Programming with InterViews

Paul R. Calder

Computer Systems Laboratory
Stanford University
Stanford, California 94305

calder@interviews.stanford.edu

Copyright 1989, 1990, 1991 by Paul R. Calder

Presented at the

Fifth Annual Technical Conference on the X Window System

Boston, Massachusetts

14–16 January 1991

Outline of Tutorial

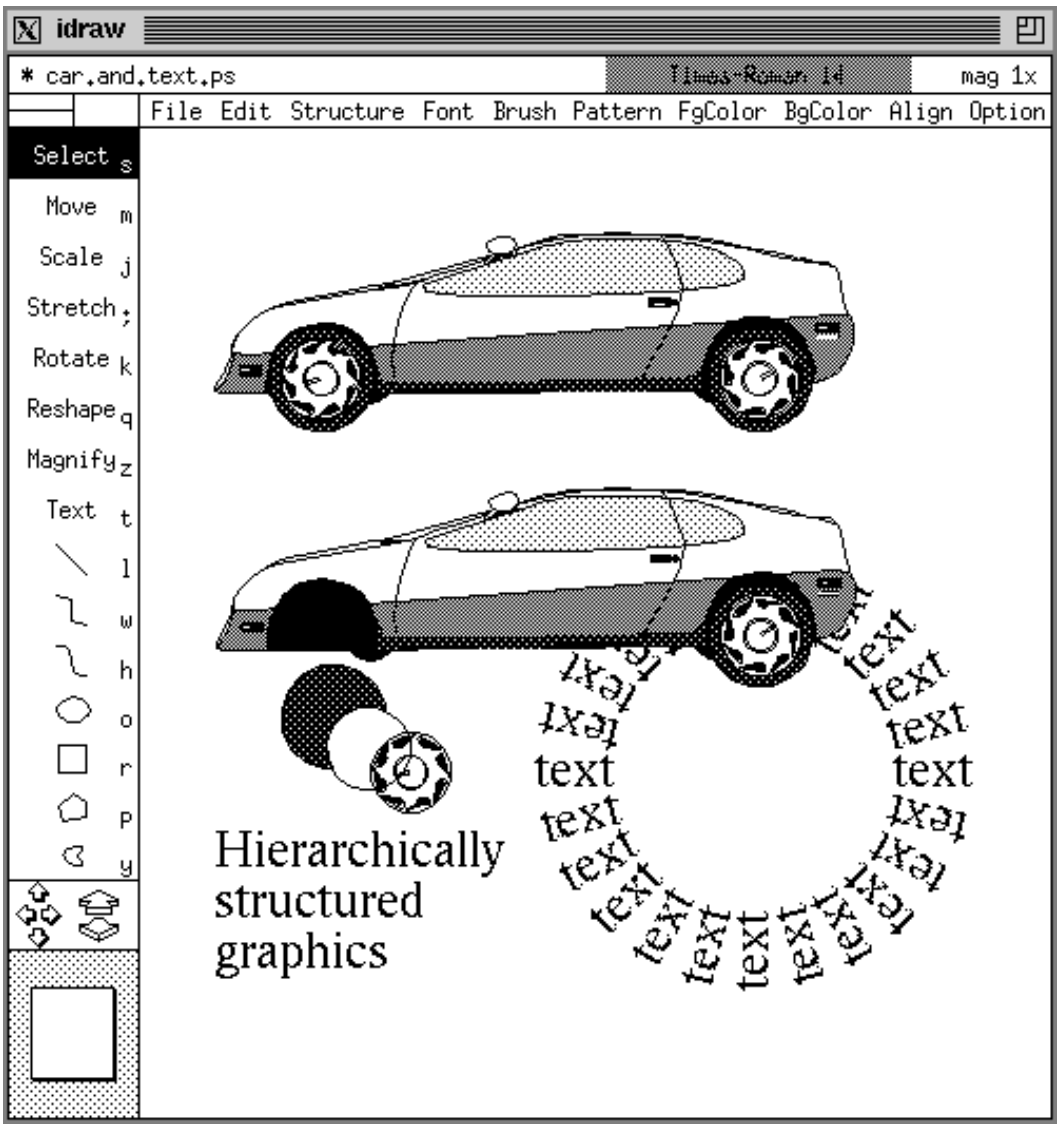
An overview of InterViews 9:00–10:15
- InterViews philosophy
- InterViews and other toolkits
- Interactors, Painters, and Sensors
- Scenes and composition

A simple InterViews application 10:30–12:00
- Bare bones
- Fleshing it out
- Making it customizable

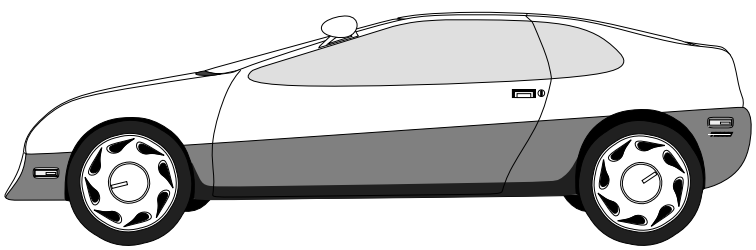
A font browser application 1:30–3:15
- Deriving new classes
- Adding more features
- Final details

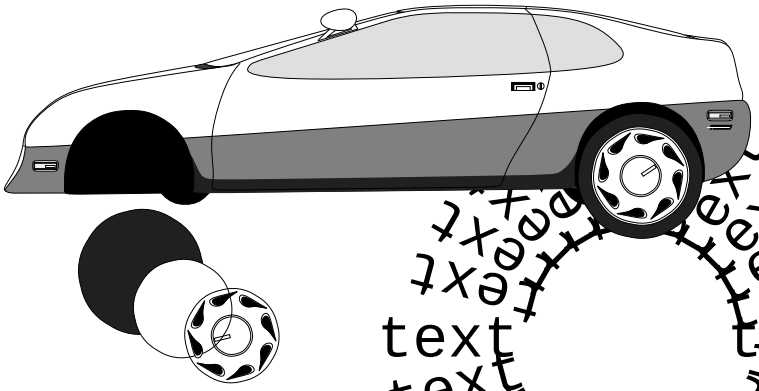
A quick look at InterViews 3.0 3:30–4:30
- Glyphs
- Unidraw
- Backward compatibility

An Overview of InterViews

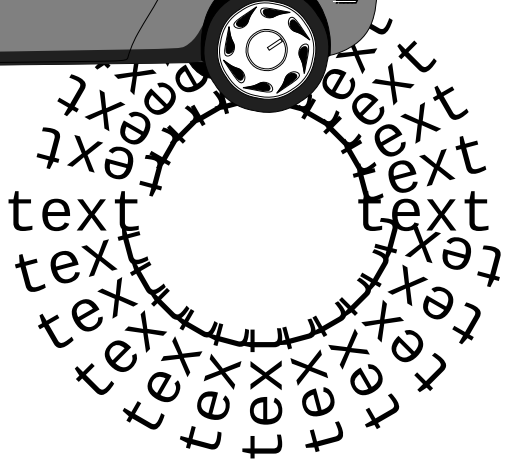


IdrawintheInterViewsdrawingeditor





Hierarchically
structured
graphics


text
text
text

User interfaces...

Allow different interface styles

- standard guidelines
- some users like different styles
- consistency important too

Provide support for text and graphics

- mixture of text and graphics is best
- support below the ‘window’ level

Use an object-oriented model

- object-oriented interfaces are natural
- users directly manipulate ‘real’ objects

Separate abstraction and interactive objects

- with these ‘subjects’ and ‘views’
- good for customization and update

User Interface Toolkits...

Must help the programmer

- familiar objects and metaphors
- components “snap together”
- easy extension

Must be “fast enough”

- fast enough is good enough
- not fast enough is bad

Should be part of larger environment

- InterViews is more than a toolkit
- window system is the “operating system”
- window system controls devices

InterViews: Interactive Views

Emphasize composition

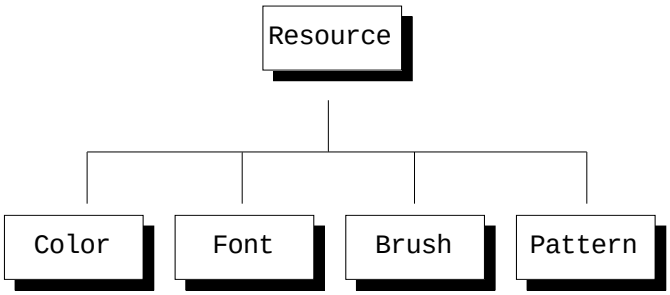
- Scene: Boxes and glue, Tray, ...
- Picture: structured graphics

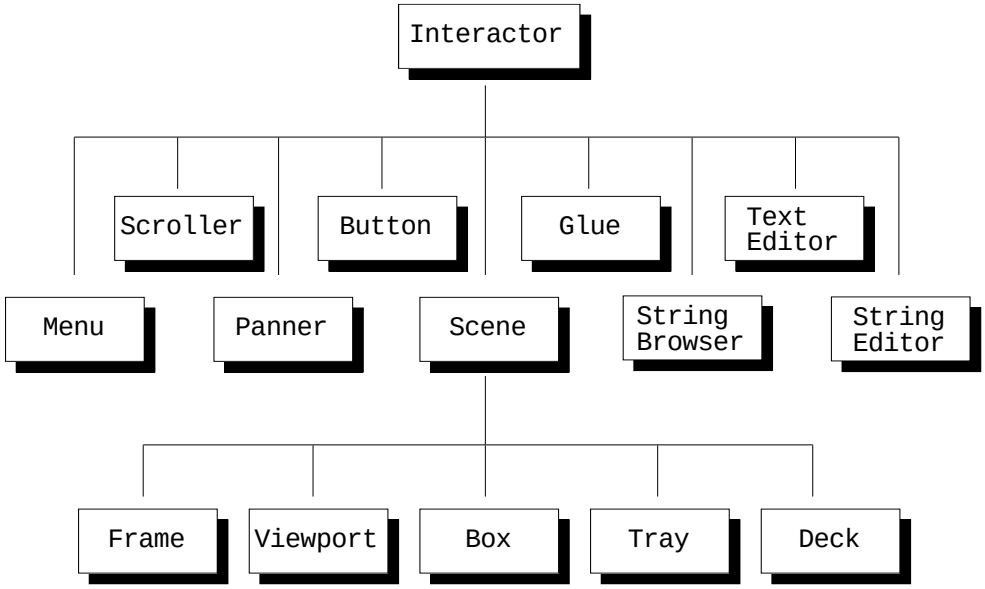
Encapsulate window system

- object-oriented interface
- higher-level graphics
- multiple window systems

Keep it simple

- find the right abstractions
- keep the class hierarchy shallow
- when in doubt, let the application do it
- make the common case easy





Basic InterViews object class hierarchy

Interactors

Base class for all interactive objects

abstract class - no instances

Screen location

Shape specifies desired size
Canvas represents actual screen area

Input

Sensor defines input interest
Handler defines response

Output

Painter contains graphics context
Painter draws on Canvas

Shape

Desired screen allocation

```
class Shape {
    Coord width, height;
    Coord hshrink, vshrink;
    Coord hstretch, vstretch;
    Coord hunits, vunits;
};

const Coord hfil, vfil;
```

Painter

Generating output

```
class Brush, Color, Font, ...;

class Painter {
    void SetBrush(Brush*);
    void SetColors(Color* f, Color* b);
    void SetFont(Font*);
    void SetPattern(Pattern*);
    void SetTransformer(Transformer*);
    ...
    void Line(Canvas*, Coord ...);
    void Rect(Canvas*, Coord ...);
    ...
};
```

Sensor

Input interest

```
enum EventType {
    MotionEvent, KeyEvent,
    DownEvent, UpEvent,
    ...
};

class Sensor {
    void Catch(EventType);
    void Ignore(EventType);
    ...
};

class Event {
    Interactor* target;
    EventType eventType;
    Coord x, y;
    ...
};
```

Scenes

Base classes for compositions

- abstract classes
- no instances

Shape calculated from components

- shape propagates up

Canvases allocated to components

- allocation propagates down

Variety of composition strategies

- tiling, stacking
- aligning, overlapping
- enclosing

Box

Tiles interact side-by-side

HBox left-to-right
VBox top-to-bottom
Glue fow whitespace

Stretches/shrinks to available space

major axis natural size is sum
minor axis natural size is maximum
stretchability is max
shrinkability is min

Row or column arrangements

toolsin a palette
entriesin a menu
buttonsin a dialog

Tray

Constraint and arrangements

alignment constraints among interactors
TGlue represents white space between edges
stretches/shrinks to fit

Non-hierarchical arrangements

row-and-column arrangements
two-way alignments

Deck

Stack of interactors

only one visible at a time

One-at-a-time arrangements

multi-page documents
alternate panels

Single-Component Scenes

Monoscene — abstract base class

base class for application-specific classes

Frame — decoration around an interactor

simple solid border
drop shadow
whitespace margin

Viewport — scrolling and zooming

perspective viewing of a portion

World

The rest of the world

display — the root scene
ring bell, set keyboard, mouse parameters
current screen

Construction and initialization

program defaults
user defaults
command-line arguments

Helper Classes

Text

text display, primitive
display scrolling

Rubberbanding

abstract.

Structured Graphics

graphic objects
picture grouping
damage control

Rasterized data

bitmaps for masks and stencils
rasters or images

Standard Interactor Library

Menus

popup, pulldown, pullright, pinned

Buttons

view of button state
pushbutton, radiobutton, checkbox

Adjusters

view of perspective
scroller, panner, mover, zoomer

Text

string browser for lists
string editor for single-line text
text editor for multi-line text

How is InterViews similar?

Object-oriented

- library of object definitions

Runs on top of X

- wide range of platforms

Basic objects

- interactive and composition objects
- abstract/interactor object

How InterViews is different?

C++

- object-oriented language
- efficiency is important

Composition mechanisms

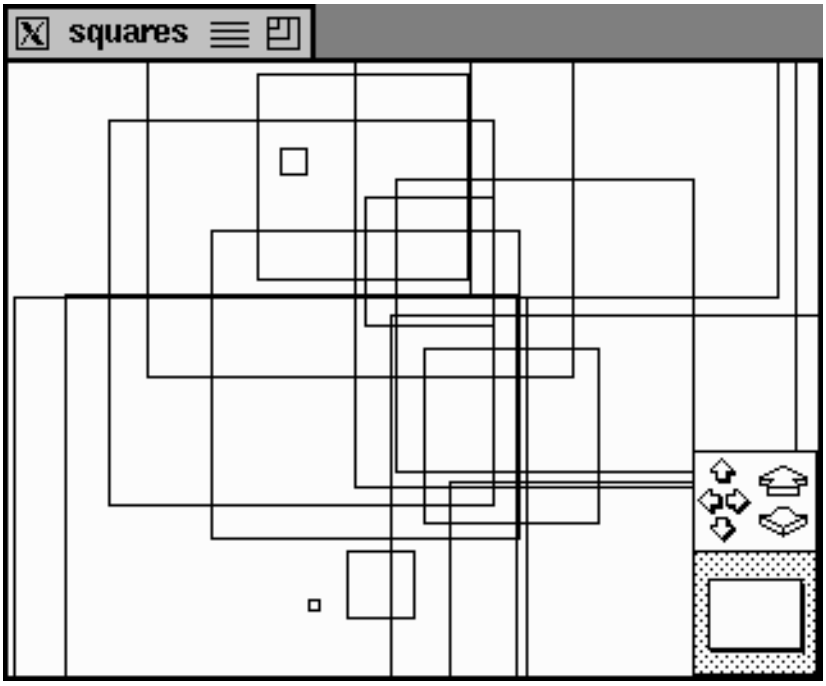
- tiling, stacking, alignment
- stretching and shrinking

Graphics

- graphics state objects
- graphical transformations
- structured graphics objects

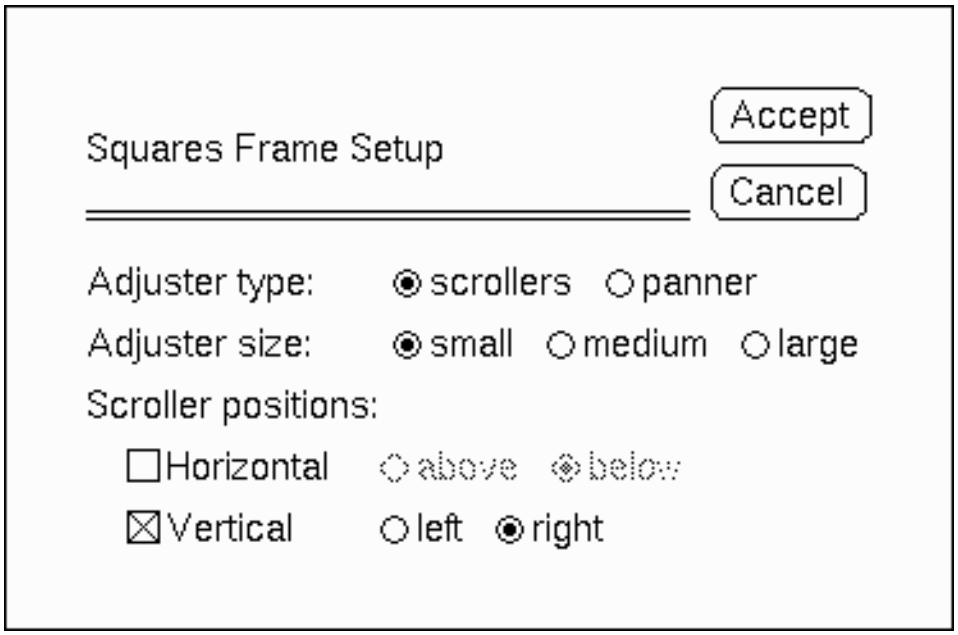
Part of a larger system

- component library
- programming environment
- applications



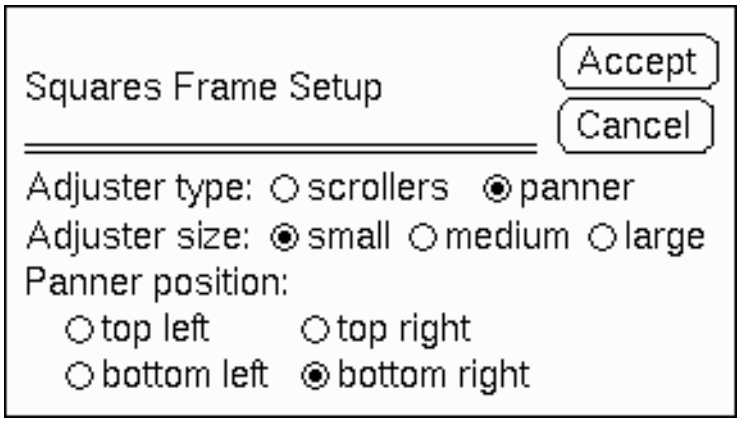
Squaresview

ViewportMenuPanner



SquaresDialog

PushButton, RadioButton, Checkbox, Message



ResizedDialog

Box, Glue, Tray, Deck

InterViews 2.6 Status

Library

- about 35,000 lines of C++
- "vanilla" look and feel

Applications

- demos — squares, graphics
- accessories — alert, clock, idraw, mailbox
- applications — idraw, classify, fb

Runs on many platforms

- X11 and Unix
- DECstation, VAXstation, Sun, HP, Apollo, ...

Publicly available

- X11R4 tape
- ftp interviews.stanford.edu (36.22.0.175)
- interviews@interviews.stanford.edu

InterViews 3.0 Status

Glyphs

flyweight objects that define ''look''
sharable for use in large numbers
Interactor derived from glyph

Unidraw

High-level library for graphical editors
Direct manipulation abstractions

Applications

ibuild—Unidraw-based interface builder
doc—glyph-based document editor

Coming soon

alpha available from interviews.stanford.edu
beta will be on the X11R5 tape

interviews@interviews.stanford.edu

ASimple InterViews Application

A InterViews font displayer

- A text sample using a specified font
- A button to push when done
- Maybe some other embellishments

Compilation details

InterViews library classes

```
#include <InterViews/class.h>
```

Standard compilation and linking

```
%CC -c app.c
%CC app.o -lInterViews -lX11 -lm
```

InterViews Imakefiles

```
CDEBUGFLAGS     = -g
SRC             = .
SRCS            = $(SRC)//**/*.c
LOCAL_LIBRARIES = $(LIBINTERVIEWS)
SYS_LIBRARIES   = -lX11 -lm
OBJS            = \
    ....o \
    main.o

ComplexProgramTarget(app)

MakeObjectFromSrc(...)
MakeObjectFromSrc(main)
```

JustaMessage

Boxes,andGlue

An InterViews ‘‘hello, world’’

```
#include <InterViews/message.h>
#include <InterViews/world.h>

int main () {
  World* world = new World();
  world->InsertApplication(
    new Message("The quick brown fox...")
  );

  world->Run();

  delete world;
  return 0;
}
```

Basic framework

World

opens connection with window system
outer level (‘‘root’’ window)

Message

static display

Build hierarchy bottom up

don’t keep parent first
‘‘declare then specify’’

Event loop hidden in ‘‘Run’’

window maintenance events automatic

Boxing the message

```
#include <InterViews/box.h>
#include <InterViews/glue.h>
#include <InterViews/message.h>
#include <InterViews/world.h>

int main () {
  World* world = new World();
  world->InsertApplication(
    new HBox(
      new HGlue(round(0.5*inch)),
      new VBox(
        new VGlue(round(0.3*inch)),
        new Message("The quick brown fox..."),
        new VGlue(round(0.3*inch)),
      ),
      new HGlue(round(0.5*inch)),
    )
  );

  world->Run();

  delete world;
  return 0;
}
```

Simple composition

Box (HBox, VBox)

- tight composition
- simplified TeX model
- left-to-right and top-to-bottom

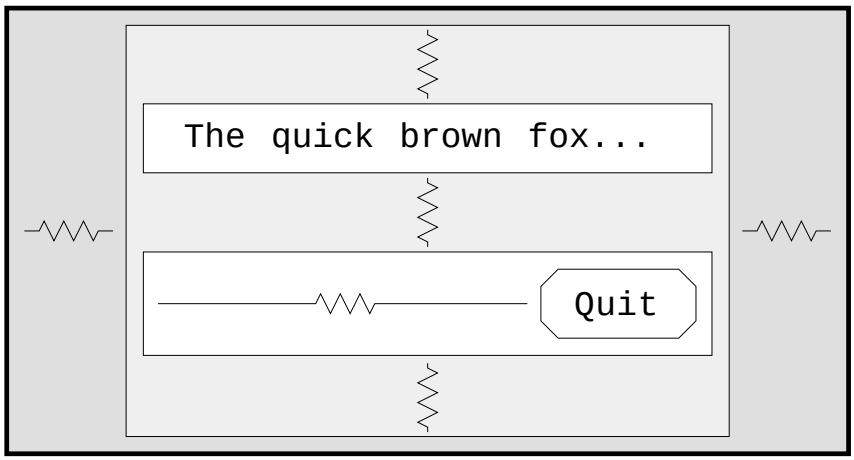
Glue (HGlue, VGlue)

- invisible filler between objects
- natural, stretch, and shrink

A button to push in a Dialog

Set the glue to shrink...

...and stretch



Arrangement of Boxes and Glue

Adding a button

```
#include <InterViews/box.h>
#include <InterViews/button.h>
#include <InterViews/dialog.h>
#include <InterViews/glue.h>
#include <InterViews/message.h>
#include <InterViews/world.h>

int main () {
  World* world = new World();
  ButtonState* quit = new ButtonState(false);
  Dialog* browser = new Dialog(
    quit,
    new HBox(
      new HGlue(0.5*inch, 0.4*inch, 2.0*inch),
      new VBox(
        new VGlue(0.3*inch, 0.2*inch, 1.0*inch),
        new Message("The quick brown fox..."),
        new VGlue(0.3*inch, 0.2*inch, 0),
        new HBox(
          new HGlue(0.3*inch, 0.2*inch, hfil),
          new PushButton("Quit", quit, true)
        ),
        new VGlue(0.3*inch, 0.2*inch, 1.0*inch),
      ),
      new HGlue(0.5*inch, 0.4*inch, 2.0*inch),
    )
  );
  world->InsertApplication(browser);
  browser->Accept();
  world->Remove(browser);
  delete browser;
  Unref(quit);
  delete world;
  return 0;
}
```

Simple user interaction

ButtonState

- a subject (a piece of state)
- a list of views (buttons)
- views notified when state changes

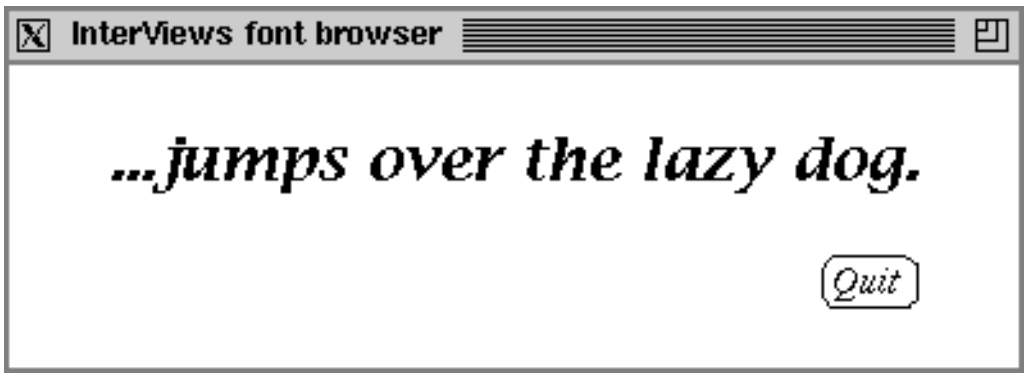
Button

- user pushes the button to change the state
- other types of button available

Dialog

- event loop inside
- exits when button state is set to ''false''

Customization



...for the font and sample

Making it customizable

```
#include ...

static PropertyData properties[] = {
  { "tutorial*sample", "The quick brown fox..." },
  { "tutorial*font",
    "*-helvetica-medium-r-*-240-*" },
  { "tutorial*PushButton*font",
    "*-times-medium-i-*-140-*" },
  { nil }
};

static OptionDesc options[] = {
  { "-sample", "tutorial*sample", OptionValueNext },
  { nil }
};

void main (int argc, char* argv[]) {
  World* world = new World(
    "tutorial", properties, options, argc, argv
  );
  ...
  Dialog* dialog =  new Dialog(
    ...
       new Message(world->GetAttribute("sample")),
    ...
  );
  ...
}
```

Properties and Customization

Properties allow customization

- application defaults
- user preferences

Pass arguments to World

- standard XDG mechanism
- command line arguments

PropertyData defines defaults

- hierarchical structure
- name/value specification
- InterViews handles standard properties

OptionDesc controls command parsing

- pattern, property, type
- matches removed from argument list

Summary

Create a World to initialize

pass default and command-line arguments

Build interaction hierarchy

specify layout with Boxes and Glue
Message for simple text
Button for interactively setting state

Insert top-level interactors into the world

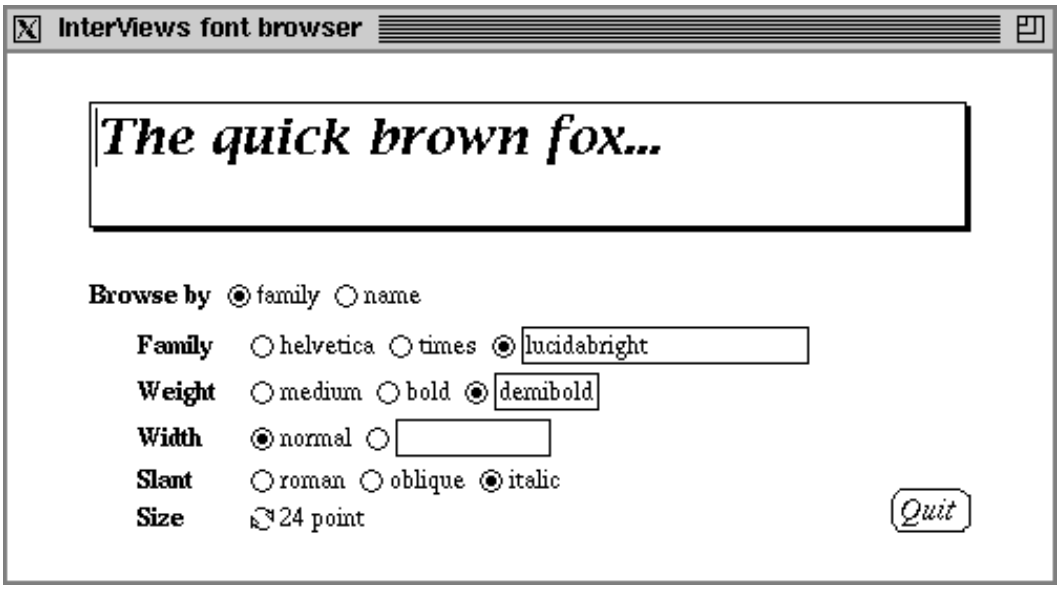
window appears on display

Loop reading events

Dialog does this here

AmterViews FonBBrowser

An InterViews font browser



- An editable text sample
- Buttons to browse fonts
- Type in from non-standard entries

Thecodeasitstands

```
#include ...

static PropertyData properties[] = {...};
static OptionDesc options[] = {...};

void main (int argc, char* argv[]) {
  World* world = new World("tutorial", ...);
  ...
  ButtonState* quit = new ButtonState(false);
  Dialog* browser =  new Dialog(
    quit,
    new HBox(
      ...
        new Message(...),
      ...
    )
  );
  ...
  world->InsertApplication(dialog);
  browser->Accept();
  ...
}
```

Defining a new class

```
class FontSample : public Interactor {
public:
  FontSample(const char* sample);
  FontSample(const char* name, const char* sample);
  virtual ~FontSample();

  void ShowFont(const char* name);
protected:
  virtual void Reconfig();
  virtual void Redraw(Coord, Coord, Coord, Coord);
protected:
  Font* font;
  char* fontname;
  char* sample;
private:
  void Init(const char*);
};
```

Useittoshowthesample

```
#include ...
#include "fontsample.h"

...

void main (int argc, char* argv[]) {
  ...
  FontSample* sample = new FontSample(
    world->GetAttribute("sample")
  );
  sample->ShowFont(
    world->GetAttribute("font")
  );
  Dialog* browser =  new Dialog(
    ...
      sample
    ...
  );
  ...
}
```

Constructors and destructors

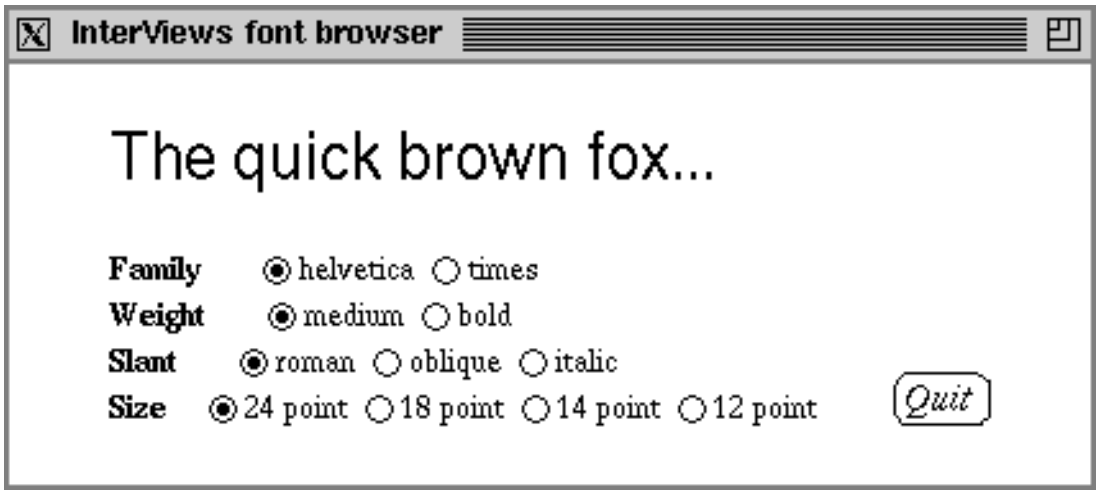
```
FontSample::FontSample(
  const char* sample
) : Interactor() {
  Init(sample);
}

FontSample::~FontSample () {
  delete sample;
  delete fontname;
  Unref(font);
}

void FontSample::Init (const char* samp) {
  SetClassName("FontSample");
  font = nil;
  fontname = new char [BufferSize];
  sample = new char [BufferSize];
  strcpy(sample, samp);
}
```

Configuration

```
void FontSample::Reconfig () {
  Painter* tmp = new Painter(output);
  Unref(output);
  output = tmp;
  output->Reference();
  output->SetFont(font);
  Interactor::Reconfig();
  shape->Rect(font->Width(sample), font->Height());
  shape->Rigid(hfil, hfil, 0, 0);
}
```

Redraw

```
void FontSample::Redraw (
  Coord l, Coord b, Coord r, Coord t
) {
  output->ClearRect(canvas, l, b, r, t);
  output->Text(canvas, sample, 0, 0);
}

void FontSample::ShowFont (const char* name) {
  if (strcmp(name, fontname) != 0) {
    strcpy(fontname, name);
    Unref(font);
    font = new Font(fontname);
    font->Reference();
    if (output != nil) {
      if (font->Valid()) {
        output->FillBg(true);
        output->SetPattern(solid);
        Reconfig();
      } else {
        output->FillBg(false);
        output->SetPattern(gray);
      }
      Draw();
    }
  }
}
```



Buttons to browse the font

FontBrowser – a special Dialog

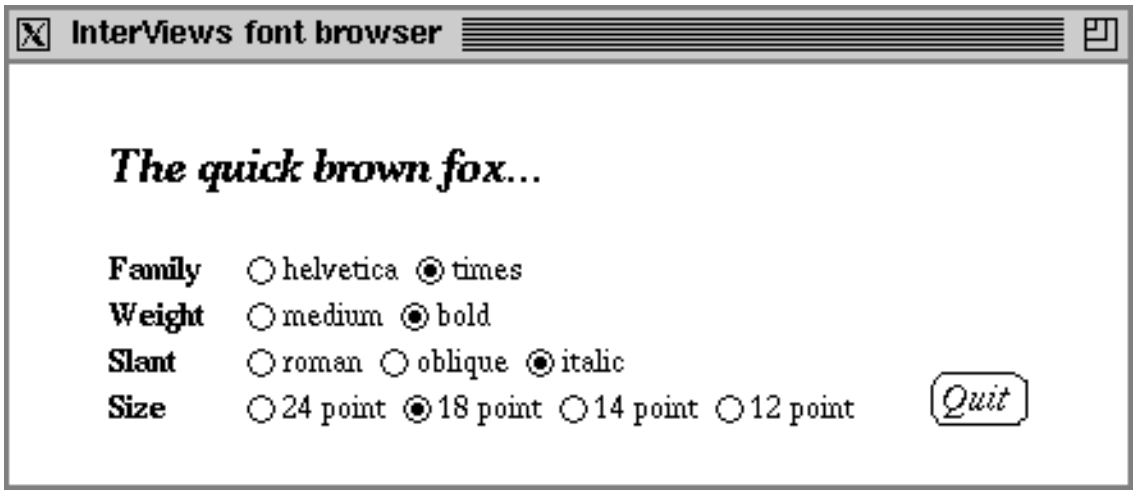
```
class FontBrowser : public Dialog {
public:
  FontBrowser(ButtonState*, const char* sample);
  virtual ~FontBrowser();

  virtual boolean Accept();
  const char* Fontname();
protected:
  void UpdateFontname();

protected:
  char* fontname;
  FontSample* sample;
  ButtonState* family;
  ButtonState* weight;
  ButtonState* slant;
  ButtonState* point;
private:
  void Init(const char*);
};
```

Attributes customization

```
static PropertyData properties[] = {
    ...
    { "tutorial*choice*font", ...},
    { "tutorial*label*font", ...},
    ...
    { nil }
};

...

int main(...) {
    ...
    world->InsertApplication(browser);
    browser->Accept();
    world->Remove(browser);

    puts(browser->Fontname());
    ...
}
```

Building a fontname

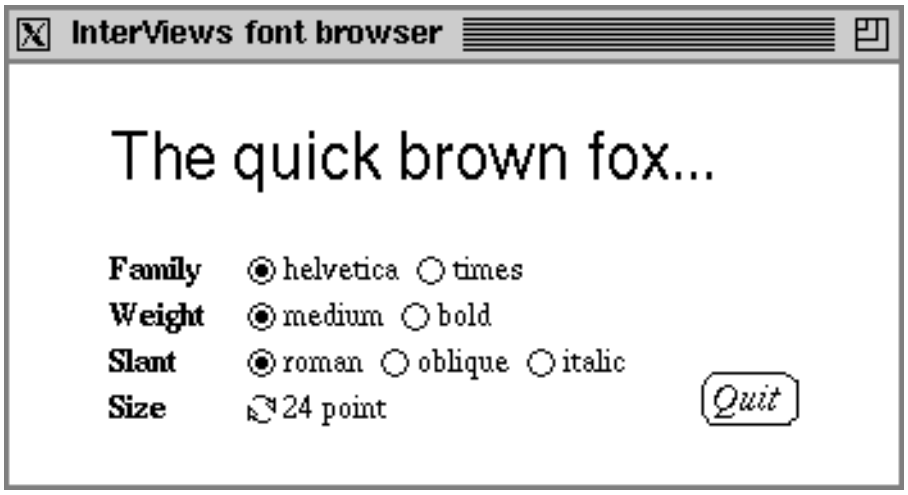
```
boolean FontBrowser::Accept () {
    Event e;
    int v;
    do {
        Read(e);
        e.target->Handle(e);
        UpdateFontname();
        state->GetValue(v);
    } while (v == 0 && e.target != nil);
    return v == 1 || e.target == nil;
}

void FontBrowser::UpdateFontname () {
    void* v;
    family->GetValue(v);
    const char* fam = (char*)v;
    weight->GetValue(v);
    const char* wgt = (char*)v;
    slant->GetValue(v);
    const char* sl = (char*)v;
    point->GetValue(v);
    const char* pnt = (char*)v;
    sprintf(fontname,
        "*-*-%s-%s-%s-normal-*-*-%s0-...",
        fam, wgt, sl, pnt
    );
    sample->ShowFont(Fontname());
}
```



ATraylinesthingsup.

Using a Tray for the layout

```
Tray* t = new Tray();
Interactor* familylabel = new Message(
  "label", "Family"
);
Interactor* familychoices = ButtonBox(
  "choice", family, FamilyChoices
);
...
t->VBox(
  t,
  familychoices,
  new VGlue(0.05*inch,0.05*inch, 0),
  ...
  t
);
...
t->HBox(
  t,
  familylabel,
  new HGlue(0.2*inch, 0.1*inch, 0),
  familychoices,
  new HGlue(0, 0, hfil),
  t
);
t->Align(
  VertCenter, familylabel, familychoices
);
...
t->Align(Left, familylabel, ...);
t->Align(Left, familychoices, ...);
```



Anewtypeofbutton.

CycleButton

```
class CycleButtonChoice {
public:
  const char* label;
  void* value;
};

class CycleButton : public Button {
public:
  CycleButton(
    ButtonState*, CycleButtonChoice* choices
  );
  virtual ~CycleButton();
protected:
  virtual void Reconfig();
  virtual void Redraw(Coord, Coord, Coord, Coord);
  virtual void Refresh();
  virtual void Press();
  virtual void Update();
protected:
  CycleButtonChoice* choices;
  int count;
  int current;
private:
  void Init(CycleButtonChoice*);
};
```

Constructors and destructors

```
...
static Bitmap* CycleMask;
static Bitmap* CyclePlain;
static Bitmap* CycleHit;
...
CycleButton::CycleButton (
  ButtonState* subject, CycleButtonChoice* choices
)  : Button(subject, nil) {
  Init(choices);
}
...
CycleButton::~CycleButton () { }
...
void CycleButton::Init (CycleButtonChoice* c) {
  SetClassName("CycleButton");
  if (CycleMask == nil) {
    CycleMask = new Bitmap(...);
    CycleMask->Reference();
    CyclePlain = new Bitmap(...);
    CyclePlain->Reference();
    CycleHit = new Bitmap(...);
    CycleHit->Reference();
  }
  choices = c;
  for (
    count = 0; choices[count].label != nil; ++count
  ) { }
  current = 0;
}
```

Configuration

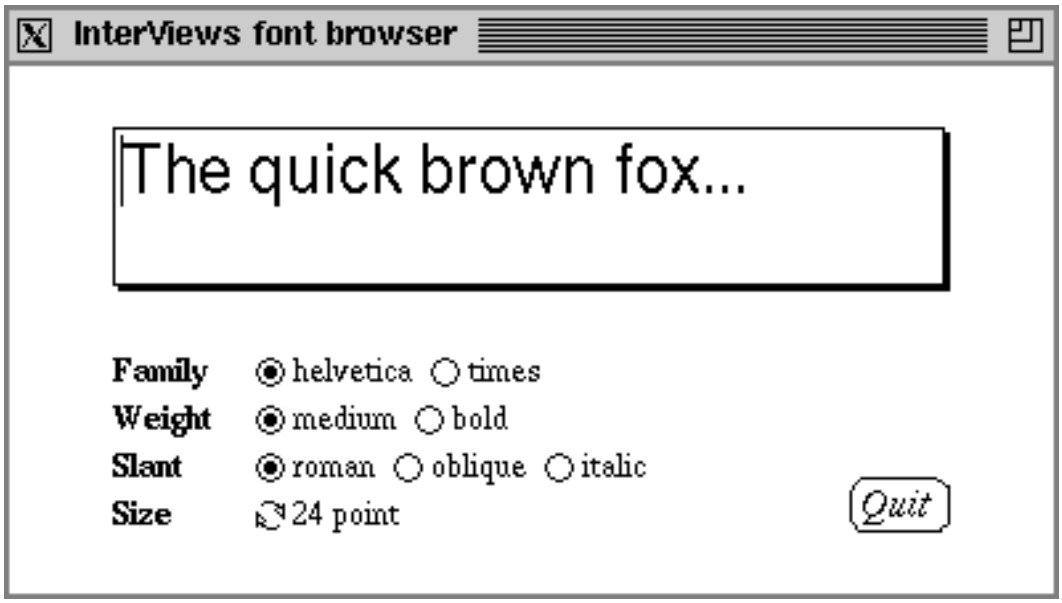
```
void CycleButton::Reconfig () {
  Button::Reconfig();
  Font* font = output->GetFont();
  int width = 0;
  for (int i = 0; i < count; ++i) {
    width = max(
      width,
      font->Width(choices[i].label)
    );
  }
  shape->Rect(
    width + cycle_width + Separator, font->Height()
  );
  shape->Rigid();
}
```

Drawing

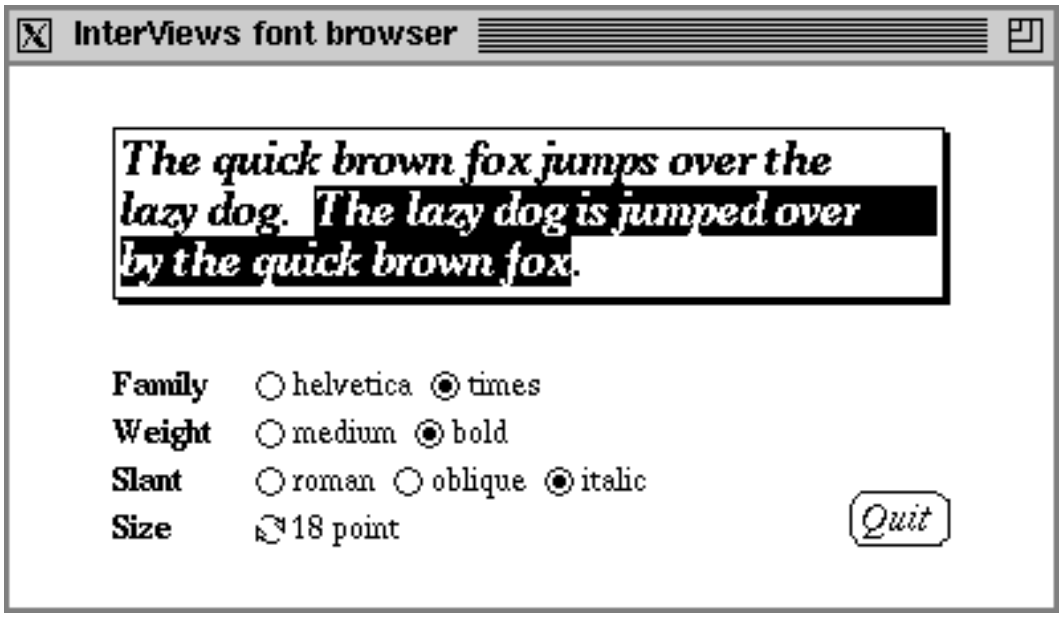
```
void CycleButton::Redraw (
  Coord l, Coord b, Coord r, Coord t
) {
  Coord textx = cycle_width + Separator;
  Coord texty =
    (ymax + 1 - output->GetFont()->Height()) / 2;
  output->ClearRect(canvas, l, b, r, t);
  output->Text(
    canvas, choices[current].label, textx, texty
  );
  Refresh();
}

void CycleButton::Refresh () {
  Coord bitsx = 0;
  Coord bitsy = (ymax + 1 - cycle_height)/2;
  if (!hit) {
    output->Stencil(
      canvas, bitsx, bitsy, CyclePlain, CycleMask
    );
  } else {
    output->Stencil(
      canvas, bitsx, bitsy, CycleHit, CycleMask
    );
  }
}
```

Update

```
void CycleButton::Press () {
  subject->SetValue(
    choices[(current + 1) % count].value
  );
}

void CycleButton::Update () {
  void* v;
  subject->GetValue(v);
  if (v != choices[current].value) {
    for (int i = 0; i < count; ++i) {
      if (v == choices[i].value) {
        current = i;
        break;
      }
    }
  Draw();
  }
}
```



An editable text sample...



...for larger samples.

Changes to FontSample

```
class FontSample : public TextEditor {
  ...
  virtual void Handle(Event&);
private:
  ...
  TextBuffer* text;
};
```

Construction and destruction

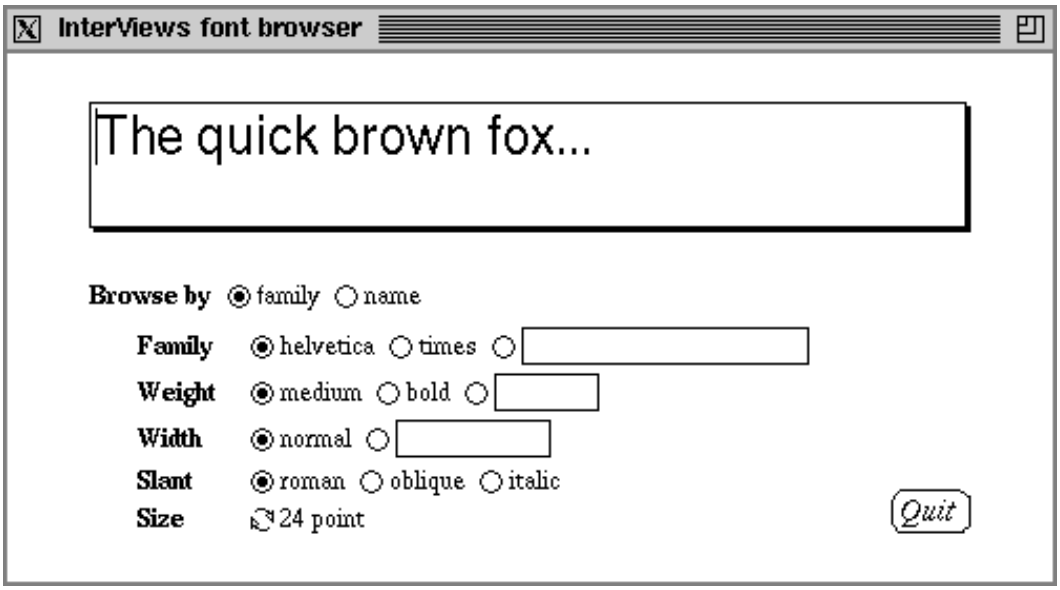
```
FontSample::FontSample(
  const char* sample
) : TextEditor(2, strlen(sample), 8, Reversed) {
  Init(sample);
}

FontSample::~FontSample () {
  delete text;
  delete sample;
  delete fontname;
  Unref(font);
}

void FontSample::Init (const char* samp) {
  ...
  text = new TextBuffer(sample, 0, BufferSize);
  text->Insert(0, samp, strlen(samp));
  Edit(text);
  input = new Sensor();
  input->Catch(KeyEvent);
  input->Catch(DownEvent);
}
```

Configuration and display

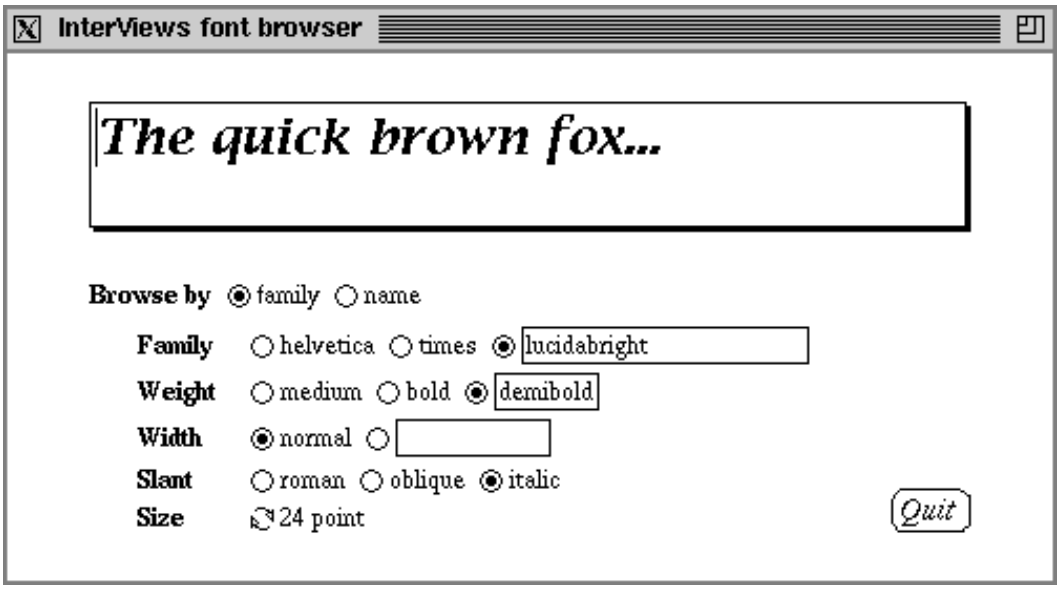
```
void FontSample::Reconfig () {
  ...
  TextEditor::Reconfig();
  shape->vshrink = shape->height - shape->vunits;
}

void FontSample::ShowFont (const char* name) {
  if (strcmp(name, fontname) != 0) {
    ...
    if (output != nil) {
      if (font->Valid()) {
        ...
        Resize();
      } else {
        ...
      }
      Draw();
    }
  }
}
```

Event handling

```
void FontSample::Handle (Event& e) {
  ...
  switch (e.eventType) {
  case KeyEvent:
    if (e.len > 0) {
      char c = e.keystring[0];
      switch(c) {
      case '\006':
        ForwardCharacter(1);
        break;
      case ...
        break;
      default:
        if (c == '\015' || !iscntrl(c)) {
          if (Dot() != Mark()) {
            DeleteSelection();
          }
          InsertText(c=='\015' ? "\n" : &c, 1);
        }
      }
    }
    ScrollToSelection();
    break;
  case DownEvent:
    Select(Locate(e.x, e.y));
    do {
      ScrollToView(e.x, e.y);
      SelectMore(Locate(e.x, e.y));
      Poll(e);
    } while (
      e.leftmouse || e.middlemouse || e.rightmouse
    );
    break;
  }
}
```



Edit buttons...



...four usual values.

StringButton

```
class StringButton : public RadioButton {
public:
  StringButton(ButtonState*, const char* sample);
  ~StringButton();
protected:
  virtual void Handle(Event&);
  virtual void Redraw(Coord, Coord, Coord, Coord);
  virtual void Resize();
  virtual void Reconfig();
private:
  char* text;
  TextDisplay* display;
  int offset;
private:
  void Init();
};
```

Construction and destruction

```
StringButton::StringButton (
  ButtonState* s, const char* sample
) : RadioButton(sample, s, nil) {
  Init();
}

StringButton::~StringButton () {
  delete text;
  delete display;
}

void StringButton::Init () {
  text = new char[1000];
  text[0] = '\0';
  value = (void*)text;
  display = new TextDisplay();
  display->ReplaceText(0, text, strlen(text));
  display->CaretStyle(NoCaret);
  offset = 0;
}
```

Configuration and drawing

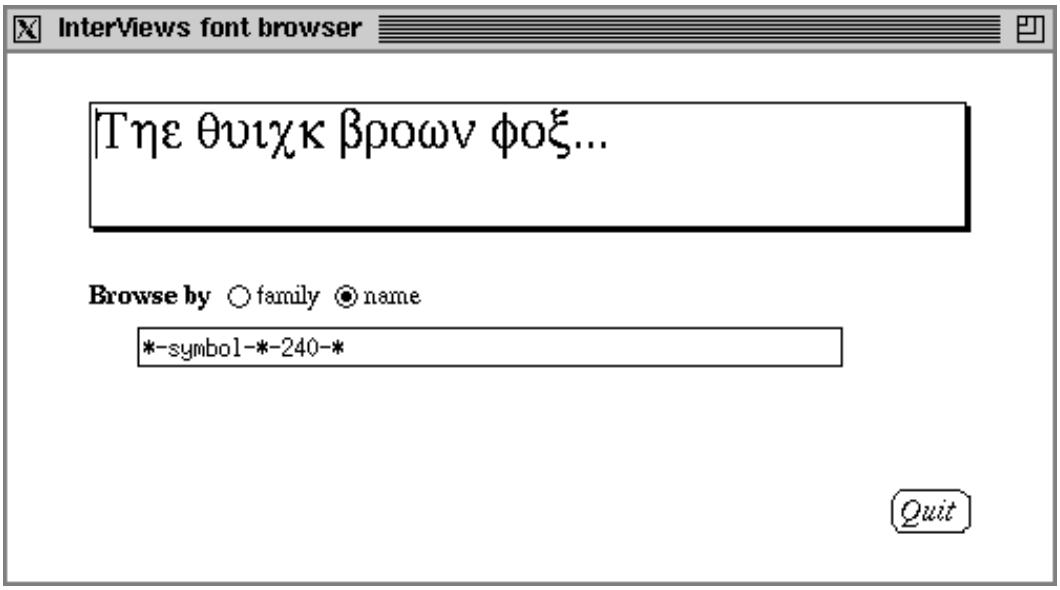
```
void StringButton::Reconfig () {
  RadioButton::Reconfig();
  shape->width += 4;
  shape->height += 4;
}

void StringButton::Resize () {
  int h = output->GetFont()->Height();
  offset = h;
  display->Resize(offset+2, 2, xmax-2, ymax-2, h, 0);
}

void StringButton::Redraw (...) {
  display->Redraw(output, canvas, l, b, r, t);
  output->Rect(canvas, offset, 0, xmax, ymax);
  Refresh();
}
```

Handle...

```
void StringButton::Handle (Event& e) {
  if (e.eventType == DownEvent && e.x >= offset) {
    hit = true;
    Refresh();
    boolean done = false;
    int p = strlen(text);
    display->CaretStyle(BarCaret);
    do {
      switch (e.eventType) {
      case KeyEvent:
        ...
        break;
      case DownEvent:
        if (e.target != this) {
          UnRead(e);
          done = true;
        }
        break;
      }
      display->Caret(0, p);
      if (!done) {
        Read(e);
      }
    } while (!done);
    display->CaretStyle(NoCaret);
    hit = false;
    Refresh();
    subject->SetValue(value);
    subject->Notify();
  } else {
    RadioButton::Handle(e);
  }
}
```

...more Handle

```
...
char c = e.keystring[0];
if (e.len > 0) {
  if (c == '\015') {
    done = true;
  } else if (c == '\177') {
    if (p > 0) {
      –p;
      text[p] = '\0';
      display->DeleteText(0, p, 1);
    }
  } else if (!iscntrl(c)) {
    text[p] = c;
    display->InsertText(0, p, &c, 1);
    ++p;
    text[p] = '\0';
  }
}
...
```



Browsingbyfontname,too.

Twobrowsingmodes

```
void FontBrowser::Init (const char* samp) {
  ...
  mode = new ButtonState(ModeChoices[0].value);
  ...
  fontname = new StringEditor(
    nil, 0, 0, "a very long font name..."
  );
  Tray* familymode = new Tray();
  Interactor* namemode = new VBox(
    new Frame(new MarginFrame(fontname, 2, 2)),
    new VGlue(0, 0, vfil)
  );

  modeoptions = new Deck();
  modeoptions->Insert(familymode);
  modeoptions->Insert(namemode);

  Insert(
    ...
    modeoptions,
    ...
  );
}
```

Flipping the Deck

```
boolean FontBrowser::Accept () {
  ...
  void* m;
  do {
    ...
    mode->GetValue(m);
    if (strcmp((char*)m, "family") == 0) {
      modeoptions->FlipTo(1);
    } else if (strcmp((char*)m, "name") == 0) {
      modeoptions->FlipTo(2);
    }
    ...
  } while (v == 0 && e.target != nil);
  return v == 1 || e.target == nil;
}
```

Calculating the fontname

```
void FontBrowser::UpdateFontname () {
  void* m;
  mode->GetValue(m);
  if (strcmp((char*)m, "family") == 0) {
    ...
    char name[1000];
    sprintf(name,
        "*-*-%s-%s-%s-%s-*-*-%s0-*-*-*-*-iso8859-1",
        fam, wgt, sl, wid, pnt
    );
    fontname->Message(name);
  }
  ...
}
```

Glyphs: Flyweight Objects for User Interfaces

The problem

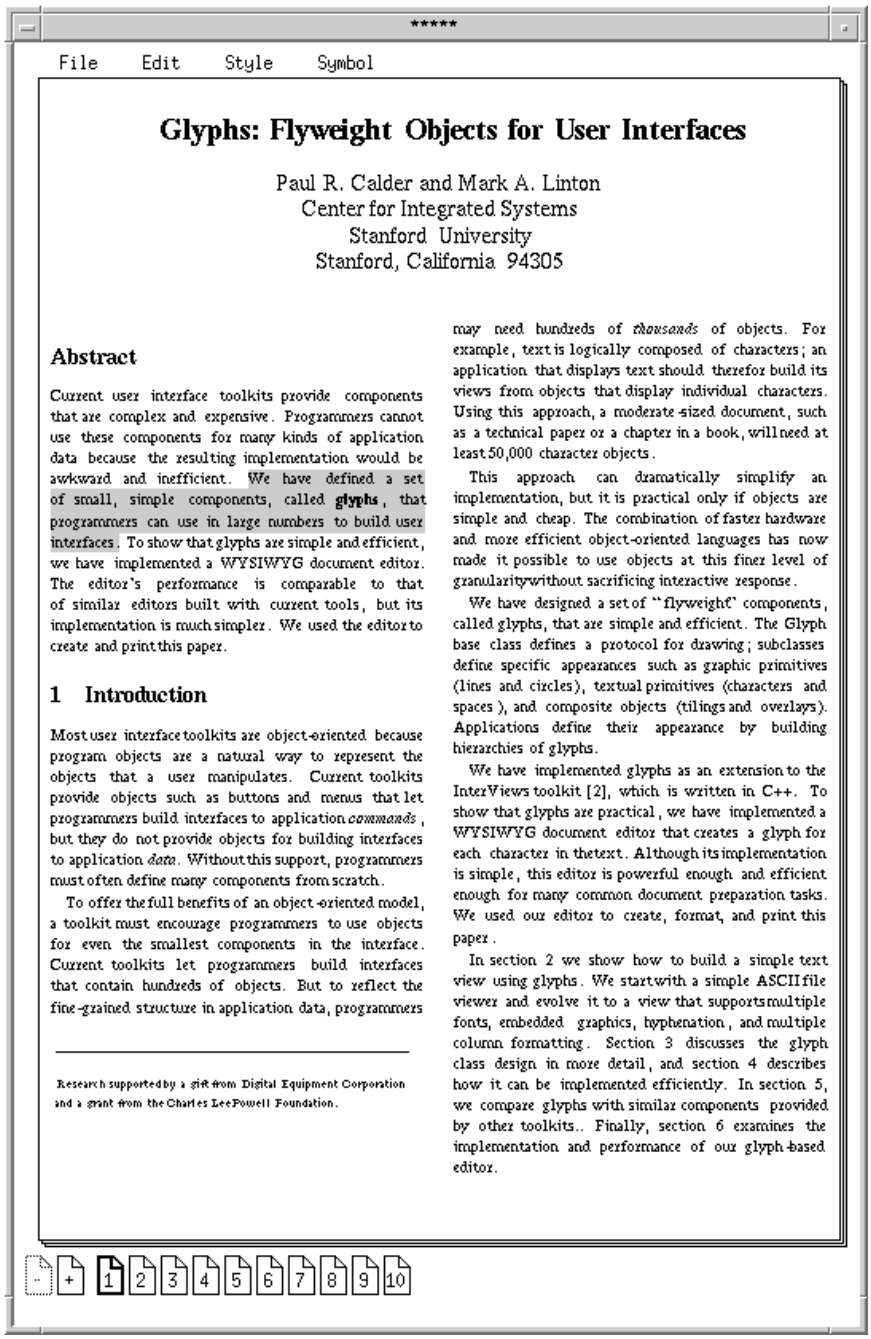
Objects are good...

- data as well as commands
- users think objects
- smallest components should be objects

...but current toolkits fall short

- gap between objects and widgets
- cannot build views of data

The experiment



The experiment

Build a document editor

- each character is an object
- graphics as well as text
- papers, thesis, slides

Why a document editor?

- familar—well understood
- high expectations—looks good, goes fast
- severe test—100,000 objects

More than just document editors

- views of application data
- smallest components are objects

Glyphs

Glyphs just define appearance

primtivies—line, circle, character, image
composites—box, deck

Glyphs are simple

Task	Methods
geometry	request, allocate
appearance	draw, pick
structure	insert, remove, change

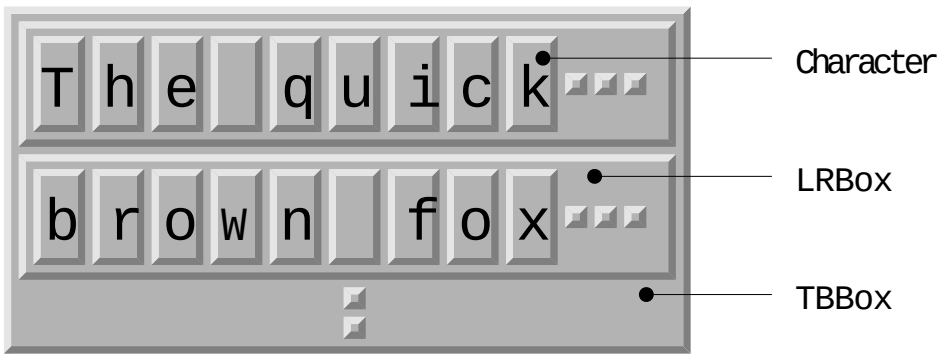
Glyphs are cheap

much cheaper than widgets

Toolkit must include more

handlers—event distribution, event mapping
interactors—buttons, mouse, scrollers

A text view...



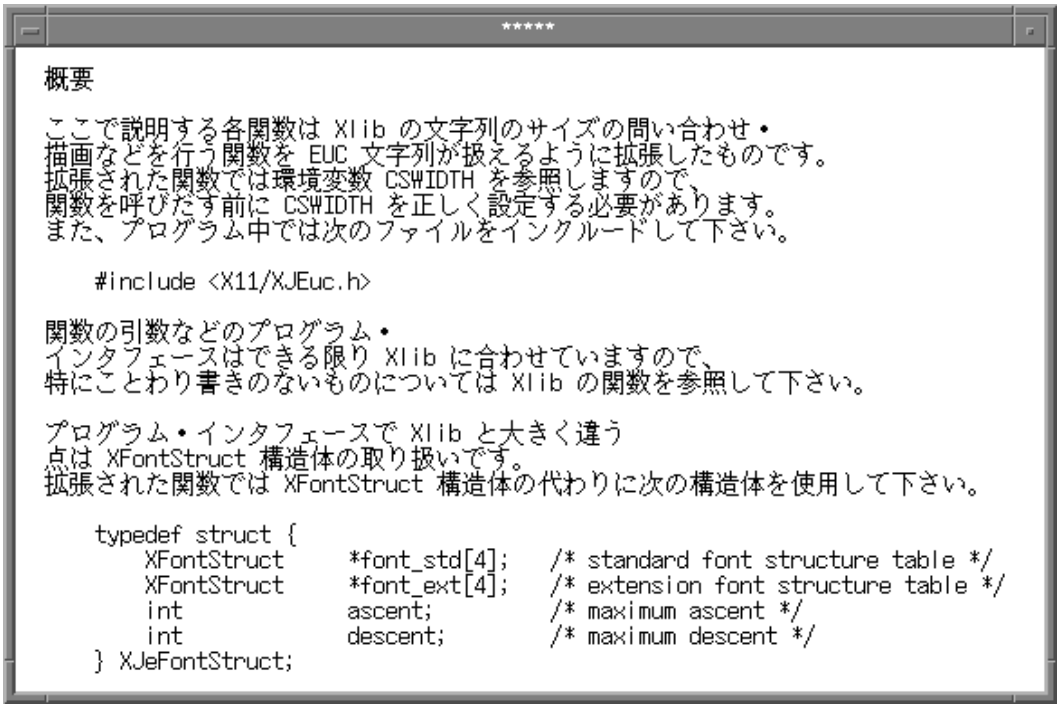
View structure reflects text structure

- glyph per character
- composite glyph per line

TextView code

```
TextView::TextView (FILE* file) {
  TBBox* page = new TBBox();
  LRBox* line = new LRBox();
  int c;
  while ((c = getc(file)) != EOF) {
    if (c == '\n') {
      page->append(line);
      line = new LRBox();
    } else {
      line->append(new Character(c));
    }
  }
  body(page)
}
```

..with multiple fonts..



*****

概要

ここで説明する各関数は Xlib の文字列のサイズの問い合わせ・
描画などを行う関数を EUC 文字列が扱えるように拡張したものです。
拡張された関数では環境変数 CSWIDTH を参照しますので、
関数を呼びだす前に CSWIDTH を正しく設定する必要があります。
また、プログラム中では次のファイルをインクルードして下さい。

```
#include <X11/XJEuc.h>
```

関数の引数などのプログラム・
インタフェースはできる限り Xlib に合わせていますので、
特にことわり書きのないものについては Xlib の関数を参照して下さい。

プログラム・インタフェースで Xlib と大きく違う
点は XFontStruct 構造体の取り扱いです。
拡張された関数では XFontStruct 構造体の代わりに次の構造体を使用して下さい。

```
typedef struct {
    XFontStruct     *font_std[4];   /* standard font structure table */
    XFontStruct     *font_ext[4];   /* extension font structure table */
    int             ascent;         /* maximum ascent */
    int             descent;        /* maximum descent */
} XJeFontStruct;
```

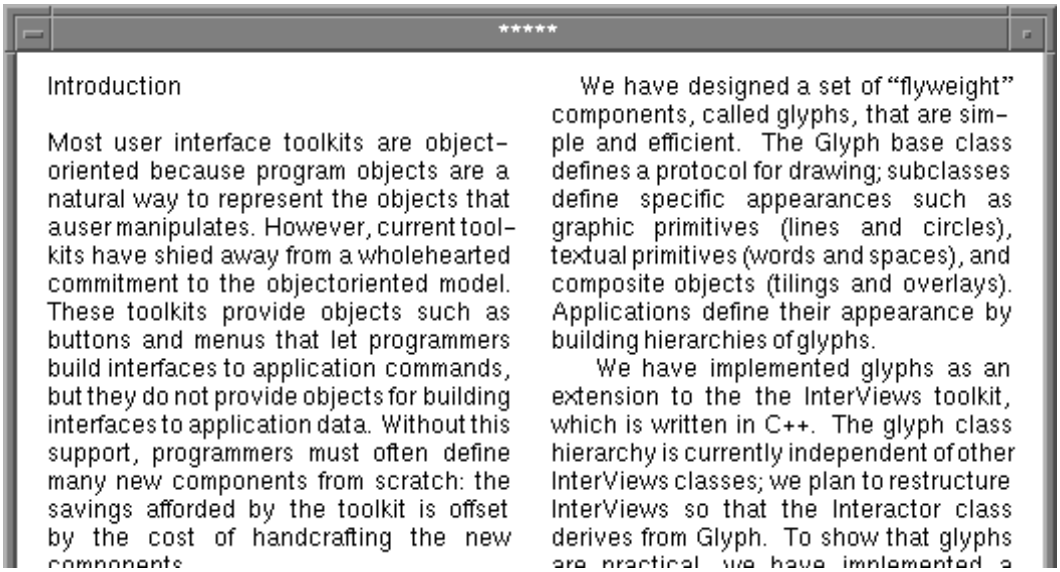
Same simple structure out..

- ASCII characters use ASCII-encoded font
- Kanji characters use JIS-encoded font

EUCTextViewCode

```
Font* kf;
Font* af;

while ((c = getc(file)) != EOF) {
  if (c == '\n') {
    page->append(line);
    line = new LRBox();
+ } else if (!isascii(c)) {
+   line->append(
      new Character(
        tojis(c, getc(file)), kf
      )
    );
  } else {
    line->append(new Character(c, af));
  }
}
```

. and formatting



Introduction

Most user interface toolkits are object-oriented because program objects are a natural way to represent the objects that a user manipulates. However, current toolkits have shied away from a wholehearted commitment to the objectoriented model. These toolkits provide objects such as buttons and menus that let programmers build interfaces to application commands, but they do not provide objects for building interfaces to application data. Without this support, programmers must often define many new components from scratch: the savings afforded by the toolkit is offset by the cost of handcrafting the new components.

We have designed a set of "flyweight" components, called glyphs, that are simple and efficient. The Glyph base class defines a protocol for drawing; subclasses define specific appearances such as graphic primitives (lines and circles), textual primitives (words and spaces), and composite objects (tilings and overlays). Applications define their appearance by building hierarchies of glyphs.

We have implemented glyphs as an extension to the the InterViews toolkit, which is written in C++. The glyph class hierarchy is currently independent of other InterViews classes; we plan to restructure InterViews so that the Interactor class derives from Glyph. To show that glyphs are practical, we have implemented a

High-level help from toolkit

line and column breaking
hyphenation

FormattedTextViewNode

```
Composition* page = new LRComposition(
  new TBBox(), new TeXCompositor(),
  nil, width
);

while ((c = getc(file)) != EOF) {
  if (c == '\n') {
    page->append(
      new Discretionary(
        PenaltyGood, new HGlue(fil),
        new HGlue(0, fil), nil, nil
      )
    );
  } else if (c == ' ') {
    page->append(
      new Discretionary(
        0, new Character(' '),
        new HGlue(0, 0), nil, nil
      )
    );
  } else {
    page->append(new Character(c));
  }
}
```

Implementation

Dynamic protocol...

- pass state rather than store it
- Character::draw(Canvas*, location)

...means small objects...

- objects needn't store graphics state
- Character need only store character code

...and fewer instances

- share objects
- one instance of Character('a') per document

Performance

Drawingspeed

Pagetype	Draw	Format
small	0.07	0.12
average	0.11	0.20
large	0.25	0.37

Secondsperpageona DECstation3100

Processsize

Document	Memoryusage
small	1500
medium	2300
large	4000

TotalKbytesallocated

Comparisons

Object size

Toolkit	Object	Size
Athena Widgets	Simple	132
Andrew Toolkit	view	44
ET++	VObject	32
Extended InterViews	Glyph	8

Sizes of base objects in bytes

Code size

Toolkit	Label	TextView
Athena Widgets	500	2000
Andrew Toolkit	300	3000
ET++	500	2000
Extended InterViews	10	20

Lines of code for views

Status

Prototype Toolkit

- glyph–primitives, decorators, composites
- layout–tiling, aligning
- compositor–array, split, (optimum fit)
- sensor–input interest
- handler–interactors

Document editor

- base editing (text, graphics, tables)
- figure prowing
- 5000 lines of code
- papers, slides, thesis

Merging Into InterViews

Conclusion

Toolkits can help build views of data

- not just command interface

Dynamic protocols for cheap objects

- think lots of objects

Leverage available technology

- already beyond interactive threshold