual_menu*menu_pull.wcConstructor:   XmCreatePulldownMenu
*manual_menu*menu_pull.wcManaged:       False
*manual_menu*menu_pull.wcChildren:      entry_1, entry_2
*manual_menu*entry_1.wcConstructor:     XmCreatePushButton
*manual_menu*entry_1.labelString:       Entry 1
*manual_menu*entry_1.mnemonic:          1
*manual_menu*entry_1.accelerator:       Ctrl<Key>1
*manual_menu*entry_1.acceleratorText:   Ctrl 1
*manual_menu*entry_2.wcConstructor:     XmCreatePushButton
*manual_menu*entry_2.labelString:       Entry 2
*manual_menu*entry_2.mnemonic:          2
*manual_menu*entry_2.accelerator:       Ctrl<Key>2
*manual_menu*entry_2.acceleratorText:   Ctrl 2
} Click to pull it down !!!
```

result:

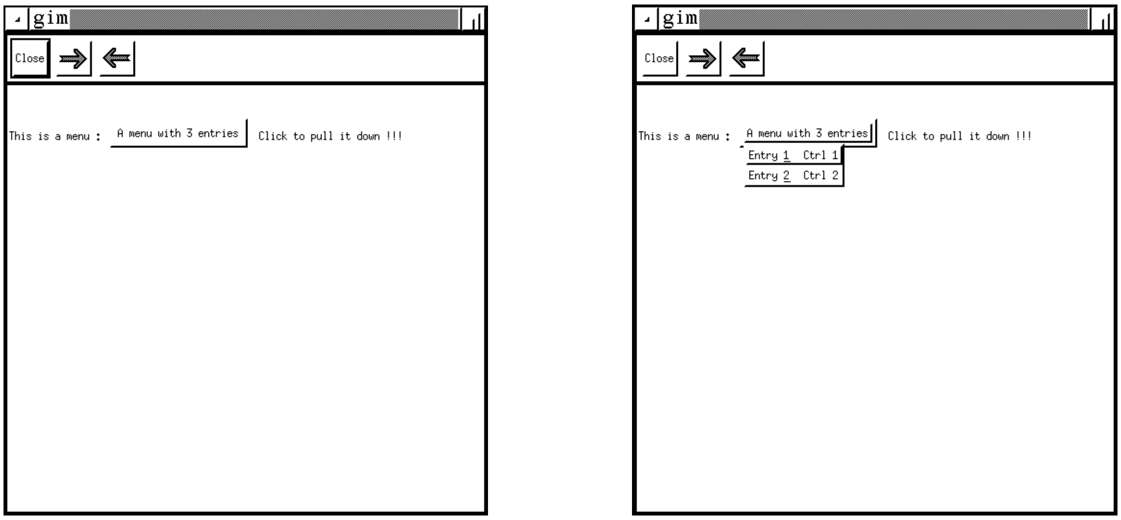


Figure 3: A non popup widget within a manual

Popup pieces of interfaces

Popup pieces of interfaces are a combination of **Motif** widgets[1] that require creation of a new window to show up. This combination is expressed using the extensions of **Wcl** for resources specifications. Please refer to **Wcl** documentation[4] for full details.

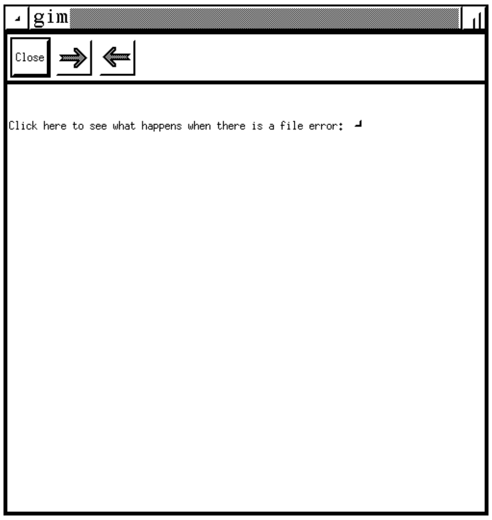
syntax: `\popupwidget{<widget_name>}{<widget_description>}`

result: an icon is displayed, which when clicked on pops up the described piece of interface.

example:

```
Click here to see what happens when there is a file error:
\popupwidget{file_err}{
*file_err.wcConstructor: XmCreateErrorDialog
*file_err.messageString: The file you selected is not valid !!!
}
```

result:



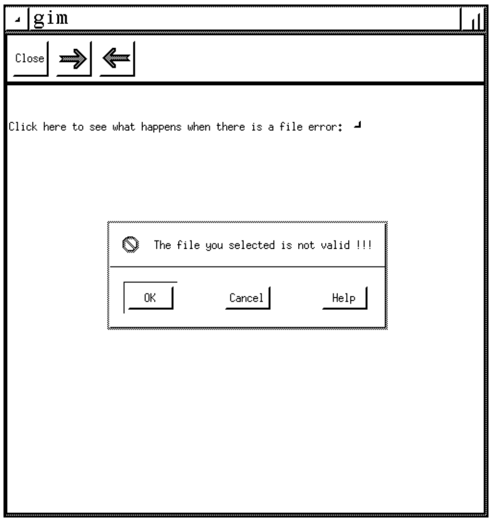


Figure 4: A popup widget within a manual

4.4 Making line elements look different

Making a line element look different consists in modifying some of its graphical characteristics.

Emphasizing a line element

syntax: `\em`

result: the next line element is emphasized.

example: `\em This is a sample text.`

result: This is a sample text.

Enlarging a line element

syntax: `\large`

result: the next line element is enlarged.

† `\large` is only significant if next line element is of type text.

example: `\large This is a sample text.`

result: This is a sample text.

Putting a line element within a box

syntax: `\fbox`

result: the next line element is drawn within a box.

example: `\fbox This is a sample text.`

result: $\boxed{\text{This is a sample text.}}$

4.5 Spacing

Spacing commands are used to put space between line elements and to align them. All spacing values are expressed in pixels.

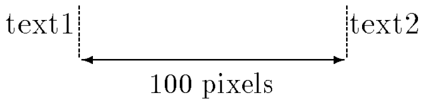
Horizontal spacing

syntax: `\hspace{<pixels>}`

result: produces a horizontal space of size `<pixels>` pixels before next line element.

example: `text1\hspace{100}text2`

result:



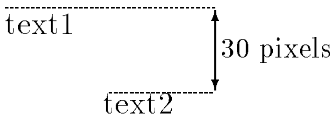
Vertical spacing

syntax: `\vspace{<pixels>}`

result: produces a vertical space of size `<pixels>` pixels before next line element.

example: `text1\vspace{30}text2`

result:



Putting a line element within an invisible box

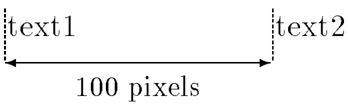
syntax: `\makebox{<pixels>}`

result: draw next line element flushed left within an invisible box of size `<pixels>` pixels wide.

example: `\makebox{100}text1`

`text2`

result:



4.6 Making links between pages

It is sometimes useful, when reading one of the pages of an on-line manual, to jump directly to another page. To handle this, you can produce a small icon (▣) that when clicked on with the mouse will cause the manual to jump to a given page.

Label

syntax:	`\label{<label-name>}`
result:	marks the current page as possibly referenced using label `<label-name>`.
example:	`This page is important.\label{importantpage}`
result:	This page is important.

Reference

syntax:	`\ref{<label-name>}`
result:	produces this icon: ▣. When clicked on it will cause the manual to jump to the page where `<label-name>` is defined.
example:	`Jump to important page: \ref{importantpage}`
result:	Jump to important page: ▣

5 Testing a GIM manual

Once you have written a **GIM** manual, you can test it, i.e. see what it looks like. To do so, proceed as follows. Note that steps (2) and (3) are only required if you are using icons of your own, the bitmap descriptions, i.e. the `.bm` files, of which are in directories <dir_1>, <dir_2>, ... <dir_n>.

  (1) `$ gim <gim_src> -o <gim_output> -t`
! (2) `$ export XBMLANGPATH`
! (3) `$ XBMLANGPATH=`<dir_1>`/%B:`<dir_2>`/%B:...:`<dir_n>`/%B`
  (4) `$ gimtest <gim_output>`

When you run `gimtest`, an additional menu is created so that you can "open" your manual and "quit" the test.

6 Setting up a stand-alone GIM manual

A stand-alone manual is a **GIM** manual that is accessed as is, i.e. outside of any application. It is a command in itself. Below, this command is referred to as `<manual_name>`. To setup such a manual, proceed as follows. Note that steps (4) and (5) are only required if you are using icons of your own, the bitmap descriptions of which are stored in files <icon$_1$.bm>, <icon$_2$.bm>, ... <icon$_n$.bm>.

  (1) $ gim <gim_src> -o <gim_output> -s <manual_name>
  (2) $ cp <gim_output> <app-default-dir>/<cfl_manual_name>
  (3) $ cp <gim-bin-dir>/gimalone <manual_name>
! (4) $ mkdir <bitmaps-dir>/<cfl_manual_name>
! (5) $ cp <icon$_1$.bm> <icon$_2$.bm> ... <icon$_n$.bm>
            <bitmaps-dir>/<cfl_manual_name>/

where `<cfl_manual_name>` stands for `<manual_name>` where you have capitalized the first letter.

You can now access your stand-alone manual by typing in:

  (1) $ <manual_name>

7 Integrating a GIM manual within X Toolkit applications

To use a **GIM** manual within your **X Toolkit** application, proceed according to the four following steps:

Step 1. Compile the GIM manual

(1) $ gim <gim_src> -o <gim_output> -x

Step 2. Modify your X Toolkit application

- in your code:
  - declare:
    extern Widget gim;
  - call xgim_setup(Widget w) just before realizing you own Widget tree. The Widget gim is then a (created) popup of your Widget w. You can now on use it as if it were a popup you created yourself.
- when linking you code, add the following libraries:
  -lxgim -lWc -lXmp -lXm -lXt -lX11

Step 3. Integrate resources

Two cases are to be considered:

- if you are using a resource file just merge <gim_output> with it;
- if you are not using a resource file proceed as follows:

```
(1) $ cp <gim_output> <app-default-dir>/<cfl_binary_name>
```

  where <cfl_binary_name> stands for the name of your executable program where you have capitalized the first letter.

Step 4. Integrate icons

Two cases are to be considered:

- if you are using icons of you own which are not included within your source code, i.e. external .bm files, make a copy of <gim-lib-dir>/*.bm into the directory containing these files;
- otherwise, proceed as follows:

```
(1) $ mkdir <bitmaps-dir>/<cfl_binary_name>
(2) $ cp <gim-lib-dir>/*.bm <bitmaps-dir>/<cfl_binary_name>/
```

  where <cfl_binary_name> stands for the name of your executable program where you have capitalized the first letter.

8 Integrating a GIM manual within Wcl/Motif interfaces

To use a **GIM** manual within your **Wcl/Motif** interface, proceed according to the three following steps:

Step 1. Compile the GIM manual

```
(1) $ gim <gim_src> -o <gim_output> -w
```

Step 2. Integrate resources

First make a copy of <gim_output> into the same directory as your **Wcl** resources file(s) and add the following line to your main (or unique) **Wcl** file:

- if your **Wcl** release is at least 2.5:

  ```
  *.wclResFiles:  <gim_output>
  ```

- for older **Wcl** releases:

  ```
  *gim.wcResFile:  <gim_output>
  ```

You can now on use the Widget gim as a popup you would have created yourself.

Step 3. Integrate icons

Two cases are to be considered:

- if you are using icons of you own which are not included within your source code, i.e. external .bm files, make a copy of <gim-lib-dir>/*.bm into the directory containing these files;

- otherwise, proceed as follows:

  ```
  (1) $ mkdir <bitmaps-dir>/<cfl_binary_name>
  (2) $ cp <gim-lib-dir>/*.bm <bitmaps-dir>/<cfl_binary_name>/
  ```

where <cfl_binary_name> stands for MRi if you just use the **Wcl** Mri interpreter, or the name of your executable program where you have capitalized the first letter.

9 A complete GIM manual

In this section we present an example that illustrates the use of all features of the **GIM** language, but pieces of interfaces.

Figures 5, 6 and 7 show the resulting manual and program 2 is the source file.

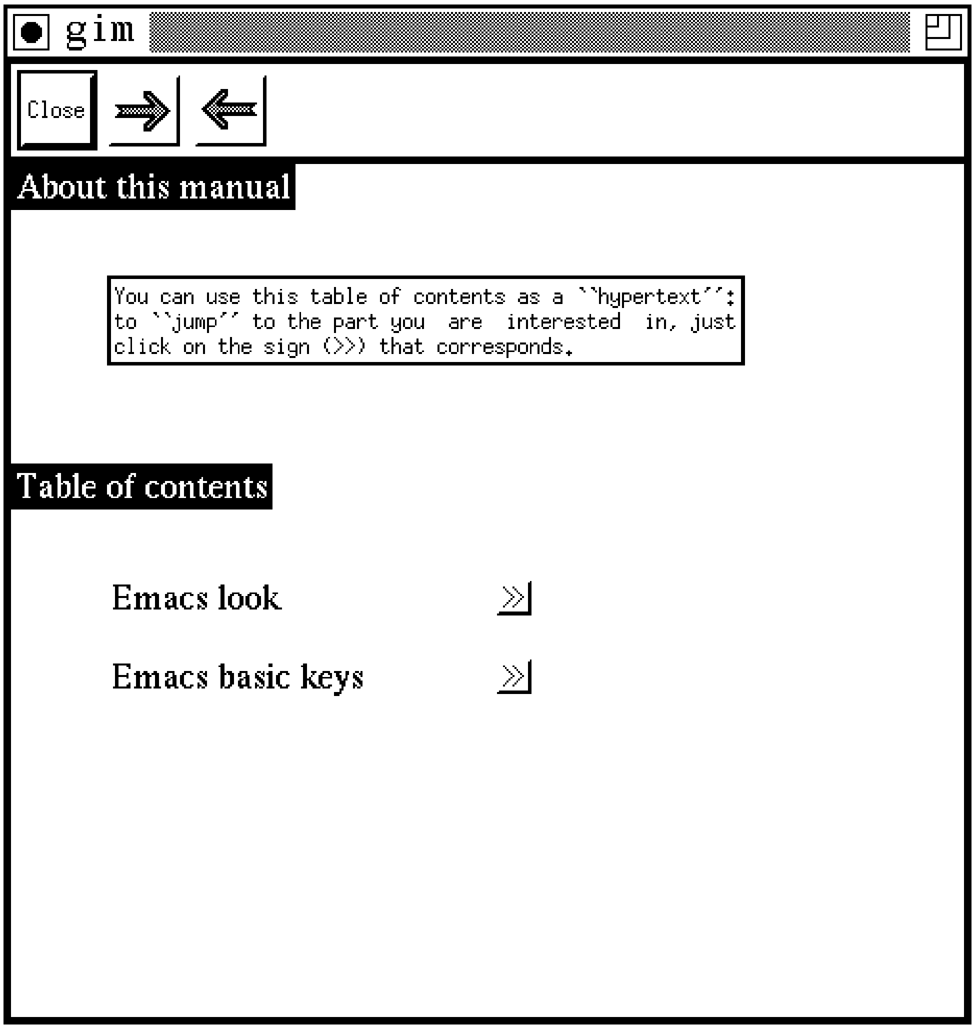


Figure 5: A complete **GIM** manual (1/3)

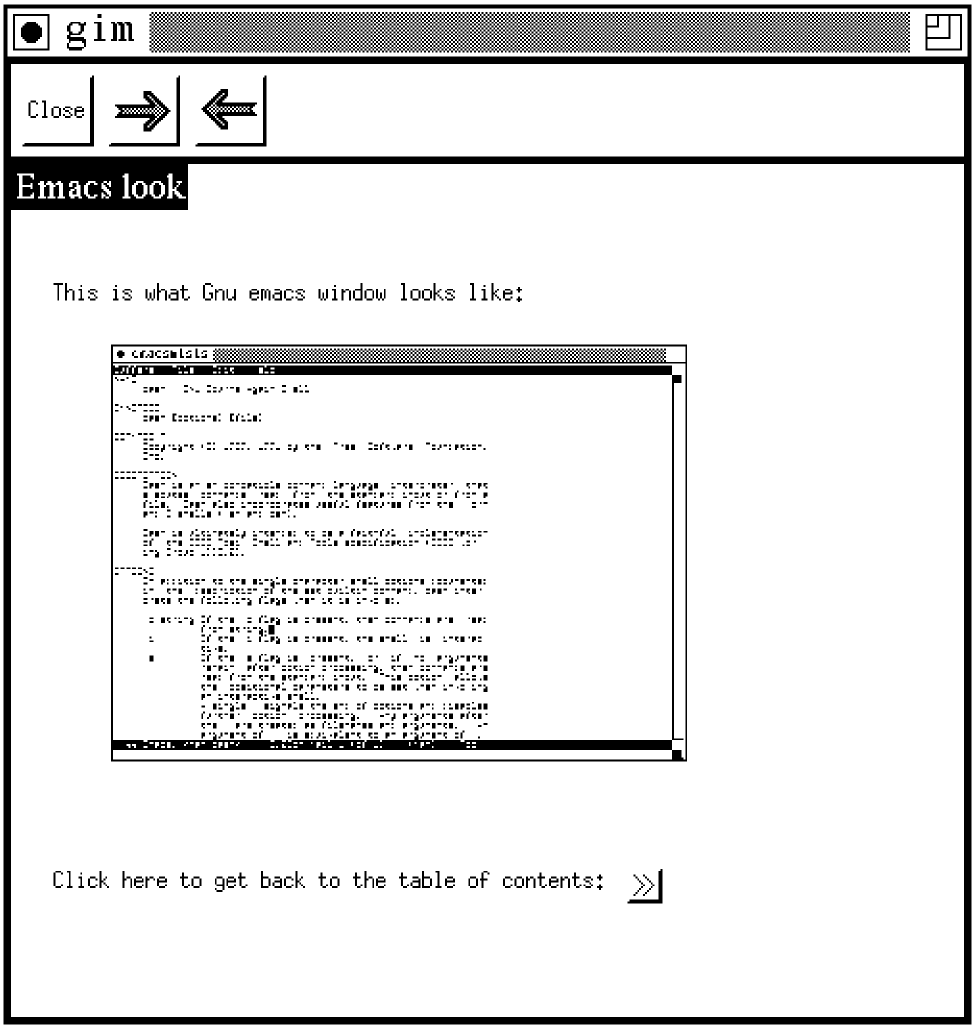


Figure 6: A complete **GIM** manual (2/3)

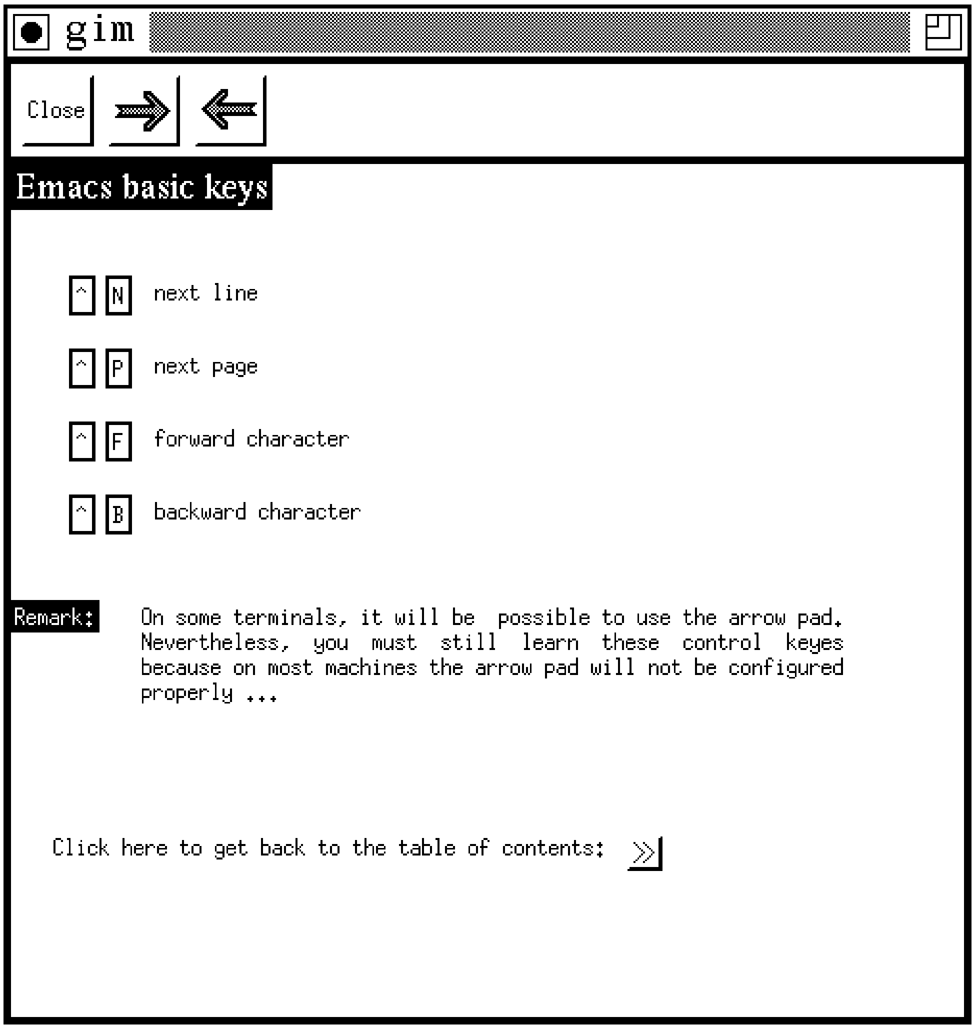


Figure 7: A complete **GIM** manual (3/3)

Program 2: Source file of a complete GIM example

```
\pagesize{500}{500}

\begin{manual}

\em\large
About this manual
\newline\newline\newline

\hspace{50}\fbox
You can use this table of contents as a ``hypertext'':
to ``jump'' to the part you  are  interested  in, just
click on the sign (>>) that corresponds.
\newline

\newline\newline\newline

\label{contents}
\em\large Table of contents

\newline\newline\newline

\hspace{50}\large\makebox{200}Emacs look \ref{look}

\newline\newline

\hspace{50}\large\makebox{200}Emacs basic keys \ref{keyes}

\newpage

\label{look}

\large\em
Emacs look
\newline\newline\newline

\hspace{20}This is what Gnu emacs window looks like:
\newline\newline

\hspace{50}\icon{emacs.bm}

\newline\newline\newline\newline

\hspace{20}Click here to get back to the table of contents: \ref{contents}

\newpage
```

```
\label{keyes}

\large\em
Emacs basic keys
\newline\newline\newline

\hspace{30}
\fbox ^
\hspace{5}
\fbox N
\hspace{10}
next line

\newline\newline

\hspace{30}
\fbox ^
\hspace{5}
\fbox P
\hspace{10}
next page

\newline\newline

\hspace{30}
\fbox ^
\hspace{5}
\fbox F
\hspace{10}
forward character

\newline\newline

\hspace{30}
\fbox ^
\hspace{5}
\fbox B
\hspace{10}
backward character


\newline\newline\newline

\em
Remark:

\hspace{20}
On some terminals, it will be  possible to use the arrow pad.
Nevertheless,  you  must  still  learn  these  control  keyes
because on most machines the arrow pad will not be configured
properly ...
```

```
\newline\newline\newline

\vspace{30}\hspace{20}Click here to get back to the table of contents:
\vspace{30}\ref{contents}

\end{manual}
```

Index

References

[1] D. Heller. *Motif Programming Manual.* O'Reilly & Associates, 1991.

[2] L. Lamport. LaTeX – *A document Preparation System.* Addison-Wesley Publishing Company, 1986.

[3] A. Nye. *X Toolkit intrinsics programming manual.* O'Reilly & Associates, 1990.

[4] D.E. Smyth. Wcl – Widget Creation Library – a thin veneer over Xrm. 1991.