Programming User Interfaces in C++ with InterViews

Paul Calder

Computer Systems Laboratory
Stanford University
Stanford, California 94305

Copyright ©1989 by Paul R. Calder

Outline of Tutorial

An overview of InterViews and C++ (9:00 – 10:30)

- InterViews philosophy
- C++ at a glance
- InterViews and other toolkits

Programming a simple InterViews application (10:45 – 12:15)

- Bare bones
- Fleshing it out
- User customization

Lunch break

A detailed look at InterViews classes (1:45 – 3:15)

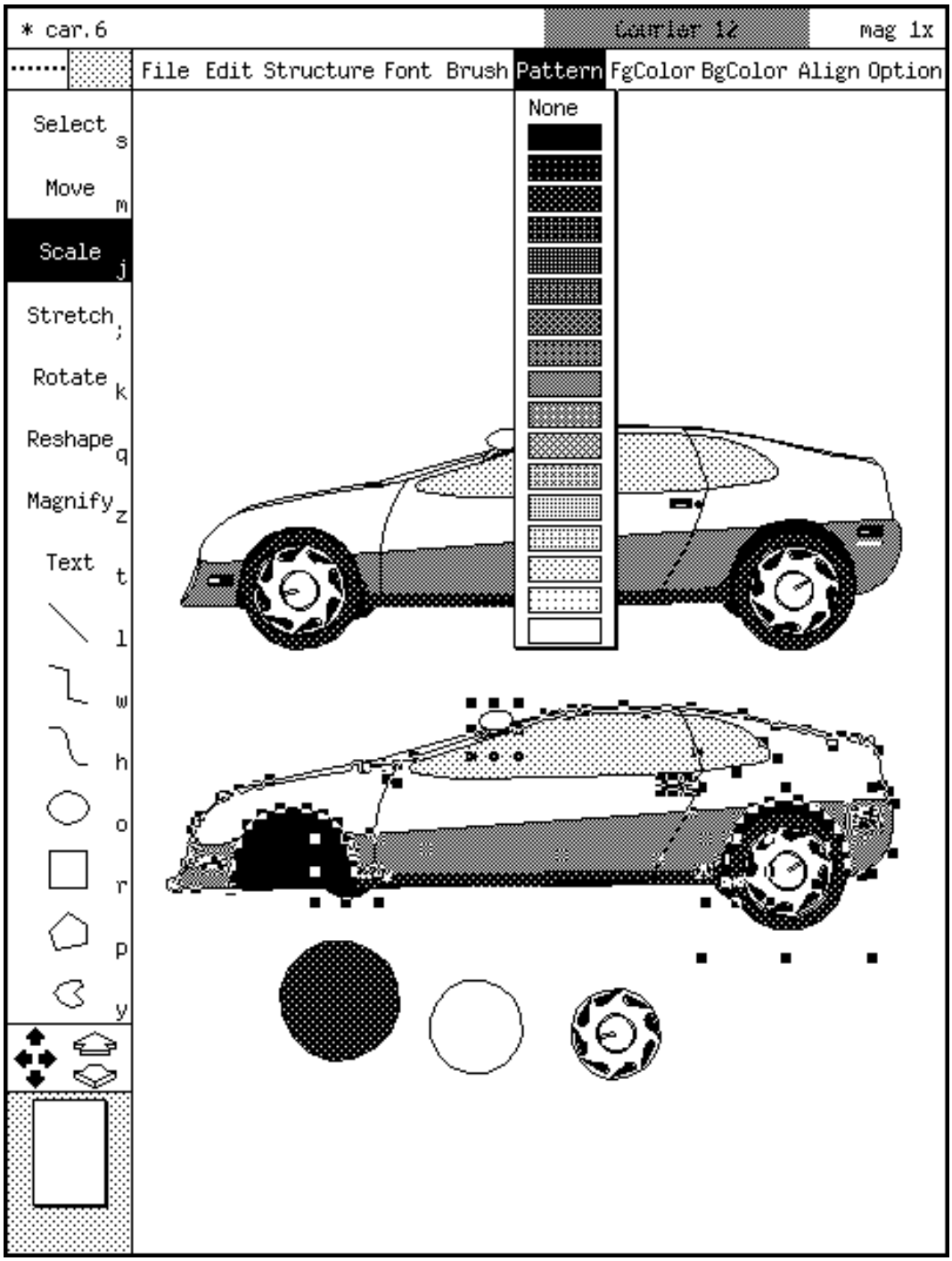
- Interactors, Painters, and Sensors
- Scenes and composition
- Subjects and views

A font browser application (3:30 – 5:00)

- Deriving new classes
- Final details

An overview of InterViews and C++

(9:00 – 10:30)



The InterViews drawing editor “idraw”.

User Interfaces

Different interface styles

- Standard guidelines
- Some users like different styles
- Consistency important too

Equal support for text and graphics

- Mixture of text and graphics is best
- Support below the "window" level

User interfaces should be object-oriented.

- Object-oriented interfaces are natural
- Users directly manipulate "real" objects

Abstract objects should be separate from interactive objects.

- We call these "subjects" and "views"
- Good for customization and update

User Interface Toolkits

Help for 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 work well with application programming environment

- InterViews is more than a toolkit

Window system should manage devices and concurrency

- The "operating system"
- Controls devices
- Allows concurrent applications

InterViews = Interactive Views

- Practical basis for our applications
  - Allegro, C++, X
    Reasonably fast, reliable, documented
- Keep it simple
- Composition
  - Scenes: Boxes and glue, Trays, ...
    Structured text
    Structured graphics
- Encapsulate window system
  - Object-oriented interface
    Higher-level graphics
    Multiple windows systems

Why C++?

- Object-oriented
- VERY portable
- Well integrated into the Unix environment
- Superset of C
- Efficient
- Reliable enough
- Easy to debug

C++ at a glance

Replace `struct` with `class`

Associate operations with classes

ptr−>Operation(args)

Allocate object with `new` *class*

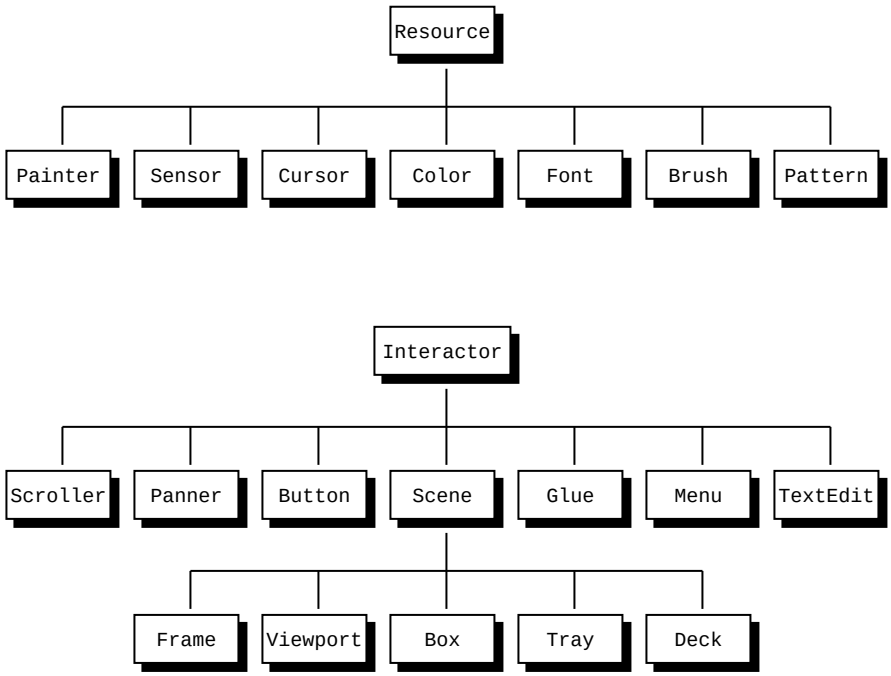
Free object with `delete` *object*

Inherit operations

Run-time binding of `virtual` operations

Overload operations by parameter types

use "`T& x`" for "`var x : T`"



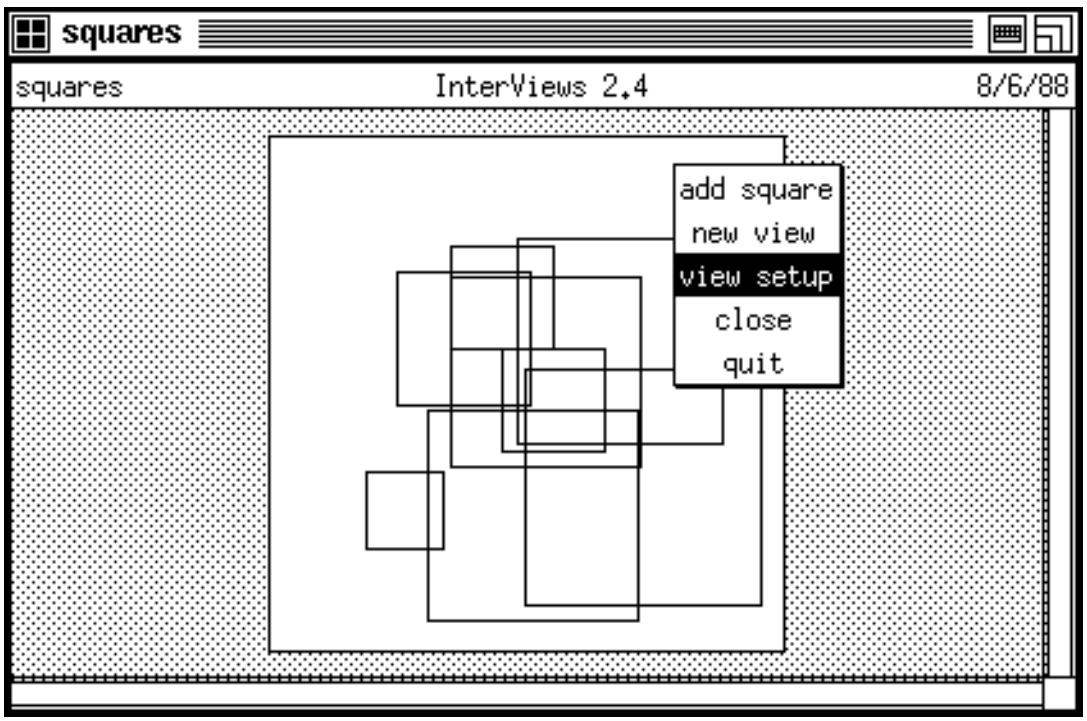
Basic InterViews object class hierarchy

How to keep it simple

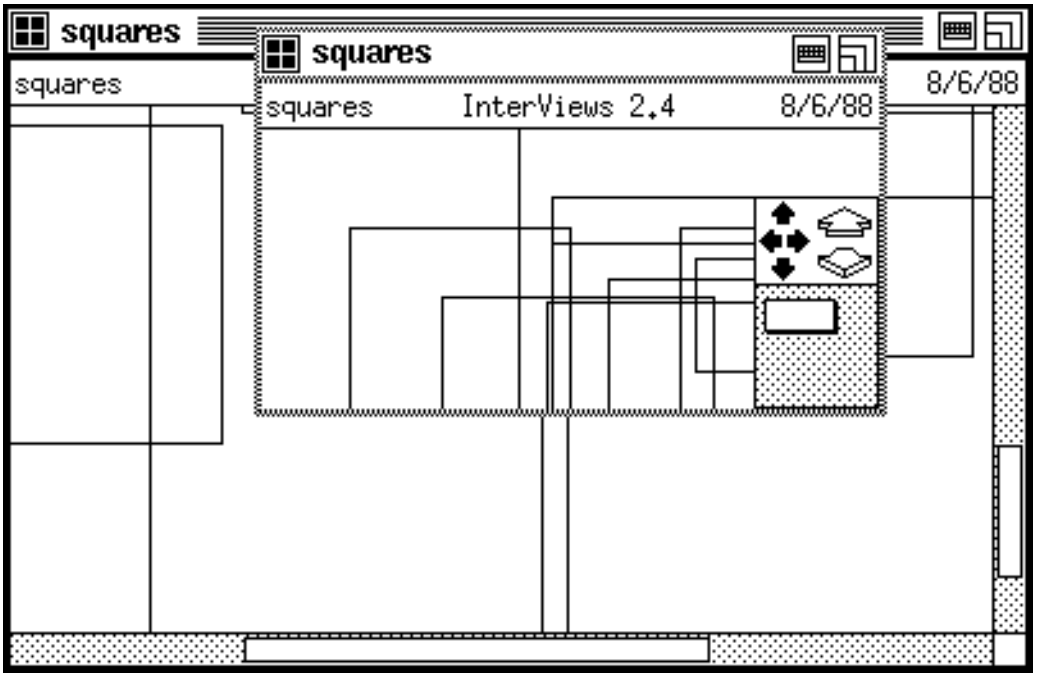
- Find the right abstractions
- Shallow class hierarchy
- When in doubt, let the application do it
- Make common case easy

What is similar about InterViews?

- Object-oriented
- Runs on top of X
- Class library
- Base interactive and composition classes
- Abstract/Interactive object split



Squares — Viewport, Menu, Scroller, Banner



Second squares view — Panner, subjects and views, Perspective

What is different about InterViews?

C++

Object-oriented, efficient

Composition mechanisms

Box, Tray, Frame, Viewport, Perspective

Graphics

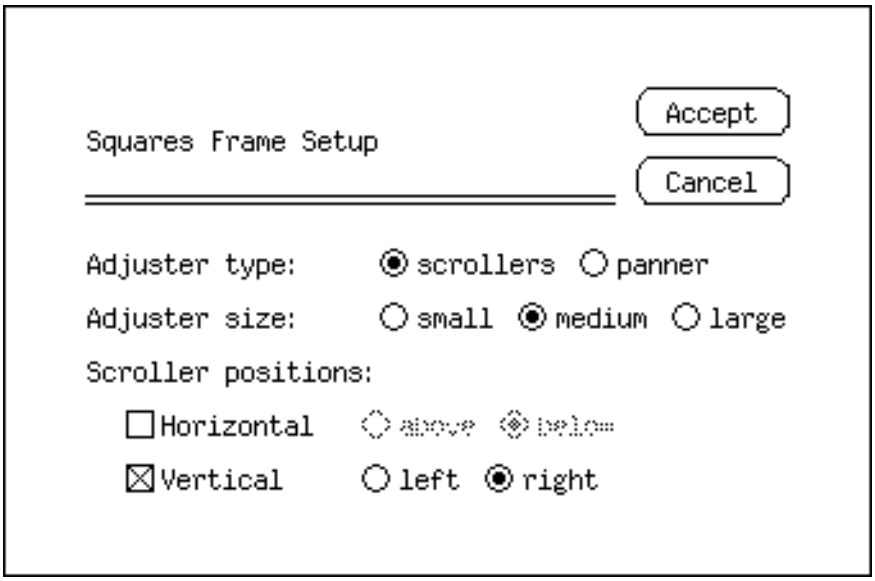
Painters, graphic library
idraw

Text

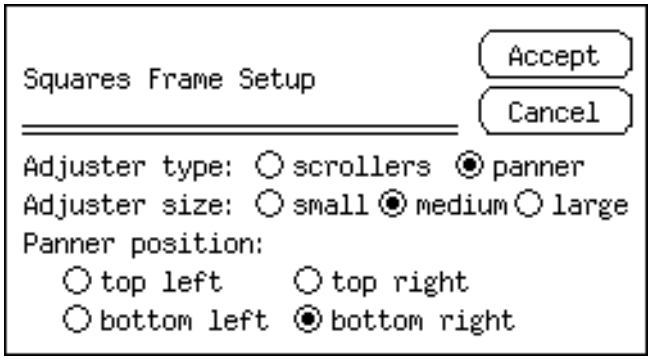
Composable primitives
Layout strategies

Part of a larger system

Component library
Programming environment
Applications



Squares dialog — PushButton, RadioButton, Checkbox, Message



Resized dialog — Box, Glue, Tray, Deck

InterViews status

Runs on VAXstations, Suns, ...

Library is about 30,000 lines of C++

Applications:

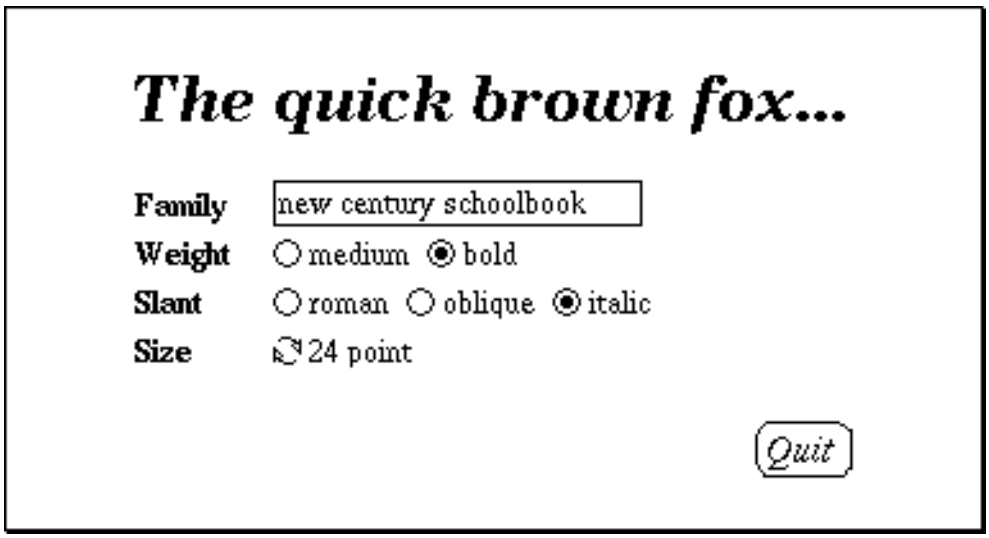
demos — squares, graphics, pages,
accessories — alert, dclock, istat, iwm, mailbox
applications — idraw,
research — ispy, inclink, insted, unidraw, ...

Publicly available

X11 tape
ftp from lurch.stanford.edu (36.22.0.14)
Email to `interviews@lurch.stanford.edu`

Programming a simple InterViews application

(10:45 – 12:15)



An InterViews font browser application

A simple InterViews application

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

Compilation details

InterViews is a library of classes

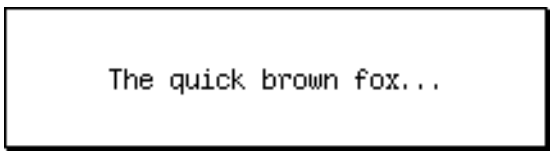
#include "<InterViews/*class*.h>"

CC -c application.c

CC application.o -lInterViews -lX -lm

The quick brown fox...

Just a Message



A ShadowFrame, Boxes, and Glue

An InterViews "hello, world"

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

void main () {
  World* world = new World();
  world->Insert(
    new Message("The quick brown fox...")
  );
  world->Run();
}
```

Basic framework

World

Opens connection with window system
Outer level ("root") window

Message

Static text display with inherited font

Build hierarchy bottom up

Don't need parent first
"declarative" specification

Event loop hidden in "Run"

Window maintenance events automatic

Framing the message

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

void main () {
  World* world = new World();
  world->Insert(
    new ShadowFrame(
      new HBox(
        new HGlue(round(0.5*inch)),
        new VBox(
          new VGlue(round(0.25*inch)),
          new Message("The quick brown fox..."),
          new VGlue(round(0.25*inch))
        ),
        new HGlue(round(0.5*inch))
      )
    )
  );
  world->Run();
}
```

Simple composition

Frame (ShadowFrame)

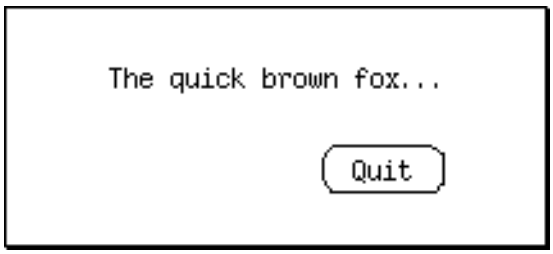
Decoration for a component
Different types available

Box (HBox, VBox)

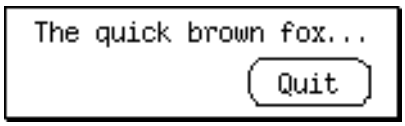
Tiled composition
Simplified TEX model
Left-to-right and top-to-bottom

Glue (HGlue, VGlue)

Natural, stretch, and shrink
Sizes in real world units

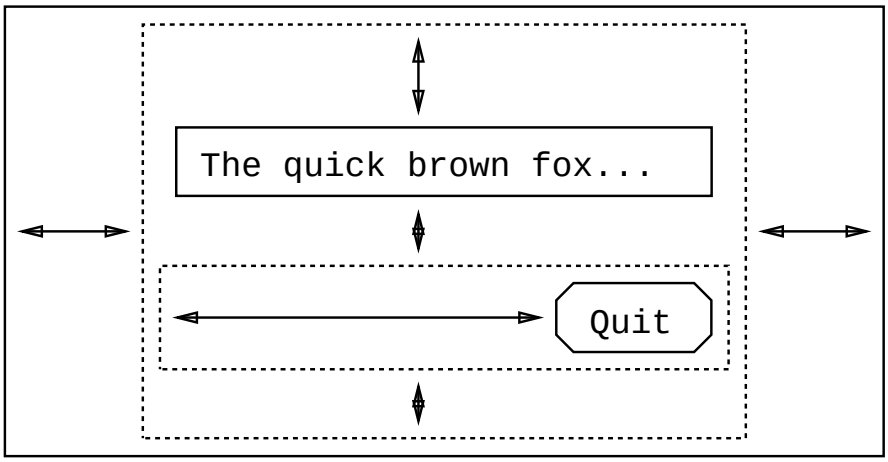


A button to push in a Dialog



See the glue shrink...

... and stretch



Arrangement of Boxes and Glue

Adding the button

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

void main () {
  World* world = new World();

  ButtonState* quit = new ButtonState(false);
  Dialog* dialog =  new Dialog(
    quit,
    new ShadowFrame(
      new HBox(
        new HGlue(0.5*inch, 0.4*inch, hfil/4),
        new VBox(
          new VGlue(0.25*inch, 0.2*inch, vfil/4),
          new Message("The quick brown fox..."),
          new VGlue(0.25*inch, 0.2*inch, 0),
          new HBox(
            new HGlue(),
            new PushButton("Quit", quit, true)
          ),
          new VGlue(0.25*inch, 0.2*inch, vfil/4)
        ),
        new HGlue(0.5*inch, 0.4*inch, hfil/4)
      )
    )
  );
  world->Insert(dialog);
  dialog->Accept();
}
```

Simple user interaction

ButtonState

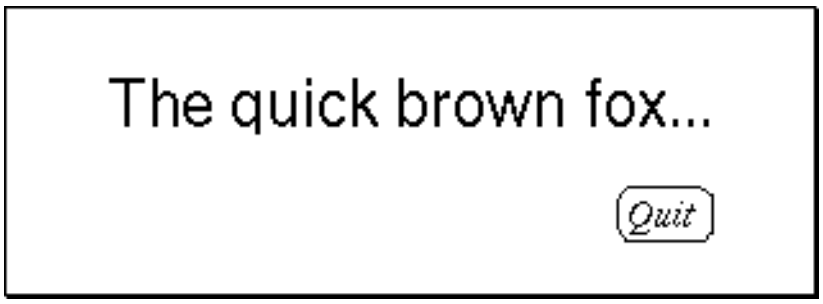
A subject (a piece of state)
A list of views (buttons)
Views notified when state changes

Button

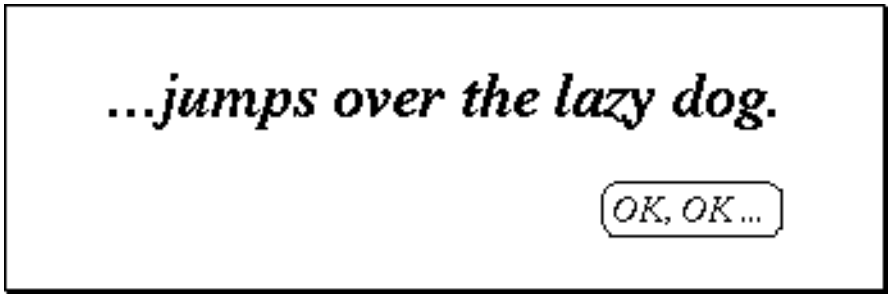
User pushes the button to change the state
Other types of button available

Dialog

Event loop inside
Exits when button state set to “false”



Customization ...



...for the font, sample, and button

Making it customizable

```
#include ...

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

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

void main (int argc, char* argv[]) {
  World* world = new World(
    "ifd", properties, options, argc, argv
  );

  const char* sample = world->GetAttribute("sample");
  const char* button = world->GetAttribute("button");

  ButtonState* quit = new ButtonState(false);
  Dialog* dialog =  new Dialog(
    ...
      new Message(sample),
      new PushButton(button, quit, true)
    ...
  );
  world->Insert(dialog);
  dialog->Accept();
}
```

Properties and customization

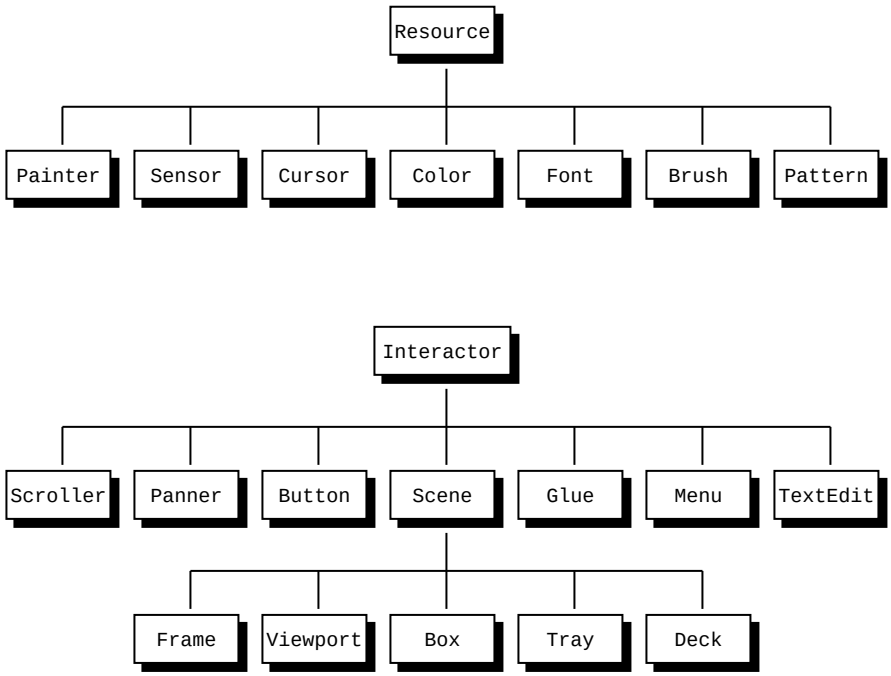
- Use properties to allow user customization
- Pass arguments to World
  - Default values in application code
  - Standard ".Xdefault" mechanism
  - Command line arguments
- PropertyData defines defaults
  - Hierarchical structure
  - Name/value specification
  - InterViews handles "standard" propeties
- OptionDesc controls command line parsing
  - Pattern/property/type
  - Matches removed from argument list

Summary

- Construct a world to initialize
- Insert top-level interactors into the world
- Specify layout with boxes, glue, frames
- Message for simple text
- Button for interactively setting state
- Dialog for input event loop
- Pass name, arguments to world

A detailed look at InterViews classes

(1:45 – 3:15)



Basic InterViews object class hierarchy

Interactors

- Base class for all interactive objects
- Abstract class–don't create instances
- Shape specifies desired size
- Canvas represents actual screen area
- Sensor defines input interest
- Painter performs output
- Top-level attributes

Shapes

int width, height

int hshrink, vshrink

int hstretch, vstretch

int hunits, vunits

int aspect

const int hfil, vfil for “infinite” stretchability

Canvases

- Normally X windows
- Interactor::SetCanvasType
  - CanvasShapeOnly, CanvasInputOnly, CanvasInputOutput, CanvasSaveUnder, CanvasSaveContents, CanvasSaveBoth
- Lower left (0,0)
- Upper right (xmax, ymax)
- Interactor::Resize called when canvas is set
- new Canvas(width, height) for off-screen

Configuration

- Usually don't know parent in constructor
- Attribute lookup requires path to world
- When ancestors set:
  - Standard output painter customized
  - Children configured
  - Reconfig called
- GetAttribute retrieves user-specified value
- Instances can be named or inherit class name

PropertySheet

- Tree of directories and attributes
- Attributes defined by string name and value
- Defaults for common attributes

Interactor Output Operations

Painter* output

Standard output for interactor

Redraw

Called when an area of an interactor should be drawn. Usually a side effect of reading input. Default is to do nothing.

RedrawList

Called when several areas should be drawn. Default is to call Redraw on each area.

Draw

Display an interactor and its components. Used to update an interactor after data has changed. Default is Redraw(0, 0, xmax, ymax).

Flush

Make sure output requests have been sent to server.

Sync

Wait for previous requests to complete.

Painters

- Brush
- Color (foreground and background)
- Font
- Pattern
- Fill fg or fg+bg
- Transformer
- Graphics operations on Canvas

Examples:

```
Painter* output;

output->SetBrush(new Brush(0xffff, 1));
output->SetPattern(new Pattern(0xa5a5));
output->FillBg(false);

Font* f = new Font("*-times-medium-r-*-180-*");
if (f->Valid()) {
    output->SetFont(f);
}

Color* c = new Color("blue");
if (c->Valid()) {
    output->SetColors(fg, nil);
}
```

Coordinates

- typedef int Coord
- Context-dependent units
- Explicit: round(3.5∗inches)
- Scale by precision
- Rotate by angle
- Ultimately translated to pixels for X

Examples:

```
Coord x[10], y[10];

output->Scale(.01*inch);
output->FillRect(canvas, 100, 150, 200, 250);
output->Rotate(15.0);
output->FillBSpline(canvas, x, y, 10);
output->Text(canvas, "a string", 175, 175);
```

Input Events

Motion, Up/Down, Key, On/Off, Timer, Channel

Target interactor (not necessarily reader)

Canvas coordinates relative to target

Button id

ASCII string mapping

Meta-key states

Sensors

```
void Catch(EventType)
void Ignore(EventType)

void CatchButton(EventType, int)
void IgnoreButton(EventType)

void CatchTimer(int sec, int usec)

void CatchChannel(int)
void IgnoreChannel(int)
```

Interactor Input Operations

Sensor* input

Standard event interest

void Read(Event&)

Take the next event from the input queue.

void UnRead(Event&)

Put event back on queue as if it had not been read.

virtual void Handle(Event&)

Receive an event from a dispatcher. Handle allows event-driven processing; Read allows procedural.

void Run()

Execute a simple dispatch loop, handling events to their targets. The loop terminates when a handler sets its event target to nil.

void Poll(Event&)
boolean Check()
void Listen(Sensor*)

Scenes

- Base class for compositions
- Abstract class
- Shape calculated from components
- Allocates (sub)canvases to components
- Variety of different composition strategies

Scene Operations

```
void Insert(Interactor*)
void Insert(
  Interactor*, Coord left, Coord bottom
)
void Remove(Interactor*)

void Move(Coord left, Coord bottom)
void Raise(Interactor*)
void Lower(Interactor*)
```

Change propagation

Parent()->Change(this)
void Propagate(boolean = true)
If propagate then recompute parent's shape
World requests to resize top-level window
Resize called on affected parts of hierarchy

Must call change after insert(s) into mapped scene.

Implementing scenes

```
virtual void DoInsert(
    Interactor*, boolean, Coord&, Coord&
)
virtual void DoChange(Interactor*)
virtual void DoRemove(Interactor*)

void UserPlace(Interactor*)

void Place(
  Interactor*,
  Coord x1, Coord y1, Coord x2, Coord y2,
  boolean mapped = true
)

void Map(Interactor*, boolean raised = true)
void Unmap(Interactor*)
```

Interactor hierarchy

```
Scene* Parent()

void GetRelative(
  Coord& x, Coord& y, Interactor*
)

virtual void GetComponents(
  Interactor**, int, Interactor**&, int&
)
```

Deleting a scene implies deleting all of its components

MonoScene

- Single component
- Useful for implementation (Menu, Panner)
- Output decoration (Frame, Viewport)
- Input handling (Dialog)
- Basis of many application-specific classes

Box

- Tiles interactors side-by-side
- Stretches/shrinks to fit available space
- HBox for left-to-right
- VBox for top-to-bottom
- Natural size is sum along major axis, maximum along minor axis
- Stretchability is min of max, shrinkability is max of min

Tray

- Overlaps/constraints on components
- Background interactor defines shape
- Alignment constraints among interactors
- Specifies edge alignments
- TGlue represents white space between edges

Deck

- Stack of interactors
- Only one visible at a time
- Flip(n), FlipTo(position) to select

Frame

- Default is solid outline
- BorderFrame grays without input focus
- ShadowFrame is thicker on two sides
- TitleFrame has banner
- Frame(Interactor*, int, int, int, int)

Viewport

- Allocates component as much shape as it wants
- Component is clipped to viewport canvas
- Perspective defines portion of component actually displayed

World

- Root scene
- Construct to initialize
  - Program defaults
  - User defaults
  - Command-line arguments
- Control/access display characteristics
  - Ring bell
  - Set keyboard, mouse parameters
  - Current screen

Standard Interactor Library

Adjuster - control for perspective

Banner - title bar

Bitmap - image data

Border - separator

Button - view of ButtonState

Dialog - popup dialog input handler

Menu - popup menus

Panner - two-dimensional scrolling and zooming

Rubberband - interactive dragging and placement

Scroller - scroll bar

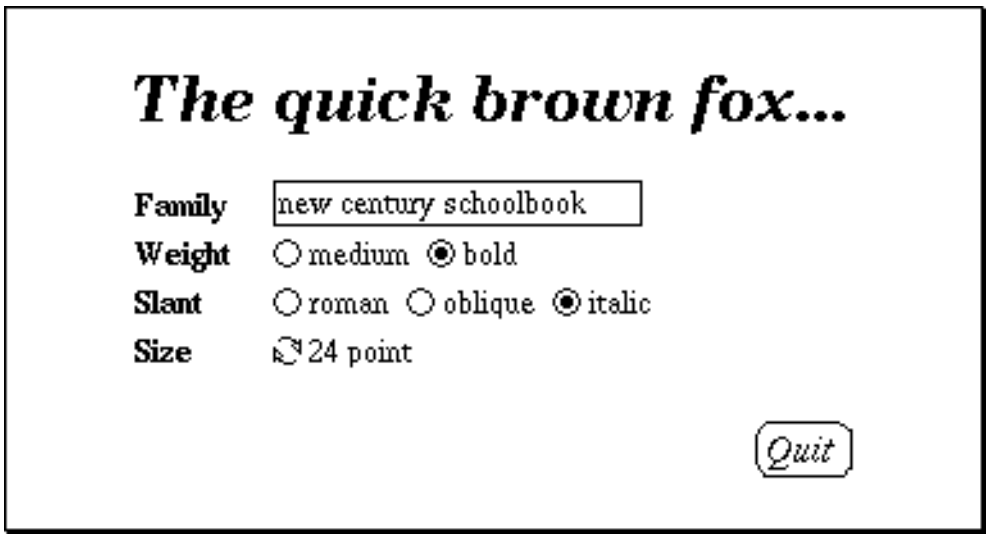
StringEdit - one line text editing

TextViewer - multi-line text editing

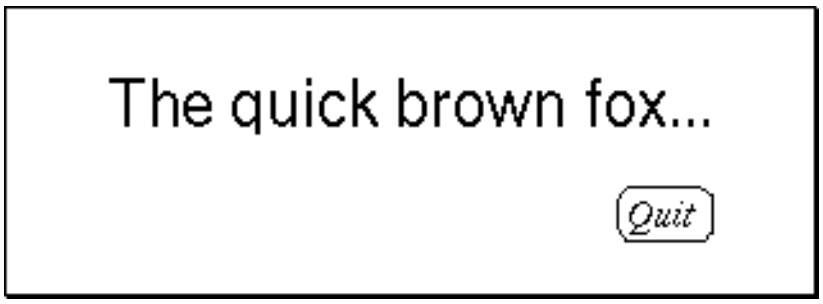
WorldView - window managers

A font browser application

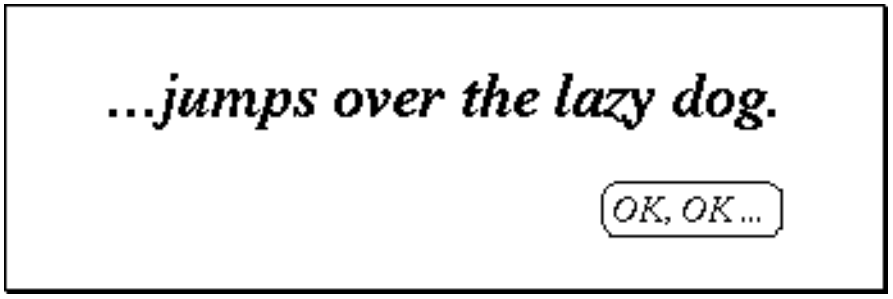
(3:30 – 5:00)



An InterViews font browser application



Customization ...



...for the font, sample, and button

Buttons to select the font

The font display uses the same button state

The code as it stands

```
#include ...

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

void main (int argc, char* argv[]) {
  World* world = new World("ifd", ...);
  ...
  ButtonState* quit = new ButtonState(false);
  Dialog* dialog =  new Dialog(
    quit,
    new ShadowFrame(
      ...
    )
  );
  world->Insert(dialog);
  dialog->Accept();
}
```

Adding font choices

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

#include "FontDisplay.h"

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

void main (int argc, char* argv[]) {
  World* world = new World("ifd", ...);
  ...
  Font* helvetica = new Font(
    "*-helvetica-medium-r-*-240-*"
  );
  Font* times = new Font(
    "*-times-medium-r-*-240-*"
  );
  Font* courier = new Font(
    "*-courier-medium-r-*-240-*"
  );

  ButtonState* quit = new ButtonState(false);
  ButtonState* family = new ButtonState(helvetica);

  ...
}
```

Using a Tray for the layout

```
FontDisplay* display = new FontDisplay(
  sample, family
);
Interactor* familylabel = new Message(
  "label", "Family"
);
Interactor* familychoices = new HBox(
  new RadioButton("helvetica", family, helvetica),
  new HGlue(0.05*inch, 0.05*inch, 0),
  new RadioButton("times", family, times),
  new HGlue(0.05*inch, 0.05*inch, 0),
  new RadioButton("courier", family, courier),
  new HGlue(0, 0, hfil)
);
Interactor* quitter = new PushButton(
  button, quit, true
);

Tray* t = new Tray();
t->HBox(...);
t->VBox(...);
t->Align(...);
t->Align(...);

Dialog* dialog =  new Dialog(
  quit, new ShadowFrame(t)
);
```

Specifying Tray alignments

```
t->HBox(
  t,
  new HGlue(0.5*inch, 0.4*inch, hfil/4),
  display,
  new HGlue(0.5*inch, 0.4*inch, hfil/4),
  t
);
t->VBox(
  t,
  new VGlue(0.25*inch, 0.2*inch, vfil/4),
  display,
  new VGlue(),
  quitter,
  new VGlue(0.25*inch, 0.2*inch, vfil/4),
  t
);
t->HBox(
  familylabel,
  new HGlue(0.2*inch, 0.1*inch, 0),
  familychoices
);
t->VBox(
  display,
  new VGlue(0.25*inch, 0.2*inch, 0),
  familylabel,
  new VGlue(0.25*inch, 0.2*inch, 0),
  quitter
);
t->Align(Right, display, quitter);
t->Align(Left, display, familylabel);
t->Align(Right, display, familychoices);
t->Align(VertCenter, familylabel, familychoices);
```

A new class definition

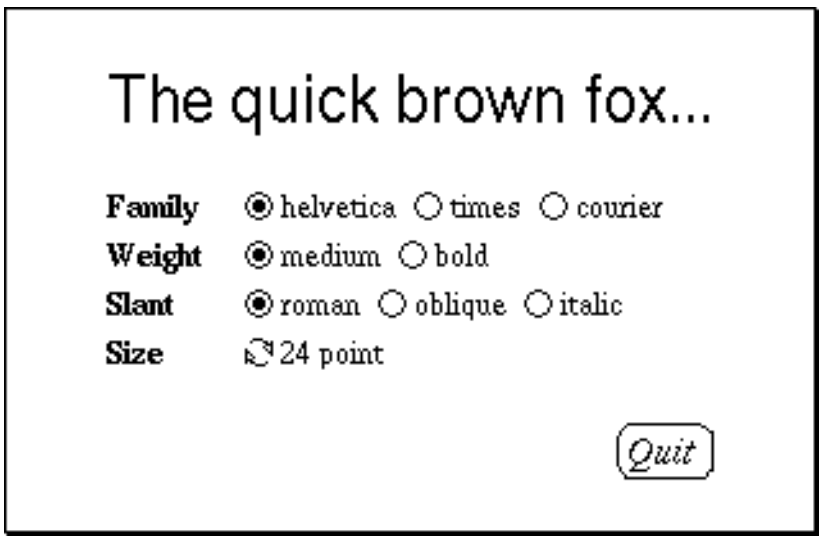
```
#ifndef FontDisplay_h
#define FontDisplay_h

#include <InterViews/scene.h>

class ButtonState;

class FontDisplay : public MonoScene {
public:
  FontDisplay(
    const char* sample, ButtonState* font
  );
  virtual void Reconfig();
  virtual void Update();
protected:
  virtual void Delete();
private:
  ButtonState* font;
};

#endif
```

Allocation and deallocation

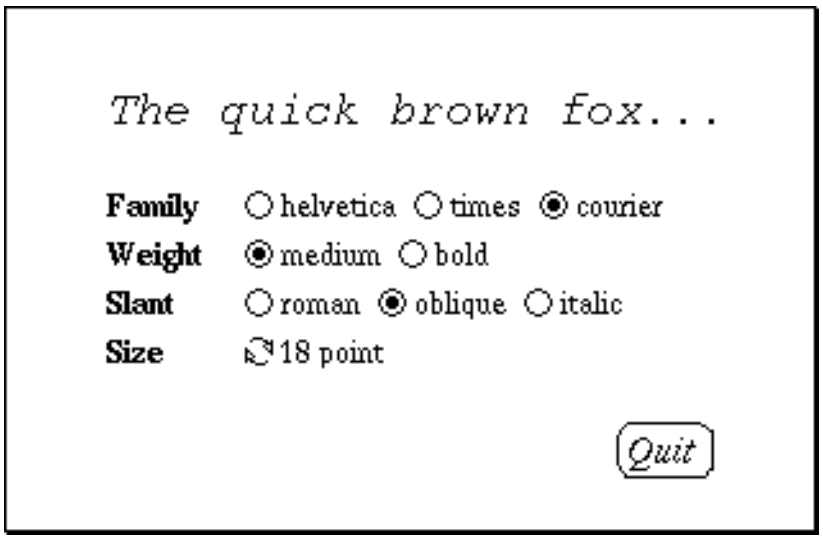
```
#include "FontDisplay.h"

#include <InterViews/button.h>
#include <InterViews/painter.h>
#include <InterViews/message.h>
#include <InterViews/shape.h>

class Font;

FontDisplay::FontDisplay(
  const char* sample, ButtonState* f
) : () {
  font = f;
  font->Attach(this);
  SetClassName("FontDisplay");
  Insert(new Message(sample, CenterLeft));
  output = new Painter();
  output->SetFont(nil);
  output->FillBg(false);
  Update();
}

void FontDisplay::Delete () {
  font->Detach(this);
  MonoScene::Delete();
}
```

Configuration and update

```
void FontDisplay::Reconfig () {
  MonoScene::Reconfig();
  shape->Rigid(shape->width, hfil, 0, 0);
}

void FontDisplay::Update () {
  Font* f;
  f = output->GetFont();
  //delete f;
  font->GetValue(f);
  output->SetFont(f);
  Draw();
}
```



Choices for font styles...



...and a new type of button.

Font style choices and a new type of button

```
#include ...

#include "CycleButton.h"

...

void main (int argc, char* argv[]) {
  World* world = new World("ifd", ...);
  ...

  char* initfamily = "helvetica";
  char* initweight = "medium";
  char* initslant = "r";
  int initsize = 24;

  ButtonState* quit = new ButtonState(false);
  ButtonState* family = new ButtonState(initfamily);
  ButtonState* weight = new ButtonState(initweight);
  ButtonState* slant = new ButtonState(initslant);
  ButtonState* size = new ButtonState(initsize);

  FontDisplay* display = new FontDisplay(
    sample, family, weight, slant, size
  );
```

Building the components

```
Message* familylabel = new Message(
  "label", "Family"
);
Box* familychoices = new HBox(
  new RadioButton("helvetica", family, initfamily),
  new HGlue(0.05*inch, 0.05*inch, 0),
  new RadioButton("times", family, "times"),
  new HGlue(0.05*inch, 0.05*inch, 0),
  new RadioButton("courier", family, "courier"),
  new HGlue(0, 0, hfil)
);
Message* weightlabel = new Message("label", "Weight");
Box* weightchoices = new HBox(...);
Message* slantlabel = new Message("label", "Slant");
Box* slantchoices = new HBox(...);
Message* sizelabel = new Message("label", "Size");
CycleButton* sizechoices = new CycleButton(
  size, "24 point", (void*)initsize
);
sizechoices->AddChoice("18 point", 18);
sizechoices->AddChoice("14 point", 14);
sizechoices->AddChoice("12 point", 12);
sizechoices->AddChoice("10 point", 10);
sizechoices->AddChoice("8 point", 8);
Button* quitter = new PushButton(
  button, quit, true
);
```

Specifying tray alignments

```
t->Align(Right,
  display, quitter
);
t->Align(Left,
  display, familylabel, weightlabel,
  slantlabel, sizelabel
);
t->Align(Left,
  familychoices, weightchoices,
  slantchoices, sizechoices
);
t->Align(VertCenter,
  familylabel, familychoices
);
t->Align(VertCenter,
  weightlabel, weightchoices
);
t->Align(VertCenter,
  slantlabel, slantchoices
);
t->Align(VertCenter,
  sizelabel, sizechoices
);
```

Changes to FontDisplay class

```
#ifndef FontDisplay_h
#define FontDisplay_h

#include <InterViews/scene.h>

class ButtonState;

class FontDisplay : public MonoScene {
public:
  FontDisplay(const char* sample,
    ButtonState* family, ButtonState* weight,
    ButtonState* slant, ButtonState* size
  );
  ...
private:
  ButtonState* family;
  ButtonState* weight;
  ButtonState* slant;
  ButtonState* pointsize;
};

#endif
```

Changes to update

```
void FontDisplay::Update () {
    char* f;
    family->GetValue(f);
    char* w;
    weight->GetValue(w);
    char* s;
    slant->GetValue(s);
    int p;
    pointsize->GetValue(p);
    sprintf(
      fontname, "*-%s-%s-%s-*-%d-*", f, w, s, p*10
    );
    Font* font = new Font(fontname);
    if (font->Valid()) {
        //delete output->GetFont();
        output->SetFont(font);
        Draw();
    }
}
```

Definition of CycleButton class

```
#ifndef CycleButton_h
#define CycleButton_h

#include <InterViews/button.h>

class Bitmap;

class CycleButton : public Button {
public:
  CycleButton(
    ButtonState*, const char* label, void* value
  );

  void AddChoice(const char*, void*);
  void AddChoice(const char*, int);

  virtual void Delete();
  virtual void Reconfig();
  virtual void Resize();
  virtual void Redraw(Coord, Coord, Coord, Coord);
  virtual void Refresh();
  virtual void Press();
  virtual void Update();
private:
  int size;
  int count;
  int current;
  const char** labels;
  void** values;
  Bitmap* plainBits;
  Bitmap* hitBits;
};

#endif
```

Allocation and deallocation

```
#include "CycleButton.h"

#include <InterViews/bitmap.h>
#include <InterViews/font.h>
#include <InterViews/shape.h>
#include <InterViews/painter.h>

static const int bits_size = 11;
static char plain_bits[] = {
   0x78, 0x00, 0x84, 0x05, 0x00, 0x07, ...
};
static char hit_bits[] = {
   0x78, 0x00, 0x84, 0x05, 0x00, 0x07, ...
};

static const int separator = 3;

CycleButton::CycleButton (
  ButtonState* subject, const char* label, void* value
) : (subject, nil) {
  SetClassName("CycleButton");
  size = 0;
  labels = nil;
  values = nil;
  count = 0;
  current = 0;
  AddChoice(label, value);
}

void CycleButton::Delete () {
  delete labels;
  delete values;
  Button::Delete();
}
```

Adding more choices

```
void CycleButton::AddChoice (
  const char* label, void* value
) {
  if (count >= size) {
    size += 10;
    const char** newlabels = new const char* [size];
    void** newvalues = new void* [size];
    for (int i = 0; i < count; ++i) {
      newlabels[i] = labels[i];
      newvalues[i] = values[i];
    }
    delete labels;
    labels = newlabels;
    delete values;
    values = newvalues;
  }
  labels[count] = label;
  values[count] = value;
  ++count;
}

void CycleButton::AddChoice (
  const char* label, int value
) {
  AddChoice(label, (void*)value);
}
```

Configuration

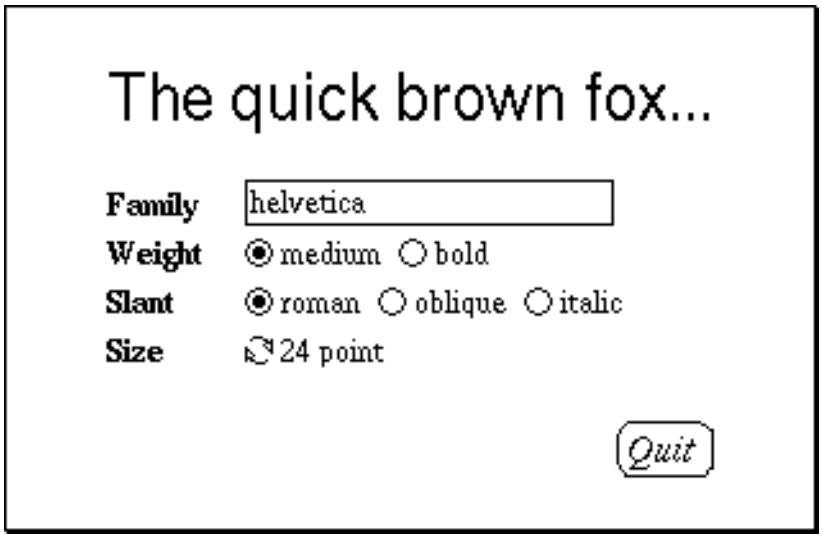
```
void CycleButton::Reconfig () {
  Font* f = output->GetFont();
  int h = f->Height();
  int w = 0;
  for (int i = 0; i < count; ++i) {
    w = max(w, f->Width(labels[i]));
  }
  shape->Rect(w + h + separator, h);
  shape->Rigid();
  Color* fg = output->GetFgColor();
  Color* bg = output->GetBgColor();
  plainBits = new Bitmap(
    plain_bits, bits_size, bits_size
  );
  plainBits->SetColors(fg, bg);
  hitBits = new Bitmap(
    hit_bits, bits_size, bits_size
  );
  hitBits->SetColors(fg, bg);
}
```

Resize and Redraw

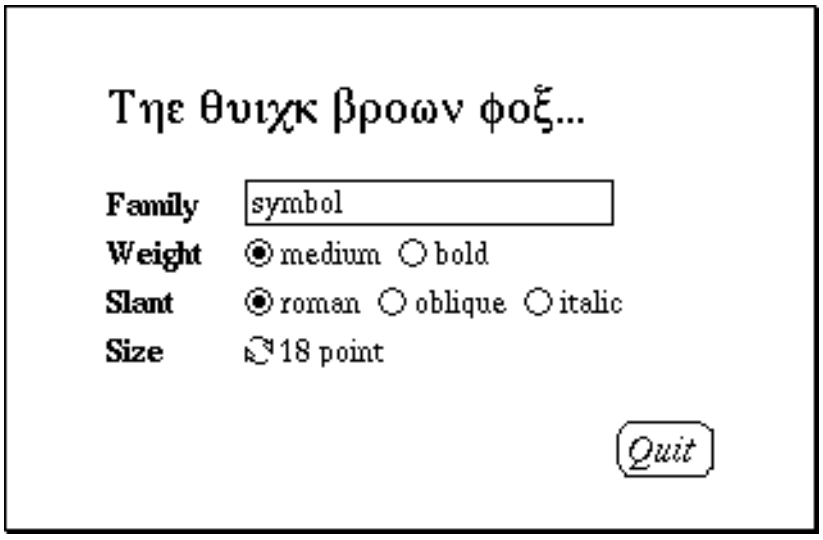
```
void CycleButton::Resize () {
  Coord bx = 0;
  Coord by = (ymax+1 - bits_size)/2;
  plainBits->SetOrigin(bx, by);
  hitBits->SetOrigin(bx, by);
}

void CycleButton::Redraw (
  Coord l, Coord b, Coord r, Coord t
) {
  Coord tx = bits_size + separator;
  Coord ty =
    (ymax + 1 - output->GetFont()->Height())
    / 2;
  output->ClearRect(canvas, l, b, r, t);
  output->Text(canvas, labels[current], tx, ty);
  Refresh();
}

void CycleButton::Refresh () {
  if (!hit) {
    plainBits->Draw(canvas);
  } else {
    hitBits->Draw(canvas);
  }
}
```

Button actions

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

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



An editable font family name...



...and we're done.

Editable font family name

```
#include ...

#include "StringButton.h"

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

void main (int argc, char* argv[]) {
  World* world = new World("ifd", ...);
  ...
  Interactor* familychoices = new Frame(
    new StringButton(
      family, initfamily,
      "a very long font family name"
    )
  );
  ...
  Dialog* dialog =  new Dialog(
    quit, new ShadowFrame(t)
  );
  dialog->Listen(allEvents);
  world->Insert(dialog);
  dialog->Accept();
  fprintf(stdout, "%s\n", display->FontName());
}
```

StringButton class definition

```
#ifndef StringButton_h
#define StringButton_h

#include <InterViews/Text/stringedit.h>

class ButtonState;

class StringButton : public StringEdit {
public:
  StringButton(
    ButtonState*,
    const char* value, const char* sample
  );

  virtual void Delete();
  virtual void Update();
  virtual void Handle(Event&);
protected:
  ButtonState* subject;
};

#endif
```

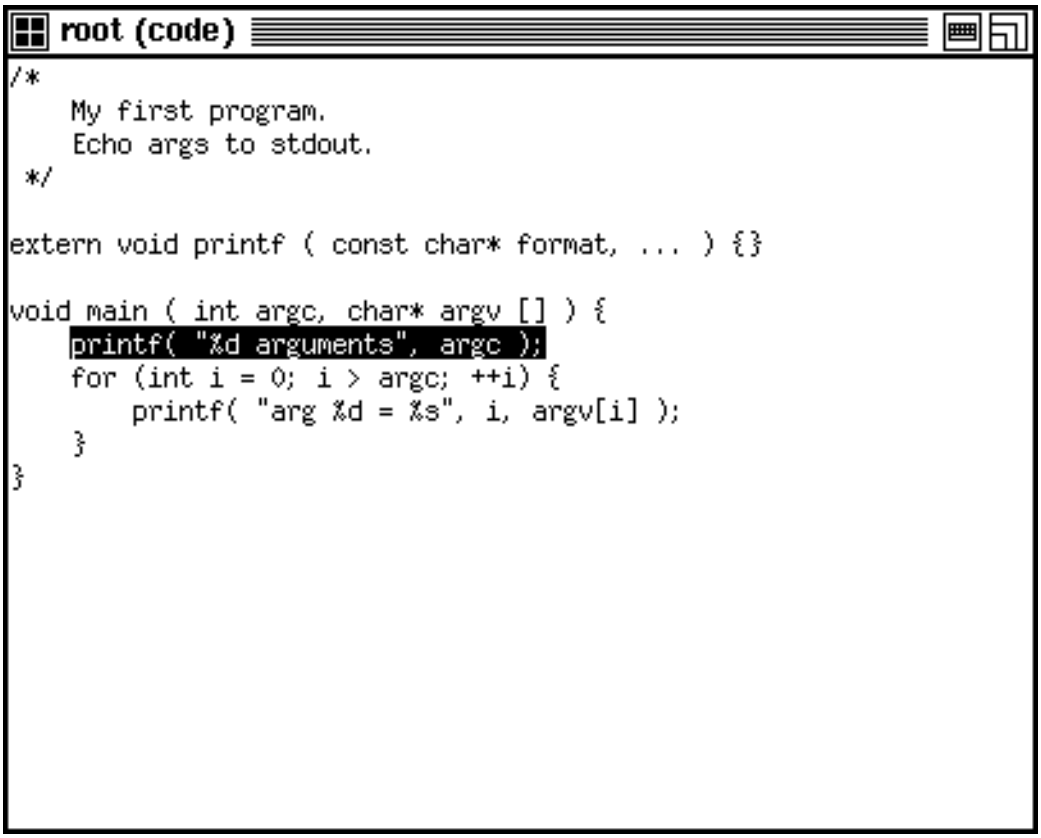
Allocation and deallocation

```
#include "StringButton.h"

#include <InterViews/button.h>

StringButton::StringButton (
  ButtonState* s,
  const char* value, const char* sample
) : (sample, 1) {
  SetString(value, false);
  subject = s;
  subject->Attach(this);
}

void StringButton::Delete () {
  subject->Detach(this);
  StringEdit::Delete();
}
```

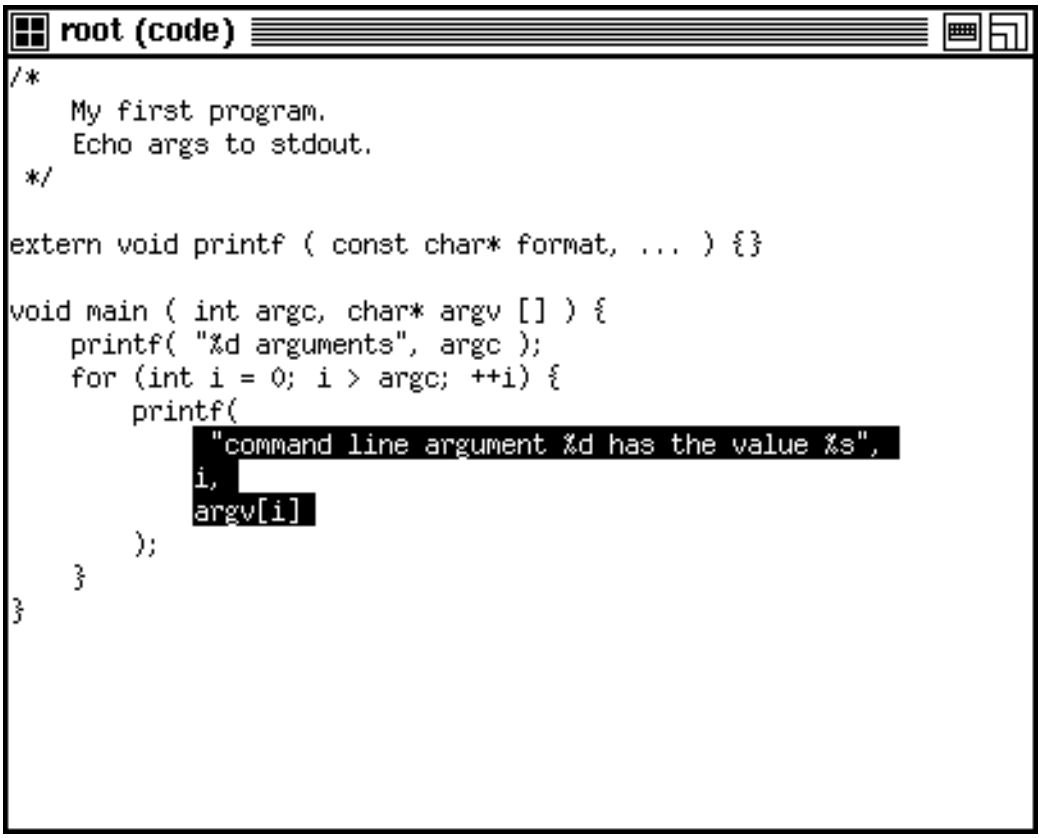
Update and Handle

```
void StringButton::Update () {
  char* text;
  subject->GetValue(text);
  SetString(text, false);
}

void StringButton::Handle (Event& e) {
  StringEdit::Handle(e);
  char* text = GetString();
  subject->SetValue(text);
}
```

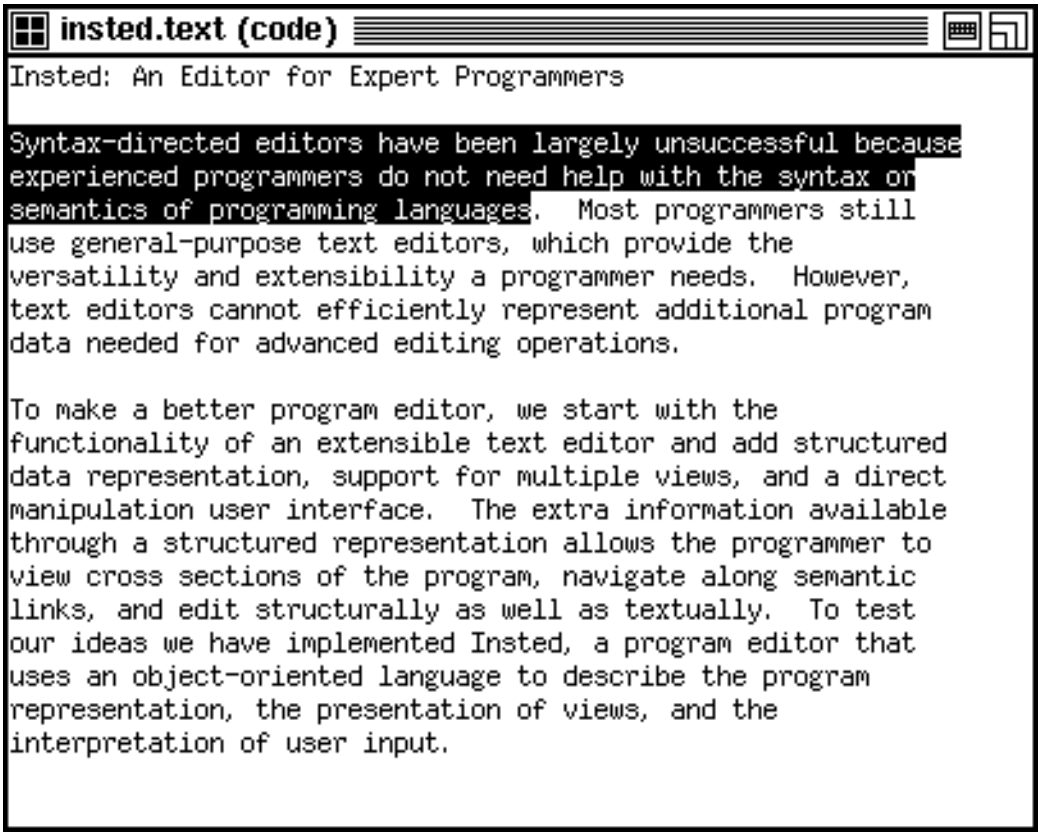
A quick look at some other applications



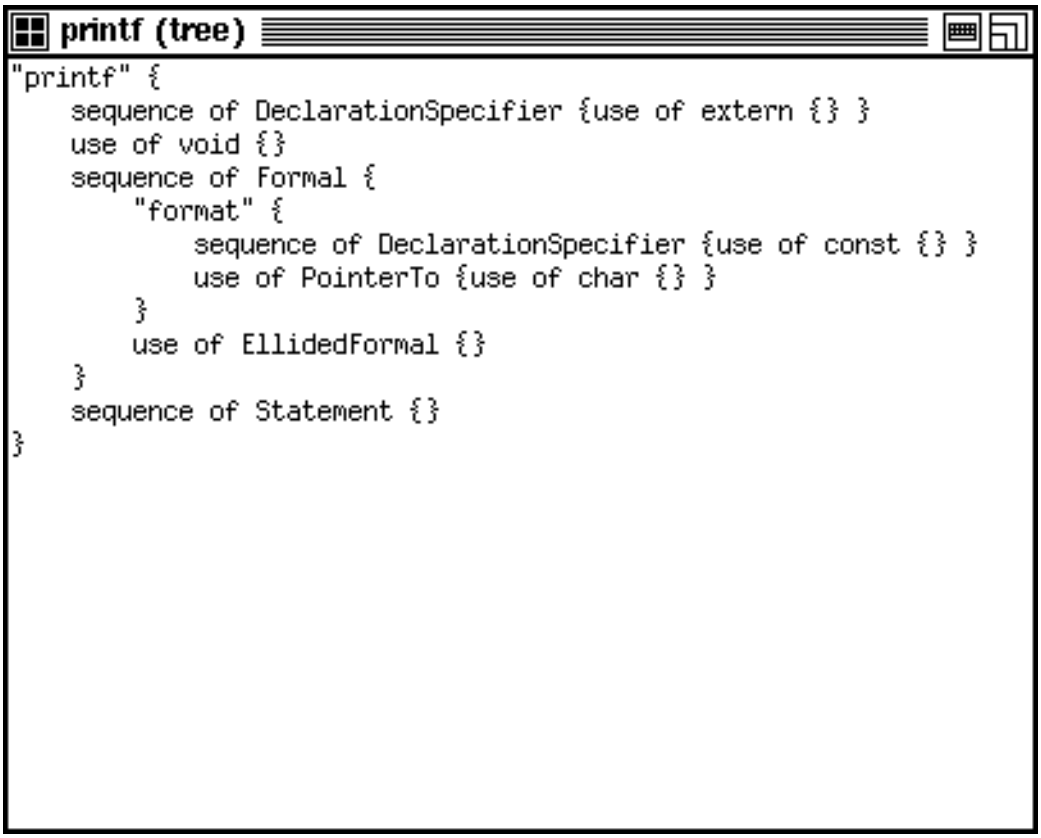
Insted — structured text editing



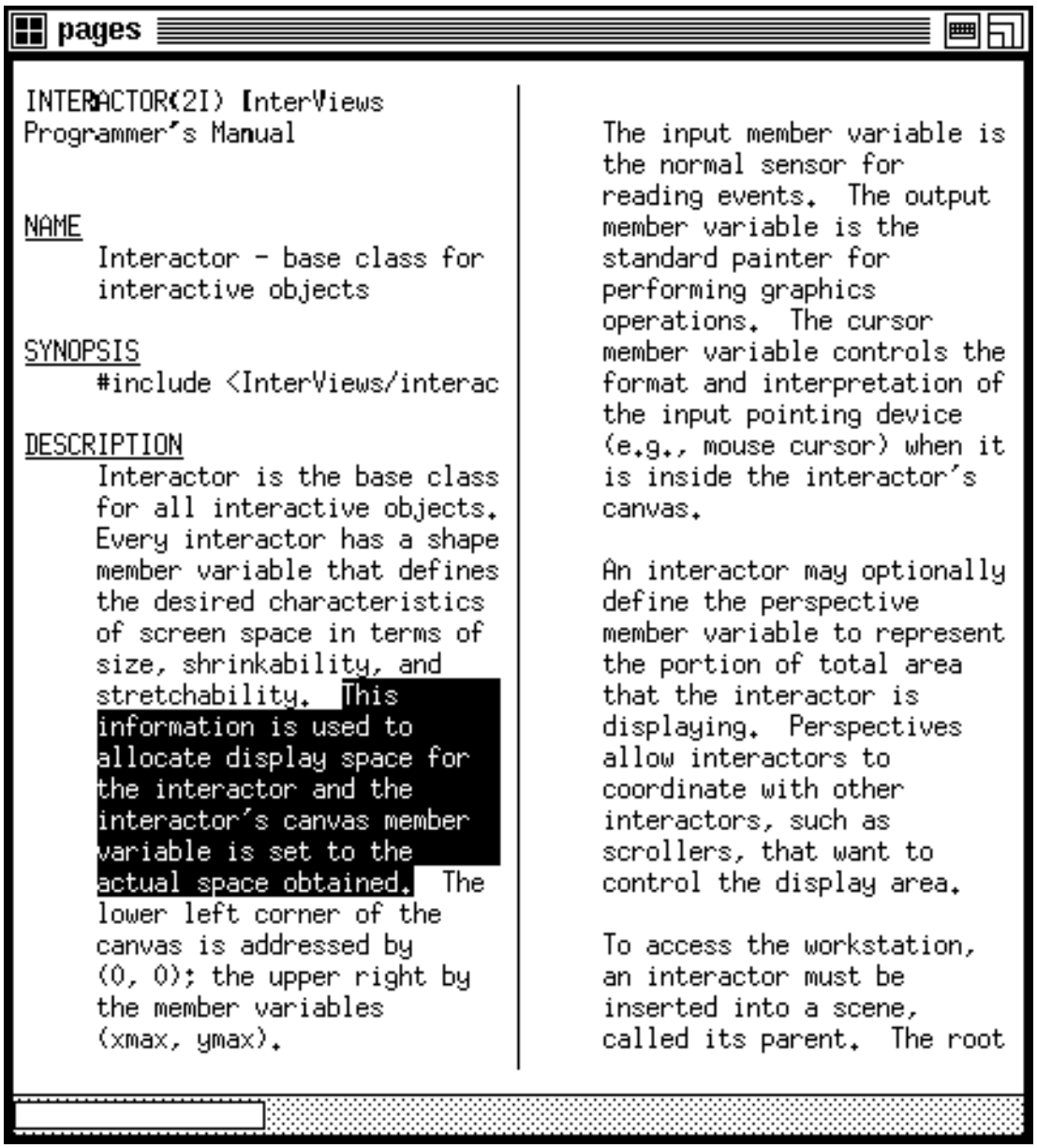
Layout specified by composition objects



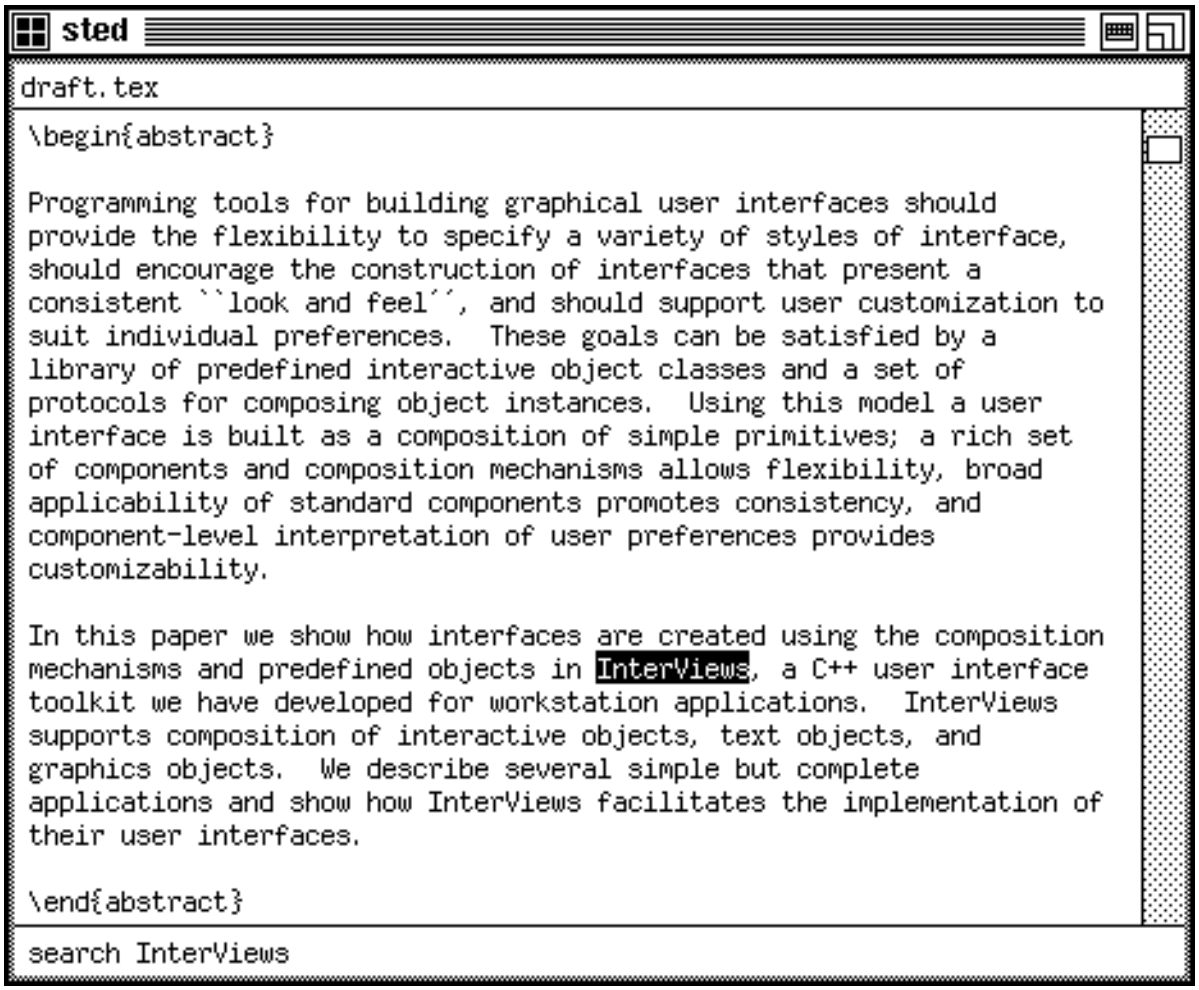
Unstructured text too



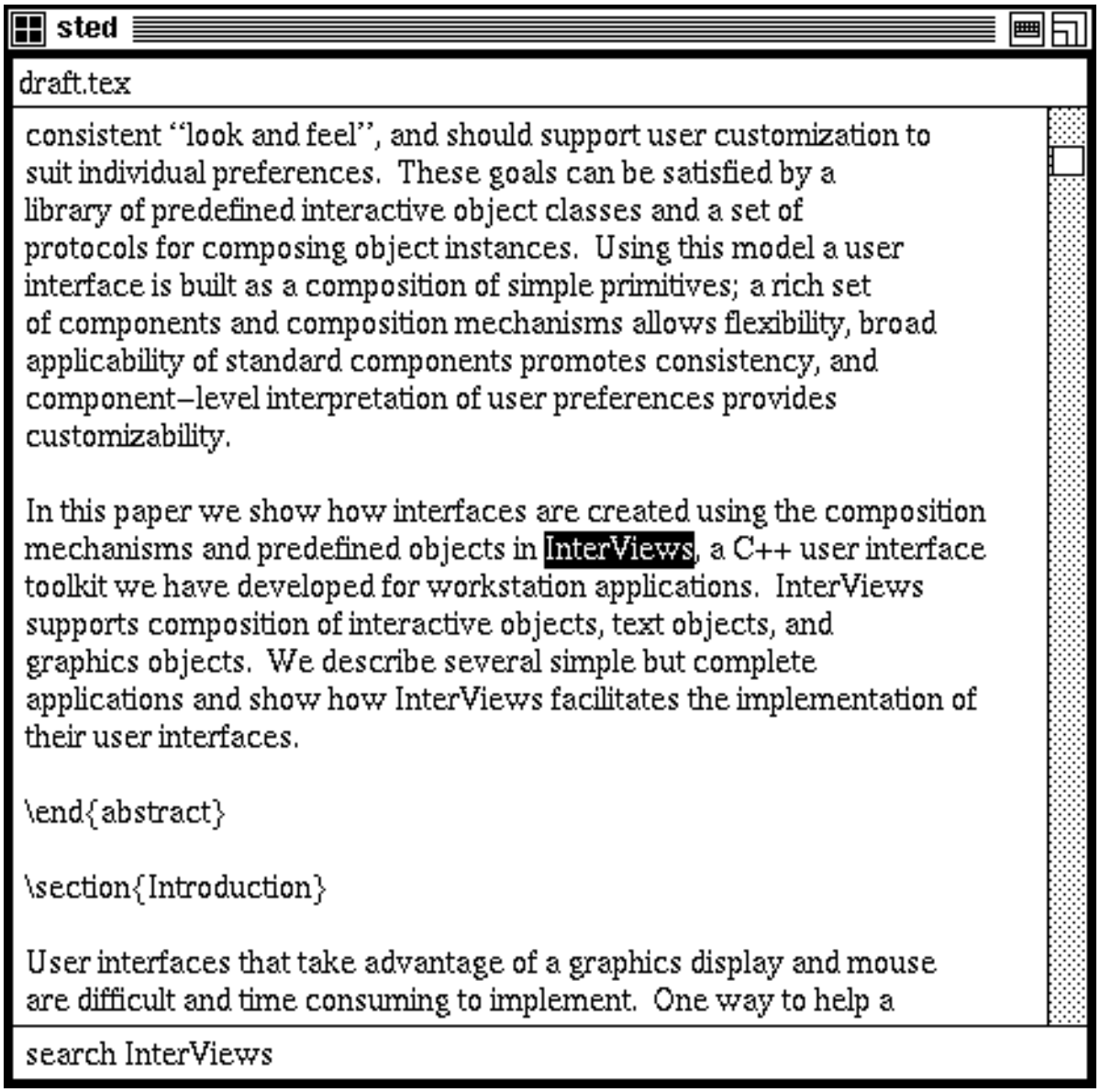
A different unparsing strategy



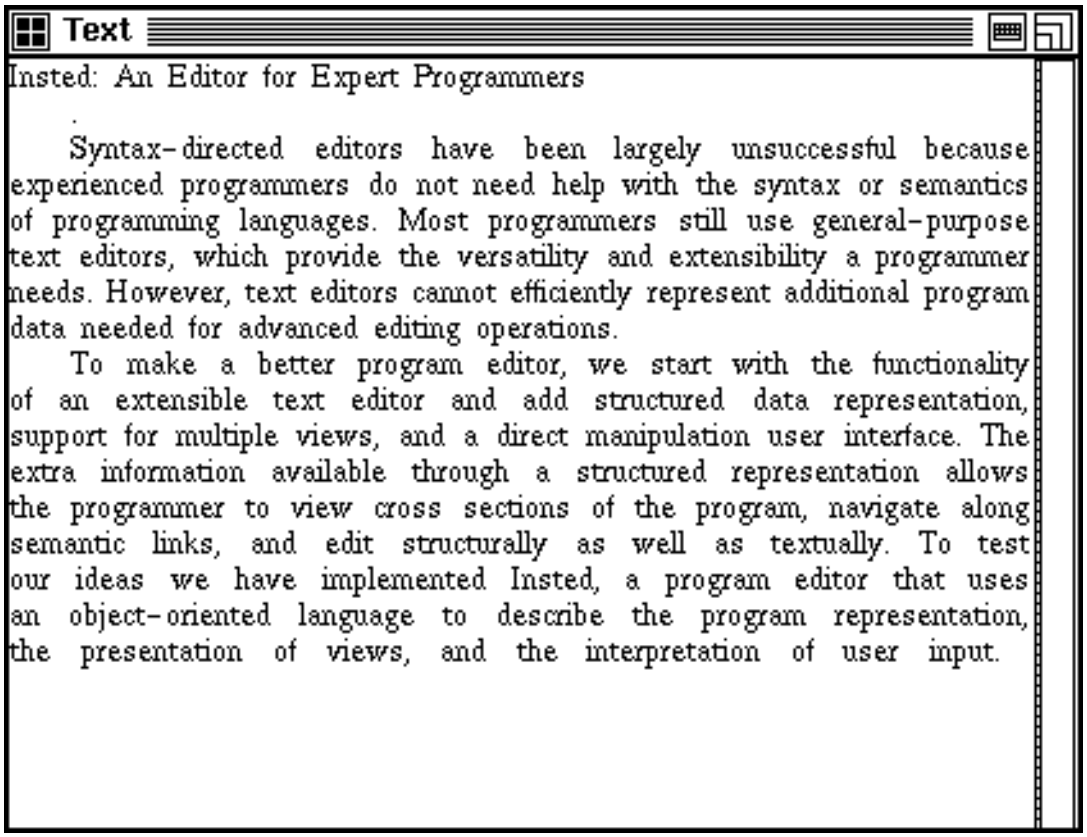
A manual page browser using structured text



A simple unstructured text editor

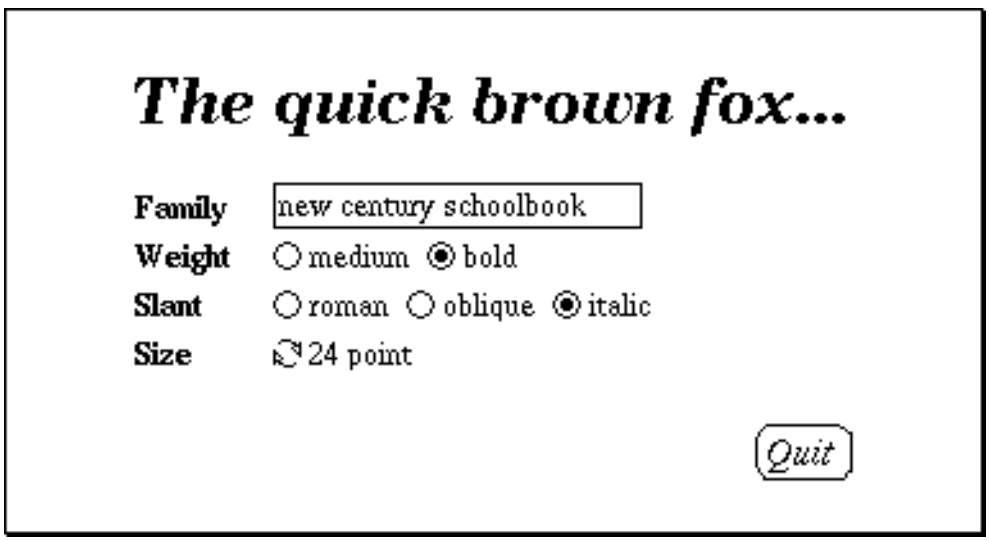


Proportional fonts are no problem



TEX style line breaking

Code listings for the Font browser



An InterViews font browser application