

Graphical user interface software

- **examples**

- HTML, CSS, Javascript (XUL, ...)
- Flash, Actionscript, ...
- X Window system, GTK
- Tcl/Tk, TkInter
- Java Swing, GWT
- Visual Basic, C#, ...

- **fundamental ideas**

- interface components: widgets, controls, objects, ...
- properties
- methods
- events: loops and callbacks
- geometry and layout management
- extensive use of hierarchy, inheritance

- **the GUI is the biggest chunk of code in many applications**

- libraries and components try to make it easier
- development environments and wizards and builders try to make it easier
- interfaces are still hard to get working
- and even harder to make work well

Properties, methods, events (Javascript)



asdf Google Wikipedia Reset

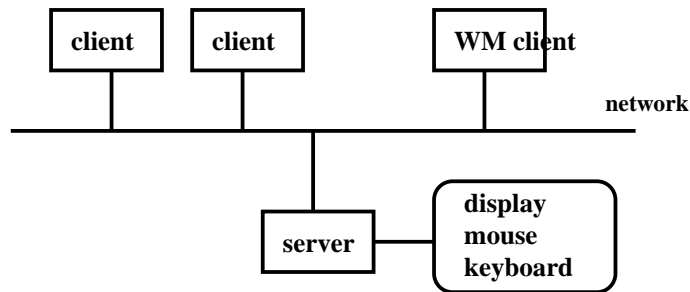
```
<head>
<script>
function setfocus() { document.srch.q.focus(); }
</script>
</head>

<BODY onload='setfocus();'>

<H1>Basic events on forms</H1>
<form action="http://www.google.com/search"
      name=srch>
  <input type=text size=25 name=q
        id=q value="" onmouseover='setfocus() '>
  <input type=button value="Google" name=but
        onclick='window.location=
"http://www.google.com/search?q="+srch.q.value'>
  <input type=button value="Wikipedia" name=but
        onclick='window.location=
"http://en.wikipedia.com/wiki/"+srch.q.value'>
  <input type=reset onclick='srch.q.value=""; >
</form>
```

X Windows (Bob Scheifler & Jim Gettys, 1984)

- **client-server over a network**
 - works on single machine too, with IPC

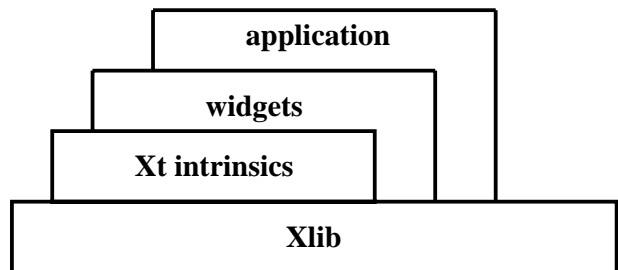


- **variants:**
 - "X terminal" (e.g., SunRay): server is only thing on server, clients are all remote
 - workstation: server is on same processor as clients
 - Exceed: server on PC, clients on (usually) Unix
- **window manager is just another client, but with more properties**
 - clients have to let the window manager manage
 - permits multiple workspaces / virtual windows / virtual desktops

X Windows model (www.x.org)

- **server runs on the local machine**
 - accepts network (or local) client requests and acts on them
 - creates, maps and destroys windows
 - writes and draws in windows
 - manages keyboard, mouse and display
 - sends keyboard and mouse events back to proper clients
 - replies to information requests
 - reports errors
- **client application**
 - written with X libraries (i.e. Xlib, Xt, GTK, ...)
 - uses the X protocol to
 - send requests to the server
 - receive replies, events, errors from server
- **protocol messages**
 - **requests:** clients make requests to the server
e.g., Create Window, Draw, Iconify, ...
 - **replies:** server answers queries ("how big is this?")
 - **events:** server forwards events to client
typically keyboard or mouse input
 - **errors:** server reports request errors to client

X Windows programming model



- **Xlib provides client-server communication**
- **intrinsics provide basic operations for building and combining widgets**
- **widgets implement user interface components**
 - buttons, labels, dialog boxes, menus, ...
 - multiple widget sets, e.g., Motif, GTK
- **applications and libraries can use all of these layers**

Xlib: bottom level library

- **basic mechanisms for**
 - requests from client to server: "draw on window", "how big is this?"
 - replies from server to client: "this big"
 - events from server to client: "button 1 pushed", "window exposed"
 - error reports: "out of memory"
- **basic Xlib-level client program**
 - connect client to server: XOpenDisplay()
 - hints, not mandatory; the WM is in charge
 - get info about screen, compute desired size for window;
 - create window: XCreateSimpleWindow()
 - sizes, window name, icon name, ...
 - set standard properties for window manager
 - select events to be received and discarded
 - create "graphics context" for storing info on color, depth, ...
 - things that don't change from request to request
 - server caches this info to cut down traffic
 - display window: XMapWindow()
 - causes it to appear; up to this point it hasn't
 - loop on events

Events

- client registers with windows system for events it cares about
- events occur asynchronously
- queued for each client
- client has to be ready to handle events any time
 - mouse buttons or motion
 - keyboard input
 - window moved or reshaped or exposed
 - 30-40 others
- information comes back to client in a giant union called XEvent, placed in a queue
- "event loop" processes the queue

```
Xevent myevent;
for (;;) {
    XNextEvent(mydisplay, &myevent);
    switch (myevent.type) {
        case ButtonPress: ...
        ...
    }
}
```

Hello world in X toolkit/ Motif widgets

```
#include <Xm/XmAll.h>

void main(int argc, char *argv[])
{
    Widget toplevel, main_w, button;
    XtAppContext app;
    XtSetLanguageProc(NULL, NULL, NULL);
    toplevel = XtVaAppInitialize(&app, "main", NULL, 0,
                                &argc, argv, NULL, NULL);
    main_w = XtVaCreateManagedWidget("main_w",
                                     xmMainWindowWidgetClass,
                                     toplevel, XmNscrollingPolicy,
                                     XmAUTOMATIC, NULL);
    button = XtVaCreateWidget("Hello World",
                              xmLabelWidgetClass, main_w, NULL);
    XtManageChild(button);
    XtRealizeWidget(toplevel);
    XtAppMainLoop(app);
}

/* cc x.c -lXm -lXt -lX11 */
```

Hello world in GTK (plus ça change...)

```
#include <gtk/gtk.h>

static void hello( GtkWidget *widget, gpointer data ) {
    g_print ("Hello World\n");
}

static gboolean delete_event( GtkWidget *widget, GdkEvent *event,
                              gpointer data ) {
    g_print ("delete event occurred\n");
    return TRUE;
}

static void destroy( GtkWidget *widget, gpointer data ) {
    gtk_main_quit ();
}

int main( int argc, char *argv[] ) {
    GtkWidget *window;
    GtkWidget *button;
    gtk_init (&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    g_signal_connect (G_OBJECT (window), "delete_event",
                     G_CALLBACK (delete_event), NULL);
    g_signal_connect (G_OBJECT (window), "destroy",
                     G_CALLBACK (destroy), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    button = gtk_button_new_with_label ("Hello World");
    g_signal_connect (G_OBJECT (button), "clicked",
                     G_CALLBACK (hello), NULL);
    g_signal_connect_swapped (G_OBJECT (button), "clicked",
                              G_CALLBACK (gtk_widget_destroy),
                              G_OBJECT (window));
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();

    return 0;
}
```

Tcl/Tk

- **Tcl: tool command language**
 - scripting language
 - extensible by writing C functions
- **Tk: (windowing) toolkit**
 - widget set for graphical interfaces
 - (IMHO) the best widget set ever
- **created by John Ousterhout**
 - Berkeley, ~1990
 - see www.tcl.tk
- **embeddings in other languages**
 - TkInter in Python
 - Perl/Tk
 - Ruby
 - ...

Tcl example

- name-value addition

```
while { [gets stdin line] > -1 } {
    scan $line "%s %s" name val
    if {[info exists tot($name)]} {
        incr tot($name) $val
    } else {
        set tot($name) $val
    }
}

foreach i [array names tot] {
    puts "[format {%10s %4d} $i $tot($i)]"
}
```

Hello world in TkInter & Ruby

- Python

```
from Tkinter import *

class App:
    def __init__(self, master):
        frame = Frame(master)
        frame.pack()
        self.button = Button(frame,
                               text="hello world",
                               command=frame.quit)
        self.button.pack()

root = Tk()
app = App(root)
root.mainloop()
```

- Ruby

```
require 'tk'
root = TkRoot.new { }
TkButton.new(root) do
    text "hello world"
    command { exit }
    pack()
end
Tk.mainloop
```

Hello world in Java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class helloworld extends JFrame {

    public static void main(String[] args) {
        helloworld a = new helloworld();
    }

    helloworld() {
        JPanel p1 = new JPanel();
        JButton b = new JButton("hello world");
        p1.add(b);
        getContentPane().setLayout(
            new BorderLayout());
        getContentPane().add(p1,
            BorderLayout.NORTH);
        pack();
        show();
    }
}
```