A Tentative User and

Reference Manual

for
TclMotif
1.0

Jean-Dominique Gascuel,
iMAGIS/IMAG,
Grenoble, France

Jean-Dominique.Gascuel@imag.fr

Jan Newmarch,

University
of
Canberra,
Australia

jan@pandonia.

canberra.
edu.au

04/20/21

Introduction

TclMotif,
alias
Tm, is

a
binding
of the
Tcl
Language
for
more
information

on *Tcl*
and
Tk,
see
the
very
neat

book
written
by
their
author,
(*An*
Introduction

To Tcl and Tk, J. Ousterout, Addison-Wesley, 1994)

to the
OSF/
Motif
widgets.
Tcl is
an
interpreted

language
originally
intended
for
use as
a
command

language
for
other
applications.
It has
been
used

for
that,
but
has
also
become
useful

as a language in its own right. *Tcl* has been

extended by a set of widgets called *Tk*. The *Tk* widgetsare not based

on the *X toolkit* intrinsics, but are built above *Xlib*. They allow an

easy way of writing *X Window* applications.

The standard set of

widgets in the X world is now the *OSF/Motif* set. This forms a

large set of widgets, and these have been through a large amount of

development over the last five years. Use of this set is sometimes

a
requirement
by
business,
and other
widget sets
try to

conform to them in appearance and behavior. Furthermore, you are

sometimes faced with toolkits that use *X toolkit*-based widgets. In

this case, you have to use a *X toolkit* compatible interface builder.

Tm allows the programmer to use the *OSF/Motif* widgets instead of

the *Tk* widgets from *Tcl* programs. This increases programmer

choices, and allows comparison of the features of both *Tcl* and the *Tk*/

OSF/Motif style of widget programming. The binding gives the

usefull subset of the *OSF/ Motif* widgets, accessible throughthe

simple
interpreted
Tcl
language.

Acknowledgments

Tm is based on *Tk* for the style of widget programming. This was

because it provides a good model, but it also allows the *Tcl*

programmer to move relatively easily between *Tk* and *OSF/Motif*

programming. An alternative style of binding to *OSF/Motif* is used in

the WKSH system, which performs a similar sort of role for the Korn

Shell. An intermediate style is provided by the *Wafe X toolkit-*

based
frontend
based on
Tcl.

*As Ijm
trying to*

understand Tm in deep, I started to insert my own notes in the user

manual provided by Jan Newmarch. As time is going, this notes

becomes more and more important, and I decided that they

may end-up in a usefull user and reference manual for Tm. They

are just my own interpretation of the Scriptures.

Reading this manual

The first section,

Getting Started, might be sufficient for programmers very

familiar both with *OSF/Motif* and *Tcl*. *Tcl* beginners should

start by reading the Ousterout book defining Tcl 7.

The second part, starting at section 2 *Basics*, is a

description of all the basics *OSF/Motif* concepts, intented for *OSF/*

Motif beginners.

The last part of this manual,

starting
from
section ?
have been
written to
be a full
reference

manual of *Tm*, with meaningfull examples, all supported resources,

default values, ...

Finally, the **index** page index should

provide an extensive and easy crossreference of all supported features.

1 Getting Started

Tcl/OSF/ Motif

programs may be run by the *Moat* (MOtif And *Tcl*) interpreter. When

called with
no
arguments
it reads *Tcl*
commands
from

standard
input.
When
called by

`moat`

file-
name
it reads *Tcl*
commands
from
file-

`name`, executes them and then enters the *Moat* event loop. This is

similar to the *Tk* iwishj and theconcept was borrowed from there.

Depending on your shell interpreter, you will probably be able to

run *Tcl*m *OSF/Motif* programs as stand alone programs. If your *Mo*-

at interpreter is installed in `/usr/local`, make this the first

line of your executable program :

[Sorry. Ignored \

```
  begintclmode
  ...  \
endtclmode]
```

1.1 A

simple example

The following example is

in the programs directory as `progEG`. The typical structure of

a *OSF/Motif* program is that the top-level object is a mainWindow. This holds

a menu bar, and a container objectsuch as a form or a rowColumn

which in turn holds the rest of the application objects. So a

mainWindow with a list and some buttons in a form would be created by

[Sorry.
Ignored \
begintclmode
... \
endtclmode]

The `xmForm` acts as what is called the kworkWindowl of the

mainWindow.
This
resource
would be
set by
[Sorry.
Ignored \

```
   begintclmode
   ...   \
endtclmode]
```
Values
would also
be set into

the list and
buttons:
[Sorry.
Ignored \
begintclmode

```
   ...   \
endtclmode]
```
Geometry would be set for the form, to

put the objects in their correct relation to each other. Suppose

this is the list on the left, with the two buttons one under the other

on the right:
[Sorry. Ignored \begintclmode

```
   ...   \
endtclmode]
```
Once evrything has been correctly

setup, we can tell *OSF/Motif* to manage all the widgets, so that they

will be shown on screen :

[Sorry. Ignored \begintclmode

```
   ...   \
endtclmode]
```
The behaviour of our application

would be set by callback functions :

[Sorry. Ignored \

```
   begintclmode
   ...   \
endtclmode]
```
And
finally,
windows

are created and the main event loop is entered:

[Sorry. Ignored \

```
   begintclmode
   ...   \
endtclmode]
```
Once entered in the main

event loop, the application is really running : widgetsare created,

displayed, and manipulated accordingly to user events that trigger the

associated
callbacks.

1.2 What next ?

Thou shall read this manual !

Tm resource names

stick to usual *OSF/Motif* name with a leading - replacing the `XmN`

prefix. The *Tm* constants are specified by their *OSF/Motif*

name, without the `Xm?` prefix, either in upper or lower case.

2
Basics

OSF/Motif use a hierarchy of

sub-
windows
to
create
interface
elements,
such

as
menu
item,
push
button
or text
entry

fields. In the *X toolkit* and *OSF/ Motif*

jargon,
they
are
called
kwidgetsl
Widget
stands

for
window
objects.
.
Widgets
are
just

those
visual
objects
that
can be
seen
on the

screen,
or
interacted
width
by the
mouse
or

keyboard. They are organized in a hierarchy, with

the
application
itself
forming
the its
root.

Programming a graphic user interface mainly consists of the

following
steps :

Creating
all
the
widgets
you

needs,
in a
suitable
hierarchy.

Configuring
colors,
sizes,

alignments,
fonts,
...
In
OSF/
Mo-
tif,

widget
get
their
configuration
options
from
so

called
resources.
These
resources
may
be
set

on
a
per
widget
basis
or
on

a
per
widget
class
basis
(e.
g.

"all
push
buttons
should
have
red
background")

.
Furthermore,
OSF/
Mo-
tif
provides
inheritance

between
widget
classes
(for
instance,
push
button

have
a
background
color
resource,
because
they

inherit
its
existance
(but
not
its
value)

from
Label,
which
inherits
it
from
Primitive,

which
inherits
it
from
Core)
.

Usually,
applications
provide
defaults
resources
for
widget

classes,
and
each
user
modify
some
of

them
on
a
per
session
basis
(file

~/
.
Xdefaults)
.
Programming
your
interface

to
react
to
user
inputs :
what
function

should
be
called
when
the
save
button

is
pushed
?
In
OSF/
Mo-
tif

jargon,
you
add
kcallbacksl
to
widgets.
A

call
back
is a
fragment
of
Tcl
code

which
is
executed
on
a
dedicated
event

(for instance, execute `puts stdout "Hello World"`

when
the
mouse
button
1 is
released
over

the
kpush me"
button)
.

The
following
sections

will detail all this concepts.

2.1

Widget Names

Tcl is a string based language (the only data type is string), and

widget are organized in a hierarchical structure. To accommodate

this, the naming of objects within this hierarchy is similar to the

kabsolute
path
namesl of
Unix files
with a i.j
replacing
the i/j of

Unix. The application itself is known as `ij`. A Form in the

application may be known as i.`form1`j.A Label in this form

may be i.
form1.
okLabelj,
and so on.
Note
that *X*
toolkit

requires
that i . j can
only have
one child
(except for
dialogs,
which are

not
mapped
inside their
parents).
This
naming
convention

is the same
as in *Tk*.

2.2
Widget

creation commands

Widgets belong to classes,

such as Label, xmPushButton or List. For each class there is a creation

command which takes the pathname of the object as first

argument
with
optional
further
arguments:

creationCommand
widgetName
?
managed?
resourceList

where :

creationCommand

is the class

of
the
widget
your
are
creating.
Basically,

all
the
OSF/
Mo-
tif
Xm-
Create-

`SomeWidget()` calls should be binded to a

`xmSomeWidget`
Mo-
at
command.
The
extensive
list

of
currently
supported
creation
command
is

given
below.
widgetName

the
full

path
name
of
the
new
widget.
Note

that
this
specify
both
the
parent
widget

(which
should
already
exists)
,
and
the

name
of
the
new
child.

managed

Before
a
widget
can
be
displayed,

it
must
be
brought
under
the
geometry

control
of
its
parent
(similar
to
placing

a
Tk
widget)
.
This
can
be

done
by
the
`manageChild`
widget
method,
or

by
using
the
`managed`
mainidxentrymanaged
35=
1236=

1237=
1238=
1264=
1291=
1293=
1295=
12

argument
at
creation
time.
If
present,
this

option
should
be
the
first
one.
A

widget
might
be
managed
but
unmaped,
in

which
case
it is
invisible
(see
-
`mapedWhenManaged`,

page rsrc_Core)
.
The
main
use
of
knot

yet
managed
widget"
are
menus
(when
they

are
not
visible)
,
and
sub-
widgets

which
will
resize
to
an
unknown
dimension

at
the
time
of
creation
of

their
parents.

resourceList