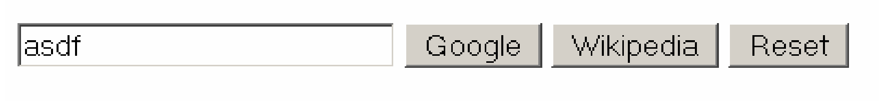
Graphical user interface software

- **examples**
  - HTML, CSS, Javascript (XUL, ...)
  - Flash, Silverlight, ...
  - X Window system, GTk
  - Tcl/Tk, TkInter, ...
  - Java Swing, GWT
  - Microsoft Visual Studio: C#, VB, ...
  - XCode Interface builder, Android XML, ...
- **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

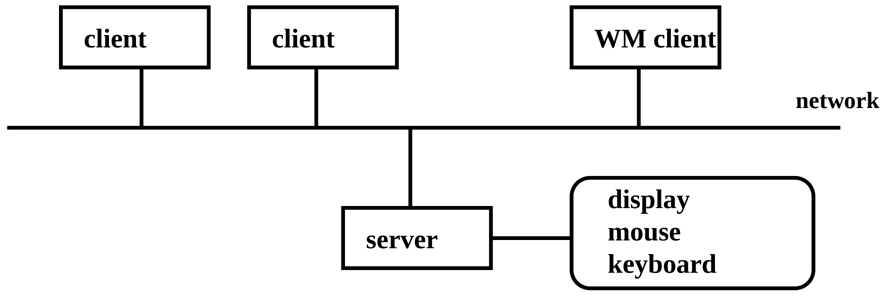
Properties, methods, events (Javascript)



```
<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

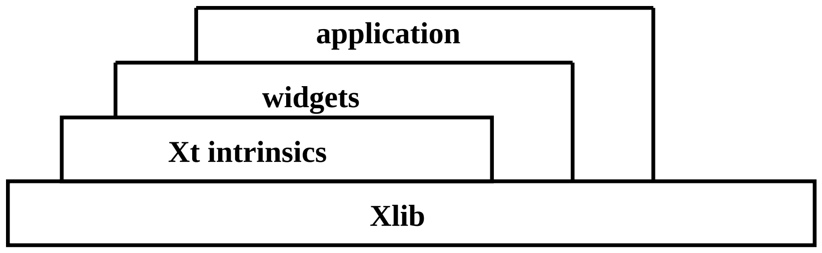


- **variants:**
  - "X terminal" (e.g., SunRay):
    server is only thing on server, clients are all remote
  - workstation: server is on same processor as clients
  - Exceed: server on PC, clients on (usually) Unix
- **window manager is just another client, but with more properties**
  - clients have 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, and 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**
  - initial connection of client to server, window creation, window properties, event mask, ...
  - sends client requests to server: draw, get size, ...
  - sends server responses, errors, etc., to client
  - send events from server, like button push, key press, window expose, ...
- **Xt intrinsics provide basic operations for creating and combining widgets**
- **widgets implement user interface components**
  - buttons, labels, dialog boxes, menus, …
  - widget set is a group of related widgets with common look and feel, e.g., Motif, GTk
- **applications and libraries can use all of these layers**

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: …
    ...
}
```

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)]"
}
```

Tcl example 2: formatter

```
set space ""; set line ""

proc addword {w} {
  global line space
  if {[expr [string length $line] + [string length $w]] > 60} {
    printline
  }
  set line "$line$space$w"
  set space " "
}

proc printline {} {
  global line space
  if {[string length $line] > 0} {
    puts $line
  }
  set line ""; set space ""
}

while {[gets stdin in] >= 0} {
  if {[string length $in] > 0} {
   for {set i 0} {$i < [llength $in]} {incr i} {
      addword [lindex $in $i]
    }
  } else {
    printline
    puts "\n"
  }
}
printline
```

Hello world in TkInter & Ruby

- **Python**

```
from Tkinter import *
root = Tk()
frame = Frame(root)
frame.pack()
button = Button(frame,
                text="hello world", command=frame.quit)
button.pack()
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() {
    JButton b = new JButton("hello world");
    b.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae){
        System.exit(0);
      }
    });
    getContentPane().add(b);
    pack();
    setVisible(true);
  }
}
```