X/Motif Short Course

http://www.hiraeth.com/alan/tutorials/xmotif/

X Motif Day 1

9:30	Fundamentals 1:	What is X? platform independence, network and client/server operation, X protocol, event model, windows
10:30	Practical 1:	first Motif program
11:00	**short break**	
11:10	Fundamentals 2:	X Motif, X intrinsics and X lib, role of libraries, naming conventions
11:40	Fundamentals 3:	Structure of Motif programs, creating and realising widgets, linking call-backs
12:10	Practical 2:	pop-up windows, using call-backs
12:30	**lunch**	
1:30	Practical 3:	looking at the elements: labels, buttons, forms, shells, modifying examples
3:20	**short break**	
3:30	Fundamentals 4:	the widget class hierarchy, resource values parent/child relationship and run-time hierarchy widgets and gadgets, geometry management
4:10	Practical 4:	constructing a main window and menu bar
4:30	**session ends**	

Motif Day 2

9:30	Practical 5:	Pull down menus
10:10	Practical 6:	A full application
11:00	**short break**	
11:10	Reading:	A look at Motif books
11:30	Practical 7:	Adding real application functionality
12:30	**lunch**	
1:30	Fundamentals 5:	Motif style guide and usability
2:00	Practical 8:	Looking at other widgets
3:00	**short break**	
3:10	Practical 9:	More complex applications
4:15	review session	
4:30	**course ends**	

X Motif Reading

1. Dan Heller and Paula Ferguson, "Motif Programming Manual, Volume 6", O'Reilly and Associates. with Motif 1.2 (ISBN 1-56592-016-3). Volume 6B is a reference book on Motif and UIL (ISBN 1-56592-038-4).

2. Douglas Youngs, "The X Window System : Applications and Programming with Xt OSF / Motif Edition", Prentice Hall,1989 (ISBN 0-13-497074-8).

3. Ellie Cutler, Faniel Gilly and Tim O'Reilly,"The X Window System in a Nutshell", O'Reilly and Associates. (ISBN 1-56592-017-1).

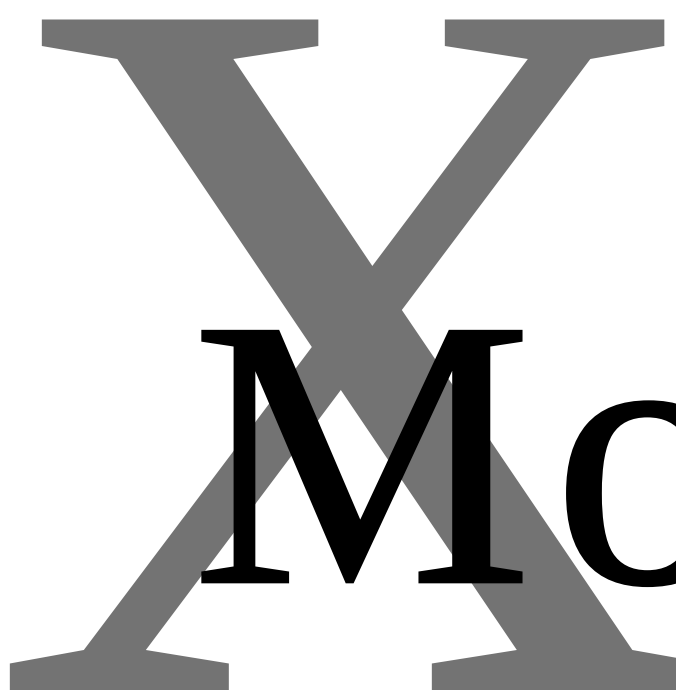
4 Robert Scheifler and James Gettys, "X Window System Release 5, Third Edition", Digital Press, 1992. Digital Press order EY-J802E-DP, ISBN 0-13-971201-1.

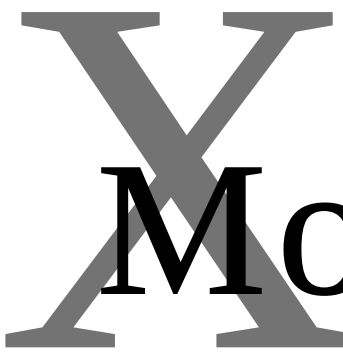
5. Open Software Foundation, "OSF / Motif Programmer's Guide, Revision 2", Prentice Hall, 1995 (ISBN 0-13-143158-7).

6. Open Software Foundation, "OSF / Motif Programmer's Reference, Revision 2", Prentice Hall, 1995 (ISBN 0-13-643115-1).

7. Open Software Foundation, "OSF / Motif Sytle Guide", Prentice Hall.

N.B. Rebadged copies of 5, 6 and 7 may be part of a commercial Motif distribution.





Part I

Fundamentals

X Motif Fundamentals 1

- what is X?
- a little history
- network and client/server operation
- the X protocol
- windows everywhere
- X event model

What is X?

X is a –

- non-proprietary
- platform independent
- client / server
- network transparent
- bitmap window system

X is not –

- a UNIX window manager
- a window manager
- only for UNIX

A Little History

Development of X

1983 Digital Western Research Laboratory
W a prototype window system ported to VS100s terminals

1984 MIT – Project Athena (sponsored by DEC & IBM)
need a UNIX bitmap window system
building on W, but modified significantly (sync → async)
W becomes X !

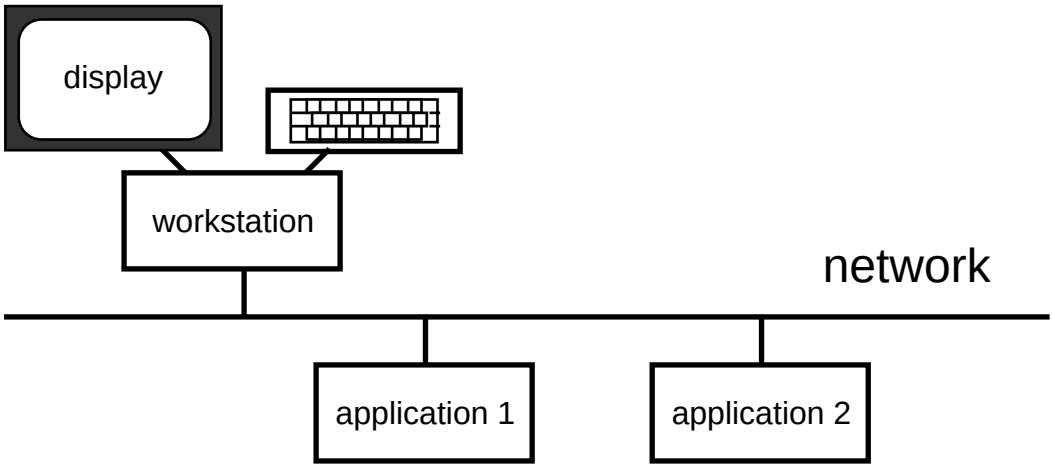
1986/7 X in version 10
ports to HP, Apollo, Sun, IBM
DEC WRL develop version 11 for public distribution

1988 MIT X Consortium founded
aim – to develop X and maintain its independence

Motif

- developed by Open Software Foundation (OSF)
  (a consortium including HP, Digital, IBM developing interoperability standards)
- standard for:
  user interface appearance and behaviour
  programmers API
- modelled on IBM Common User Access (CUA)

X – Network Window System



client/server

X server – on workstation, manages screen
X client – the application

? the wrong way round ?

database – shared resource – data
– server manages – data

X – shared resource – display
– server manages – display

Client / Server

X server

- on UNIX workstation
- on dedicated X terminal
- on PCs
- on Macs

- ❍ manages one or more screens
- ❍ may ‘control’ whole screen or operate with native window manager

X client

- anywhere ! even transcontinental

Special case

- client and server on same machine
- always run as separate processes

X Protocol

X standard

- what goes down the wire

Any wire

- Internet TCP / IP
- DECNet
- even over a modem!

☆ any duplex byte-stream connection

client → server: request e.g., put “Hello World” on screen

server → client: events e.g., user typed “X”

✘ network is bottleneck

- specification is core of X
- asynchronous
- beware of delays

✔ protocol is extensible

- e.g., Motif extensions for workspaces

X ‘windows’

X **window** = rectangle on screen

screen = actual glass thing

display = workstation
possibly many screens
but only one mouse & keyboard

lots of windows:

each button, scrollbar, menu, icon etc.

raw X is very basic

output: draw lines or text

input: react to user actions

resources: manage colour-maps, fonts

structure: hierarchy of windows
children clipped by parents

X is not a window manager

window manger

- separate but special client
- manages window controls:
  title bar, zoom box, resizing, generic menu
- can run anywhere
  same place as server
  same place as client
  somewhere else entirely

X standard defines protocol: client ↔ WM

Possible to bypass window manager . . .
. . . but not a good idea !

Different styles of window manager
including Motif window manager
(N.B. Motif applications do not need MWM)

X Events

Source

- user activities:
  mouse clicks, keyboard, mouse movement
- window manager events:
  keyboard focus, resize
- inter-application communication:
  cut/paste, drag'n'drop

Control

client: registers interest in specific events

server: selects relevant events
merges into a single event stream

Content

type:	keyboard, mouse press etc.
window:	where event occurred
sync & timestamp:	N.B. asynchronous
mouse position:	ditto!
other info:	e.g., which key

Naming

heavy use of IDs – window, fonts, colour-maps ...

X Event Loop

Client code:

```
loop forever
    read next event
    switch ( event.type ) {
        case  mouse press:
                if ( event.window = = ... )
                        . . .
        case  key press:
        case  . . .
    }
end loop
```

The Good News

Motif manages this for you !

The Bad News

Need to learn event based programming

X Motif Fundamentals 2

Different levels

- X protocol
- Xlib – raw X
  X standard API
- Xt – intrinsics
  objects oriented features
  + event management
- Xm – Motif toolkit
  buttons, menus etc.

Naming

XDrawLine

XtVaCreateManagedWidget

xmLabelGadgetClass

Xlib

programmers API for protocol

- should never use protocol directly
- except for building servers etc.

raw X

- open displays
- create windows
- draw graphics: lines, pixels and text

very basic

need toolkits to add basic widgets

e.g., Motif

Xlib calls rarely needed when using Motif

exceptions

- drawing lines, graphics etc.
- understanding event structure

X Motif

Motif compliant toolkit

N.B. Motif is the standard not the toolkit

Defines common GUI widgets

- scroll bars
- buttons
- text areas
- menus
- dialogue boxes

Does a lot of work for you!

- handles a lot of user interaction
- manages layout of windows
- automates resizing, redrawing etc.

… but you can control it if you want to!

Xt – X intrinsics

defines an OO-like layer for widgets

handles common things:

- creating widgets

```
XtVaCreateManagedWidget( "Open",
        xmPushButtonClass, button_area, NULL )
```

(defined in Xm)

- resources for widgets

  attributes such as position, size, etc.

  e.g., `XmNwidth`, `XmNlabelString`

- setting / getting resources

```
XtSetValue( widget,  XmNresName,  resValue  )
```

- reading resource databases

- the event loop

```
XtAppMainLoop( ... )
```

N.B. Motif is largely Xt class definitions

Motif Fundamentals 3

Structure of Motif Program

Main routine

① initialise toolkit
② create widgets
③ (add callbacks where necessary)
④ realise widgets
⑤ enter main event loop

Event routines

- user defined functions
- called when specific events happen
- different ones for different types of events

Initialisation

```
XtVaAppInitialise( ... )
```

① opens connection to X server

② parses command line arguments

(handles standard X arguments such as window position)

③ set-up default resources (e.g., fontname)

④ opens resource databases

⑤ creates a top level ‘shell’ widget

☆ can do it bit by bit for special purposes

Creating Widgets

```
XtVaCreateManagedWidget( name, class, parent,
                         resName1, resVal1, ..., NULL )
```

name – your name for the widget
used to link to resources
e.g., "form"

class – widget class defined by Motif
e.g., XmFormWidgetClass

parent – existing widget often contains child

remaining arguments define resource values
e.g., XmNwidth, 500,

N.B.

- creates the internal data structure
- does not put anything on screen
- does not create X 'window'

Alternative form of call uses a list of resource arguments

Realising Widgets

```
XtRealiseWidget( topWidget )
```

At this point

① an X window is created for the widget
② all its children are realised
③ where appropriate windows are displayed

Terminology:

realised – an X window exists
managed – parent handles positioning (geometry)
mapped – whether or not widget is displayed

N.B. children can be added after a widget is realised

But:

- may be computationally expensive
- appearance can be strange

OK on occasions (see examples)

Adding Callbacks

Many events managed within Motif

e.g., text entry, menu selection

but need to respond to application specific events

e.g., action on button press

need to call application code when event occurs

`XtAddCallback(` *widget, callback-type, func, my-data*)

widget – an existing widget such as a button

type – a callback resource name
which type of event to respond to

e.g., XmNactivateCallback

func – pointer to C function defined by you

e.g., quit_func

my-data – an integer or pointer to your data
passed on to your callback

Defining Callbacks

When the relevant event happens your function is called

The callback function definition:

```
void quit_func( widget, my-data, event-data )
```

widget	–	where the event occurred
my-data	–	the integer or pointer passed in the call to XtAddCallback
event-data	–	the X event structure which caused the callback

Callback functions may:

- update application data
- exit the program (cleanly!)
- add / delete / modify the Motif widgets

Motif Fundamentals 4

widget sub-class hierarchy

different kinds of widget

widget run-time hierarchy

gadgets

Widget Sub-Class Hierarchy

Sub-class hierarchy based on common behaviour

. . . sort of

Two major types of widgets

primitive – small components
e.g., button, text area etc.

composite – areas in which other widgets fit
e.g., scrolled windows, form

N.B. primitive widgets can have children too

In addition, Motif has convenience functions

these generate:

- groups of widgets
- specially configured widgets

e.g., `XmCreateMenuBar`

Some widgets create children of their own

e.g., `xmMainWindowClass`

Primitive widgets

`Label` – read-only text label

various buttons as sub-classes

N.B. not true kind-of relationship)

`CascadeButton`, `DrawnButton`, `PushButton`, `ToggleButton`

`Text` – multi-line text area

`TextField` – single-line text area

`ArrowButton` – button with arrow on it!

`Separator` – used in menus

`Scrollbar` – used in scrollable windows

`List` – usually created by `XmCreateScrolledList`

N.B. string for label is compound string

can contain multiple lines, fonts, sizes etc.

Also lots of convenience widgets:

combo box, menu bar, pull down menu ...

Composite widgets – managed widgets

Managed widgets arrange children in different ways

`RowColumn` – left / right, top / bottom or grid

`DrawingArea` – for graphics

`Frame` – boxes its child

`BulletinBoard` – application defined layout
also several useful sub-classes, inc.

- `Form` – complex arrangements

`Scale` – 'primitive' like widget
set / show values
children are ticks

`ScrolledWindow` – adds scrollbar
with sub-class (code sharing, not kind of)

- `MainWindow` – adds menubar etc.
template for many applications

Composite widgets – shells

Shells interact with the window manager

Their single child is the real application

`TopLevelShell` & `ApplicationShell`
– for application windows

`TransientShell` & `DialogueShell`
– for pop-ups

`MenuShell` – no window manager controls

For simple applications create top level shell with:

```
top_level = XtVaAppInitialise( ... )
```

or Xm convenience routines

Run-Time Hierarchy

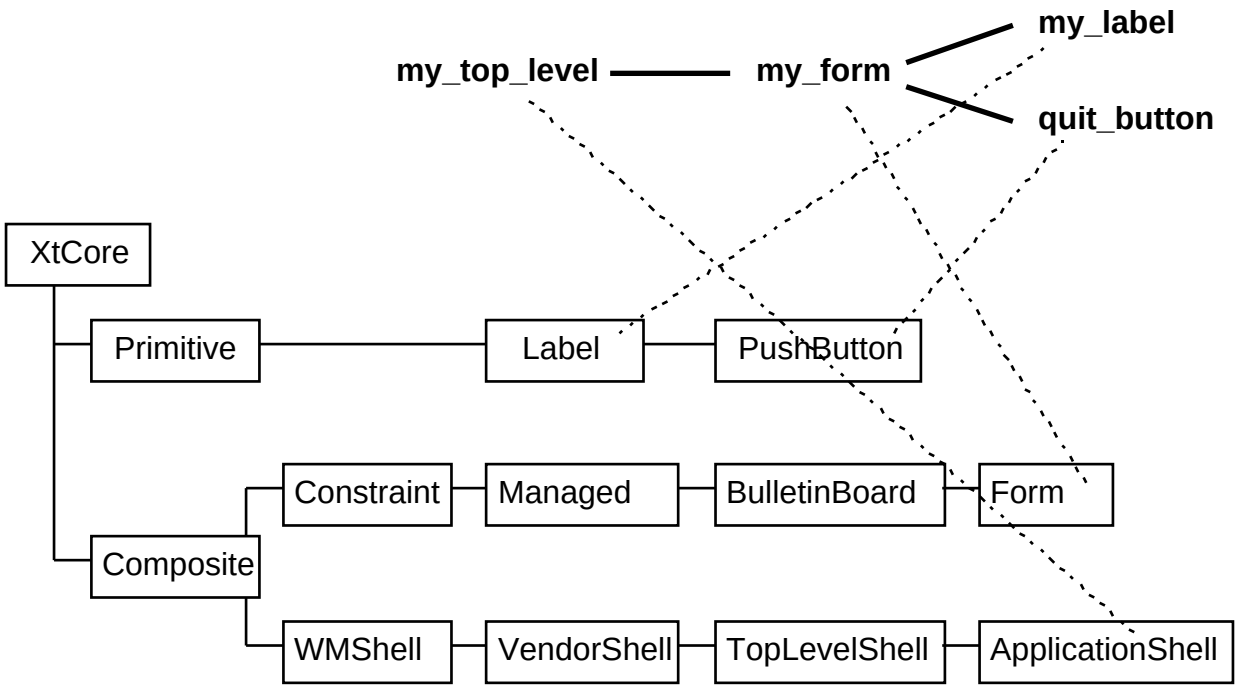
Parent/child relationship between widgets

Normally closely matches parent/child for X windows

exceptions

- dialogues, pop-up and tear-off menus

N.B. very different from sub-class hierarchy!



Gadgets

Gadgets rather like primitive widgets:

- ArrowButtonGadget
- LabelGadget – CascadeButtonGadget
  - PushButtonGadget
  - ToggleBottonGadget
- SeparatorGadget

But have no X window attached

Gadgets don't handle their own events

handled by manager widget that contains them

not full set of resources

e.g., no resource for colour

Why use them?

lots of widgets lots of X windows

No X window more efficient

But X implementations have improved

windows are cheap

Motif Fundamentals 5

Motif is a style not a toolkit

Xm – simply easy way to be Motif

Keeping to style consistency

Xm does a lot . . .

. . . but you have to work too

Reading the Style Guide

Aimed at different people:

- application designers
- widget builders
- toolkit designers
- window manager designers

Most guidelines handled by toolkit / window manager

Relevant parts for application designer:

① user interface design principles
② application design principles
③ internationalisation

\+ remember timing problems!

User Interface Design Principles

General principles and good advice for GUI designers

- adopt the user's perspective
- give the user control
  flexibility, progressive disclosure
- use real-world metaphors
  direct manipulation
  rapid (consistent) response !!
  output → input
- keep interfaces natural
  easy navigation, natural colours
- keep interfaces consistent
- communicate application actions to the user
  feedback, errors, destructive acts
- avoid common design pitfalls
  the process and product

Application Design Principles

Some aspects covered or made easy by Motif toolkit

. . . but you do have to do some work

Which components to use

- menu vs. dialogue
- check box vs. list

etc. ...

Standard menu items: File and Help

Use the right dialogue box!

Principles for arranging screens and menu items

Interaction principles:

- indications of action
- feedback
- user flexibility (customisation)

Motif

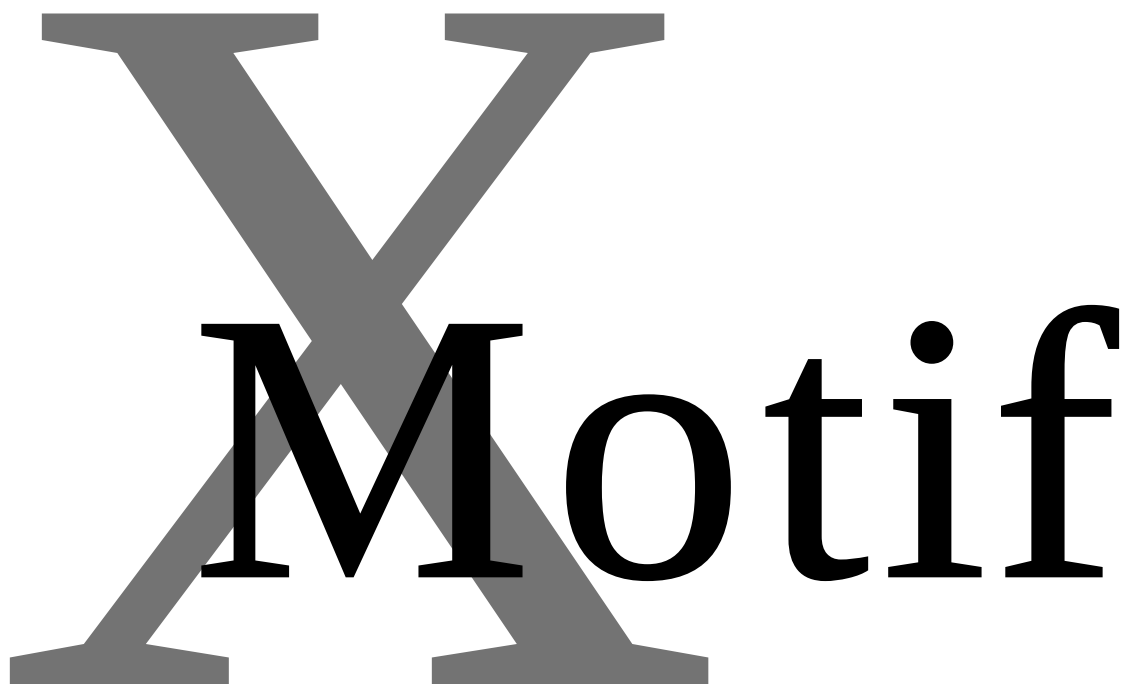
Motif

Part II

Motif

Practical

Motif



Listings

X Motif Listings

Practical 1

Files:

Makefile
XDefs.h
hello.c

Single window with “hello world” and quit button

X Motif Listings

Practical 2/3

File:

hello.c

Single window with “hello world” and quit button, and pop-up window launched by “Press me” button.

X Motif Listings

Practical 4

Files:

hello.c

As practical 2/3, but with quit on menubar rather than on button. Uses MainWindow widget to manage menubar.

X Motif Listings

Practicals 6–9

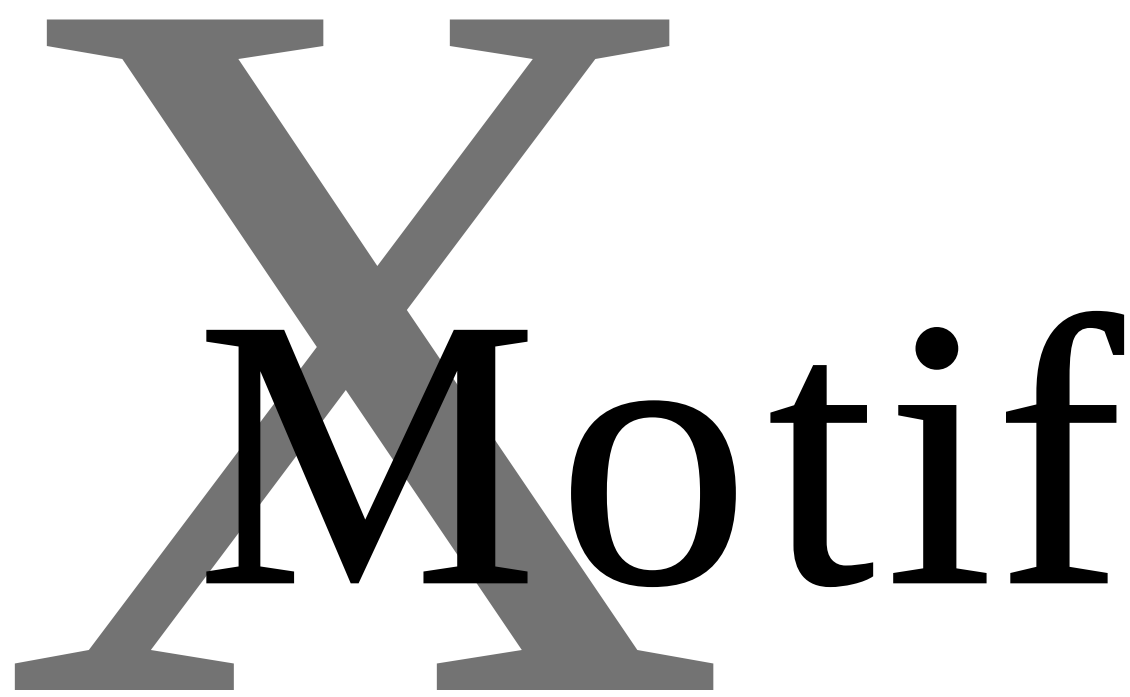
Files:

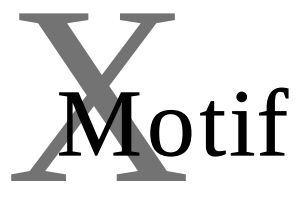
makefile
in_file
db_lib.h
db_lib.c
db.c

This is a small database application. It is written to exemplify different aspects of Motif, not as an example of good interface design.

The application is split into two halves. The front end (db.c) handles the user interface and the back end (db_lib.c) reads and writes the database and performs database searches. The header file db_lib.h defines the programmer's interface between the front end and back end.

This general structure is recommended for any user-interface design project. Obviously in a larger project there would be many files for both back end and front end, but the separation can be maintained. This allows the user interface to be modified easily and also allows different implementations of the back-end functionality. For example, the file db_lib.c could be modified to use a real database engine, rather than a C array. Because of the strong separation this would entail no changes to the user interface.





Notes

X Motif

blank for your notes

blank for your notes

blank for your notes

blank for your notes