Programming 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 (2:00 – 2:50)

- InterViews philosophy
- InterViews and other toolkits
- Interactors, Painters, and Sensors
- Scenes and composition

A simple InterViews application (3:00 – 4:50)

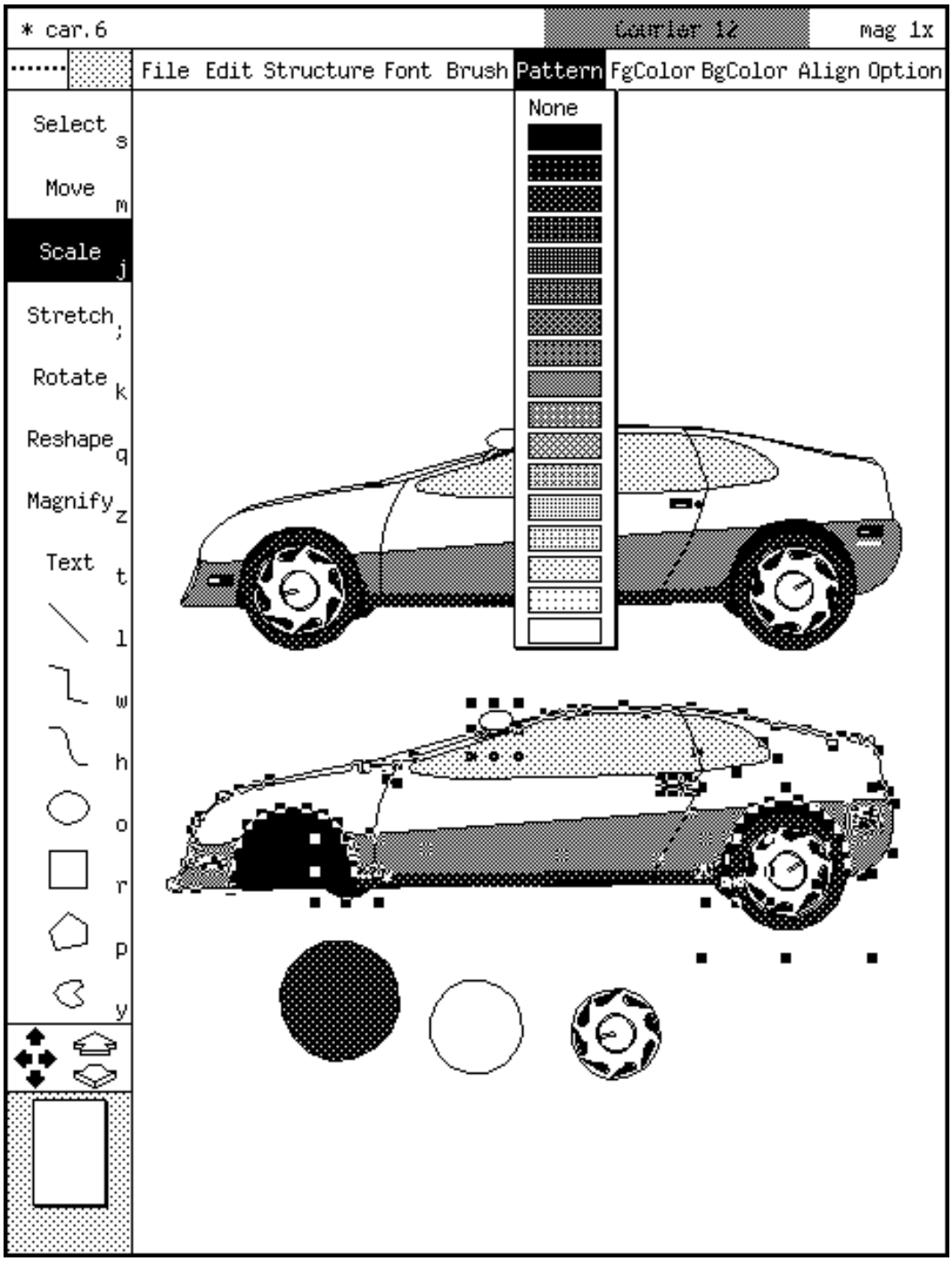
- Bare bones
- Fleshing it out
- User customization

A font browser application (4:00 – 5:50)

- Deriving new classes
- Final details

An overview of InterViews

(2:00 – 2:50)



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

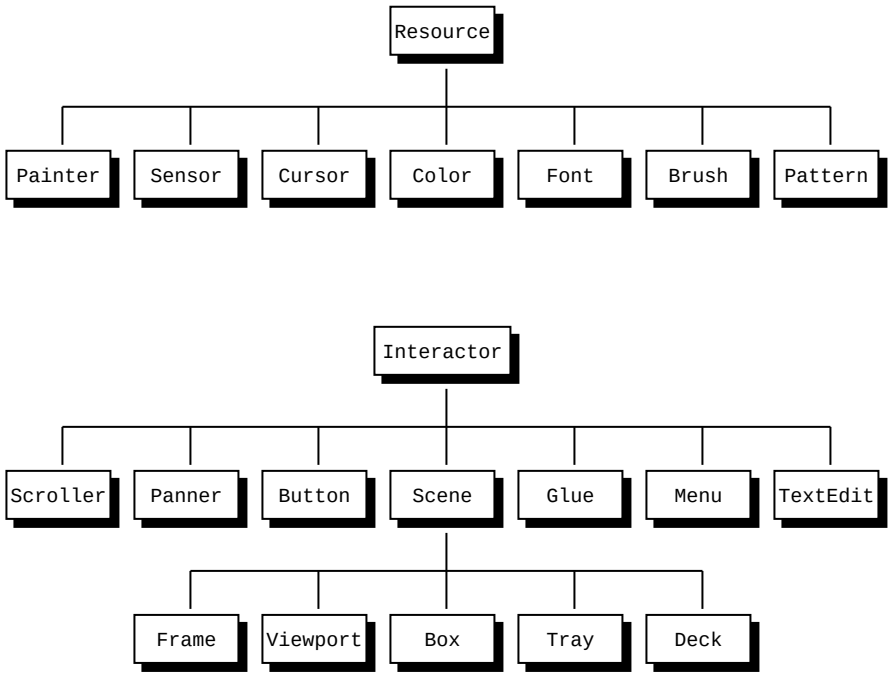
- InterViews is more than a toolkit

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



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

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

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

Painters

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

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

Scenes

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

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

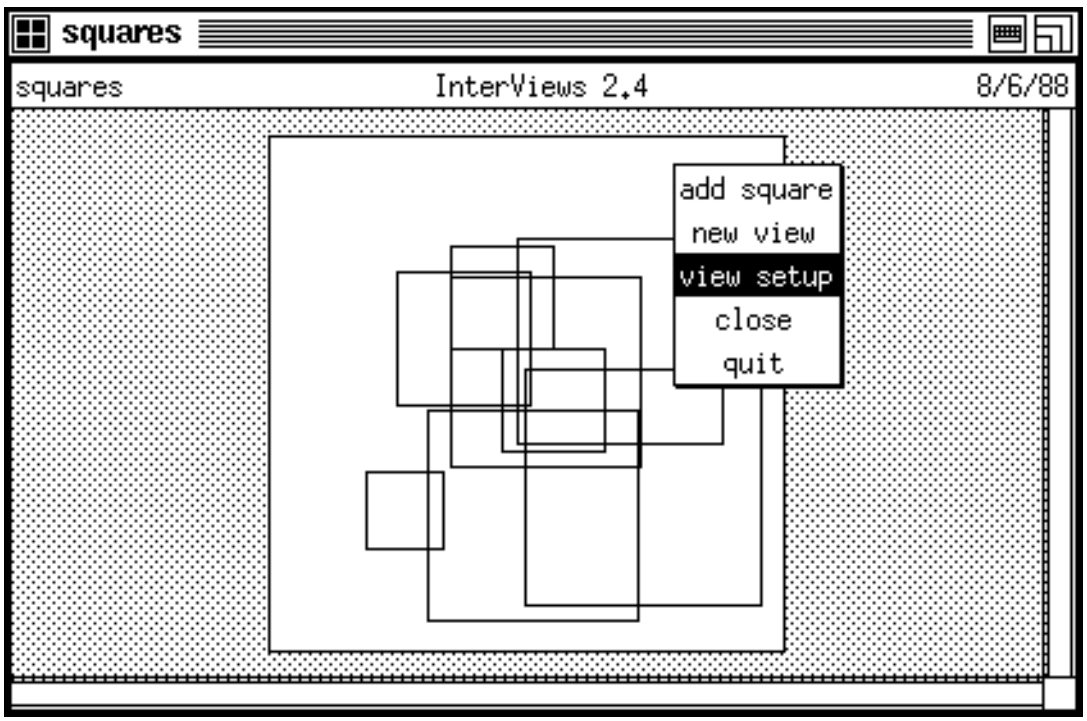
StringEdit - one line text editing

TextViewer - multi-line text editing

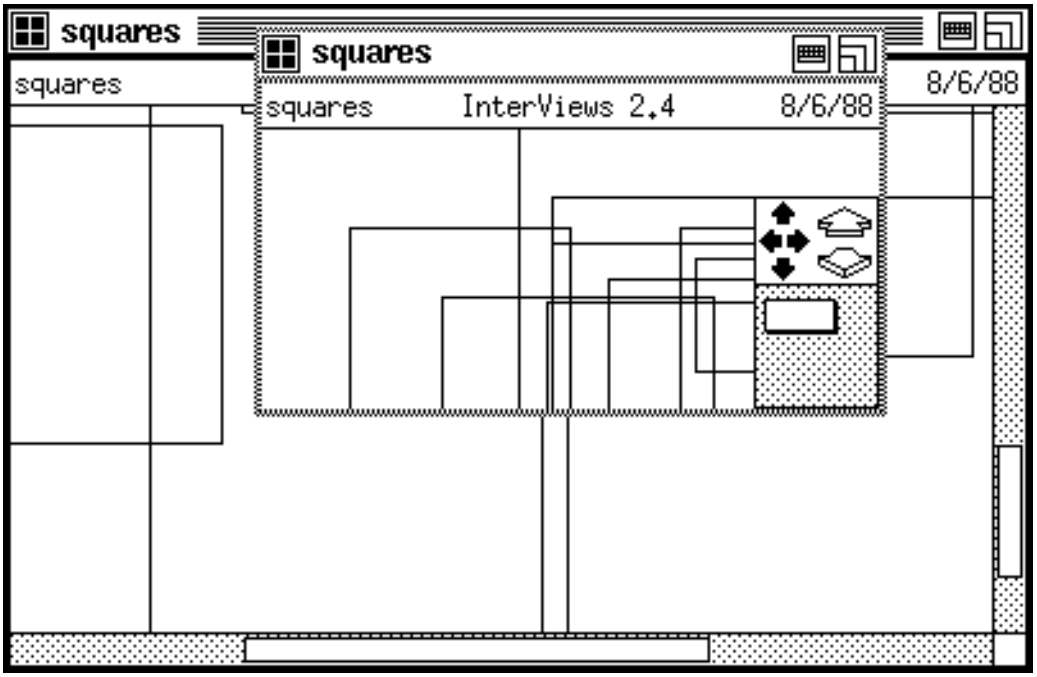
WorldView - window managers

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

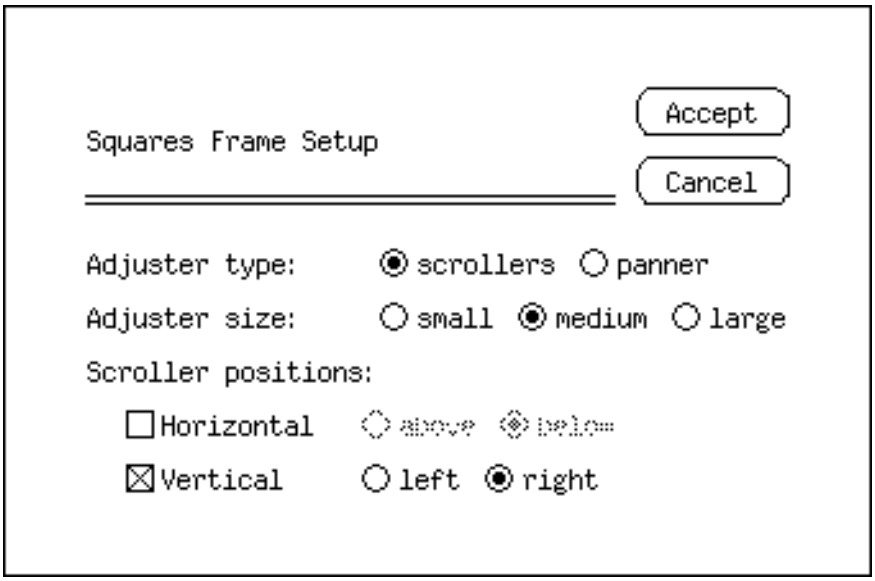
Painters, graphic library
idraw

Text

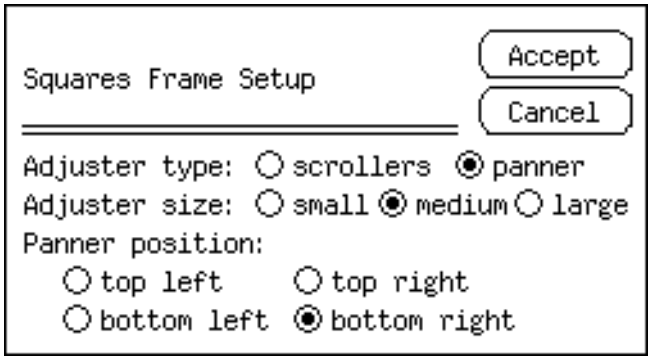
Composable primitives
Layout strategies

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:

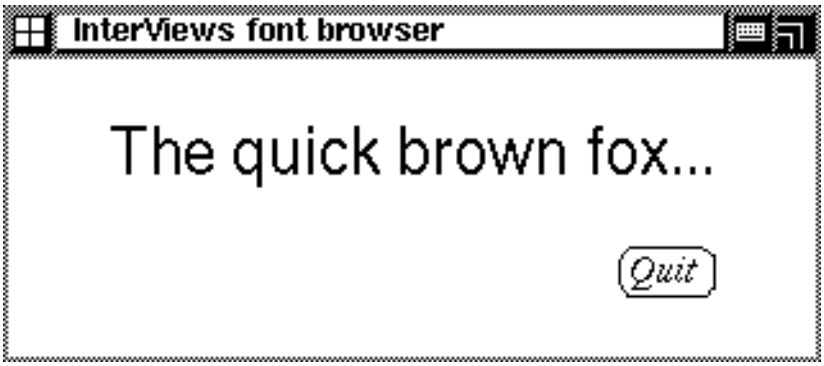
demos — squares, graphics, pages,
accessories — alert, dclock, istat, iwm, mailbox
applications — idraw,
research — ispy, inclink, insted, unidraw, ...

Publicly available

X11 tape
ftp from interviews.stanford.edu (36.22.0.175)
Email to `interviews@interviews.stanford.edu`

A simple InterViews application

(3:00 – 3:50)



An InterViews font display 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>"

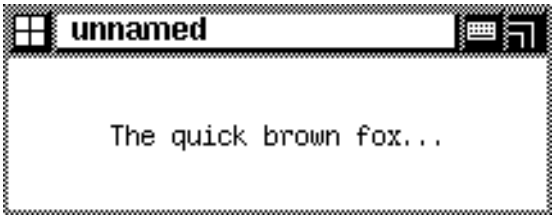
CC -c application.c

CC application.o -lInterViews -lX -lm

InterViews Imakefiles

```
CDEBUGFLAGS     = -g
SRC             = .
SRCS = $(SRC)//**/*.c
SYS_LIBRARIES = -lInterViews$X -l$X -lm
OBJS            = \
    fontsample.o \
    main.o

ComplexProgramTarget(t4)

MakeObjectFromSrc(fontsample)
MakeObjectFromSrc(main)
```

Just a Message



A ShadowFrame, Boxes, and Glue

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 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

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;
}
```

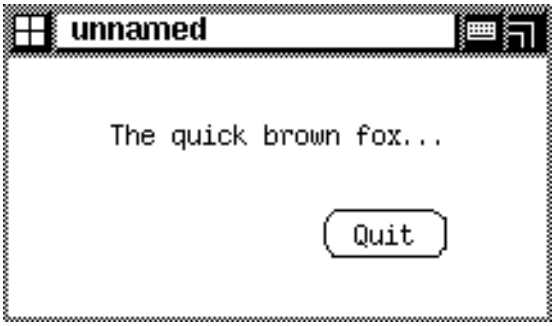
Simple composition

Box (HBox, VBox)

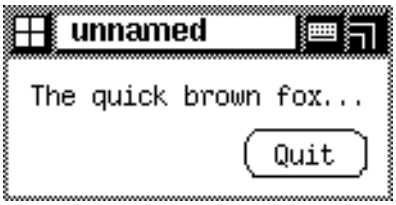
- Tiled composition
- Simplified TEX model
- Left-to-right and top-to-bottom

Glue (HGlue, VGlue)

- Natural, stretch, and shrink
- Sizes in real world units

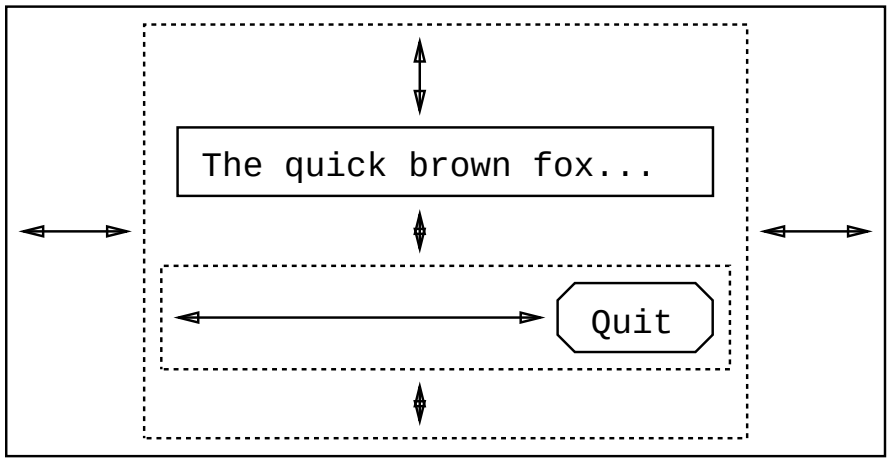


A button to push in a Dialog



See the glue shrink...

... and stretch



Arrangement of Boxes and Glue

Adding the button

```
#include <InterViews/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;
  delete quit;
  delete world;
  return 0;
}
```

Simple user interaction

ButtonState

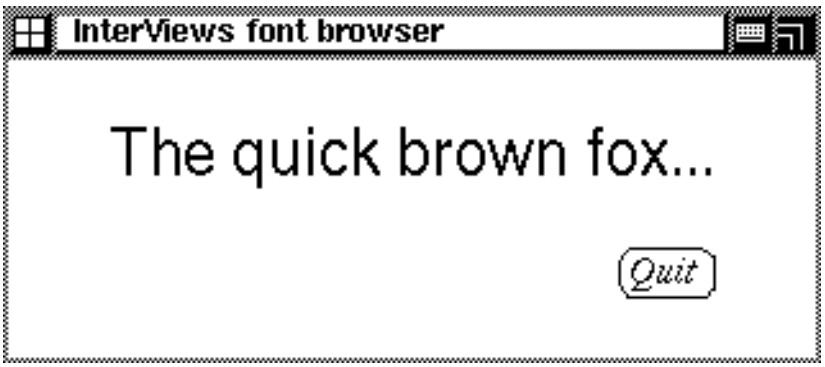
A subject (a piece of state)
A list of views (buttons)
Views notified when state changes

Button

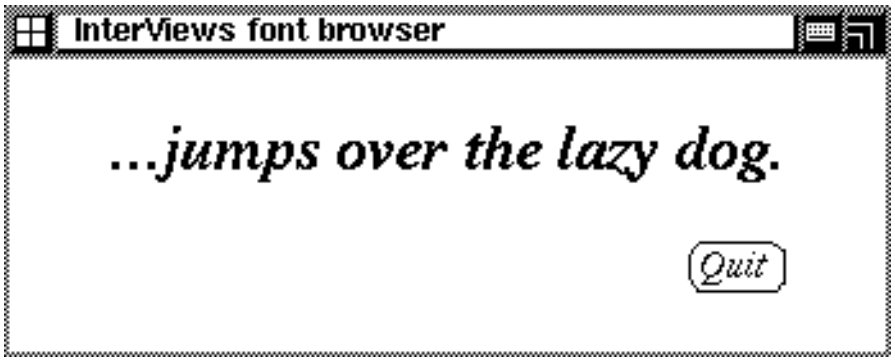
User pushes the button to change the state
Other types of button available

Dialog

Event loop inside
Exits when button state not 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", "ifd*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

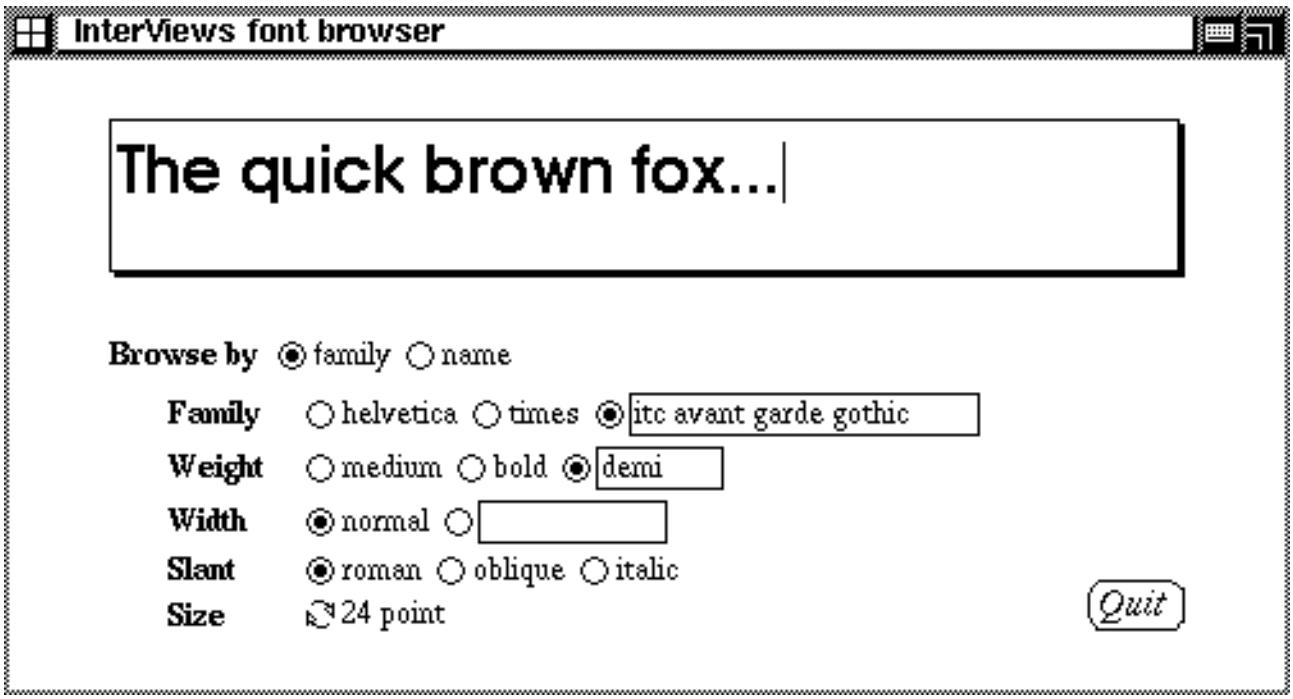
- Use properties to allow user customization
- Pass arguments to World
  - Default values in application code
  - Standard ".Xdefault" mechanism
  - Command line arguments
- PropertyData defines defaults
  - Hierarchical structure
  - Name/value specification
  - InterViews handles "standard" propeties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

Summary

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

A font browser application

(4:00 – 4:50)



An InterViews font browser application

The code as it stands

```
#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);
private:
  void Init(const char*);
private:
  Font* font;
  char* fontname;
  char* sample;
};
```

Use it to show the sample

```
#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

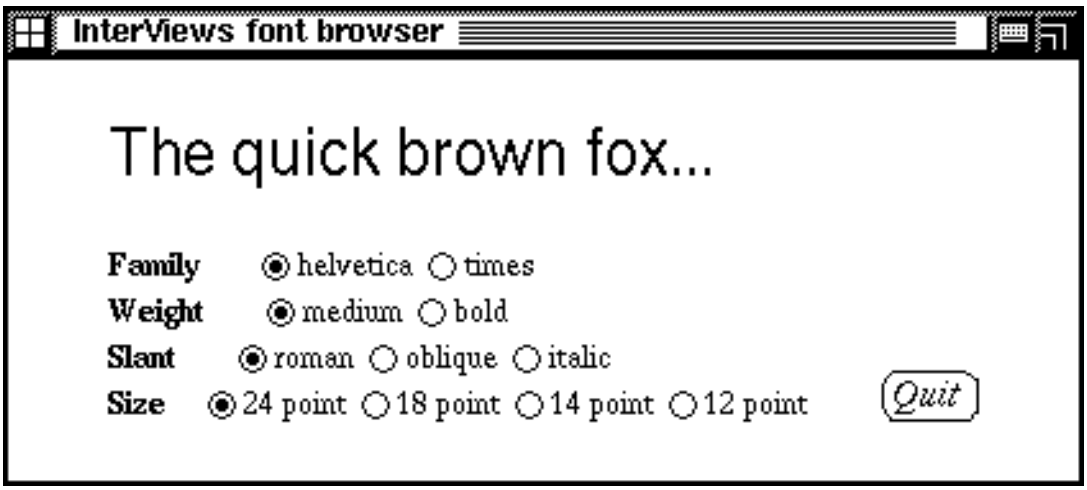
```
FontSample::FontSample(const char* sample) : () {
  Init(sample);
}

FontSample::~FontSample () {
  delete sample;
  delete fontname;
  delete 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);
  delete output;
  output = tmp;
  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);
    delete font;
    font = new Font(fontname);
    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 speical dialog

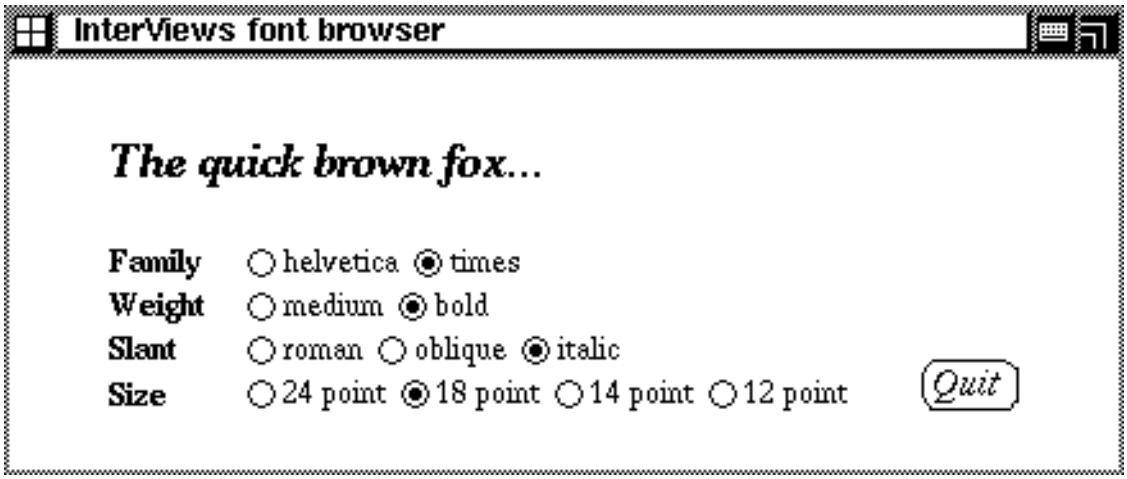
```
class FontBrowser : public Dialog {
public:
  FontBrowser(ButtonState*, const char* sample);
  virtual ~FontBrowser();

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

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

Attributes for 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 font name

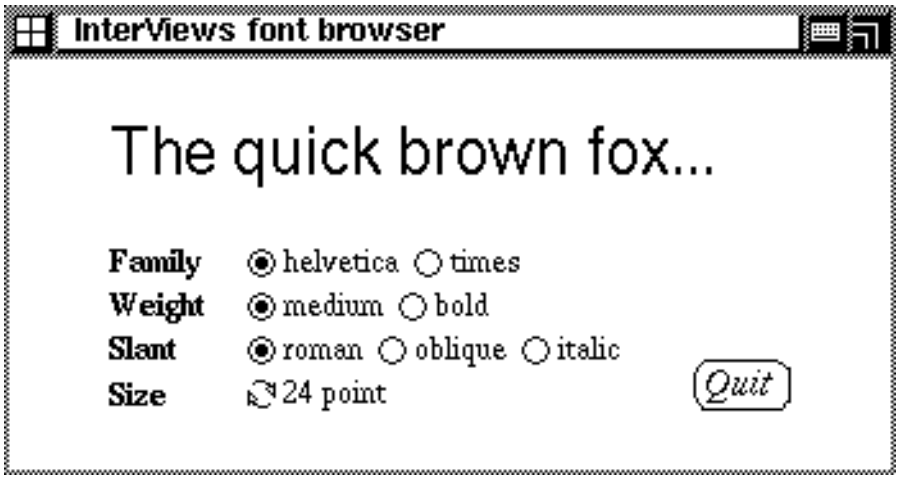
```
boolean FontBrowser::Accept () {
    Event e;
    int v;
    do {
        Read(e);
        e.target->Handle(e);
        UpdateFontname();
        state->GetValue(v);
    } while (v == 0 && e.target != nil);
    return v == 1 || e.target == nil;
}

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());
}
```



A Tray lines things up.

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, ...);
```



A new type of button.

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();
private:
  void Init(CycleButtonChoice*);

  CycleButtonChoice* choices;
  int count;
  int current;
};
```

Constructors and destructors

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

Configuration

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

MarginFrame

```
class MarginFrame : public MonoScene {
public:
  MarginFrame(Interactor*, int hmargin, int vmargin);
  MarginFrame(Interactor*,
    int hmargin, int hshrink, int hstretch,
    int vmargin, int vshrink, int vstretch
  );
  virtual ~MarginFrame();
protected:
  void Reconfig();
  void Resize();
private:
  void Init(Interactor*, int, int, int, int, int, int);

  int hmargin, hshrink, hstretch;
  int vmargin, vshrink, vstretch;
};
```

Construction and destruction

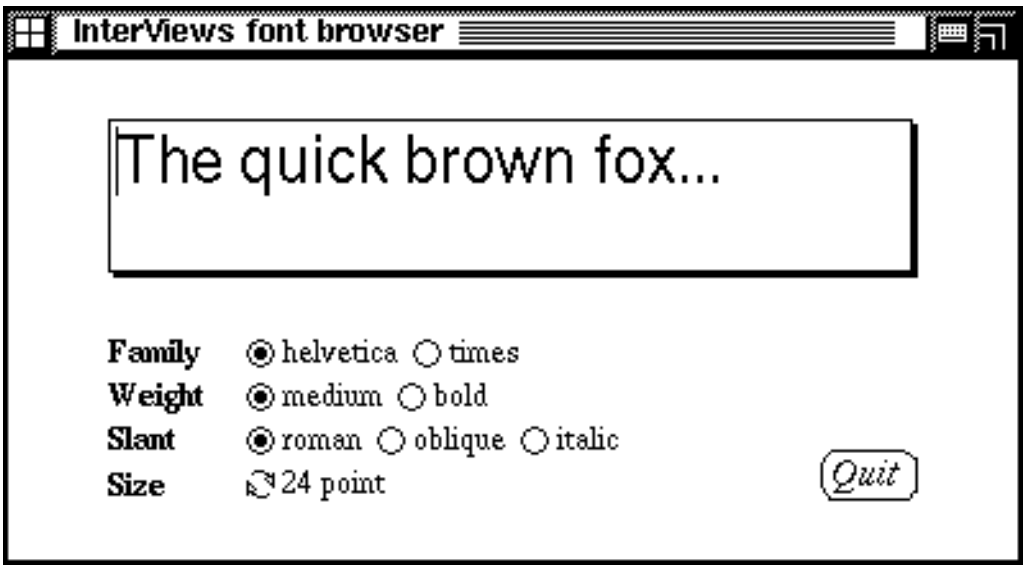
```
MarginFrame::MarginFrame (
  Interactor* i, int hmargin, int vmargin
) : () {
  Init(i, hmargin, 0, 0, vmargin, 0, 0);
}

MarginFrame::~MarginFrame () { }

void MarginFrame::Init (
  Interactor* i, int h, int hshr, int hstr, int v, int
vshr, int vstr
) {
  hmargin = h * 2;
  hshrink = hshr * 2;
  hstretch = hstr * 2;
  vmargin = v * 2;
  vshrink = vshr * 2;
  vstretch = vstr * 2;
  Insert(i);
}
```

Configuration

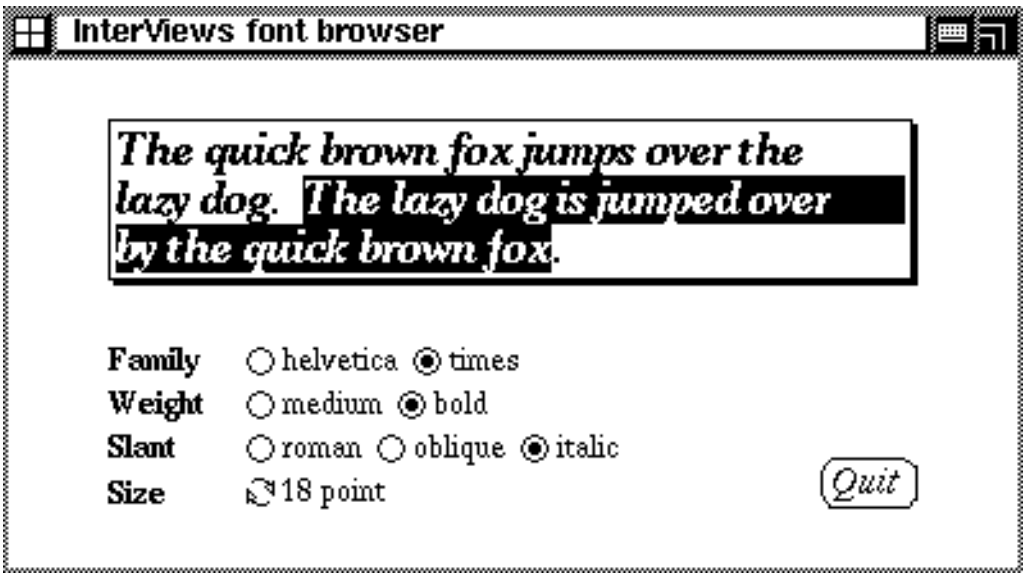
```
void MarginFrame::Reconfig () {
  MonoScene::Reconfig();
  shape->width += hmargin;
  shape->height += vmargin;
  shape->hshrink += hshrink;
  shape->hstretch += hstretch;
  shape->vshrink += vshrink;
  shape->vstretch += vstretch;
}
```

Resize

```
void MarginFrame::Resize () {
  canvas->SetBackground(output->GetBgColor());

  Coord hextra = (xmax+1) - shape->width;
  Coord h = hmargin;
  if (hextra > 0 && shape->hstretch != 0) {
    h += int(
      float(hstretch) / float(shape->hstretch)
      * float(hextra)
    );
  } else if (hextra < 0 && shape->hshrink != 0) {
    h += int(
      float(hshrink) / float(shape->hshrink)
      * float(hextra)
    );
  }
  ...

  Place(component, h/2, v/2, xmax-h/2, ymax-v/2);
}
```



An editable text sample...



...for larger samples.

Changes to FontSample

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

Construction and destruction

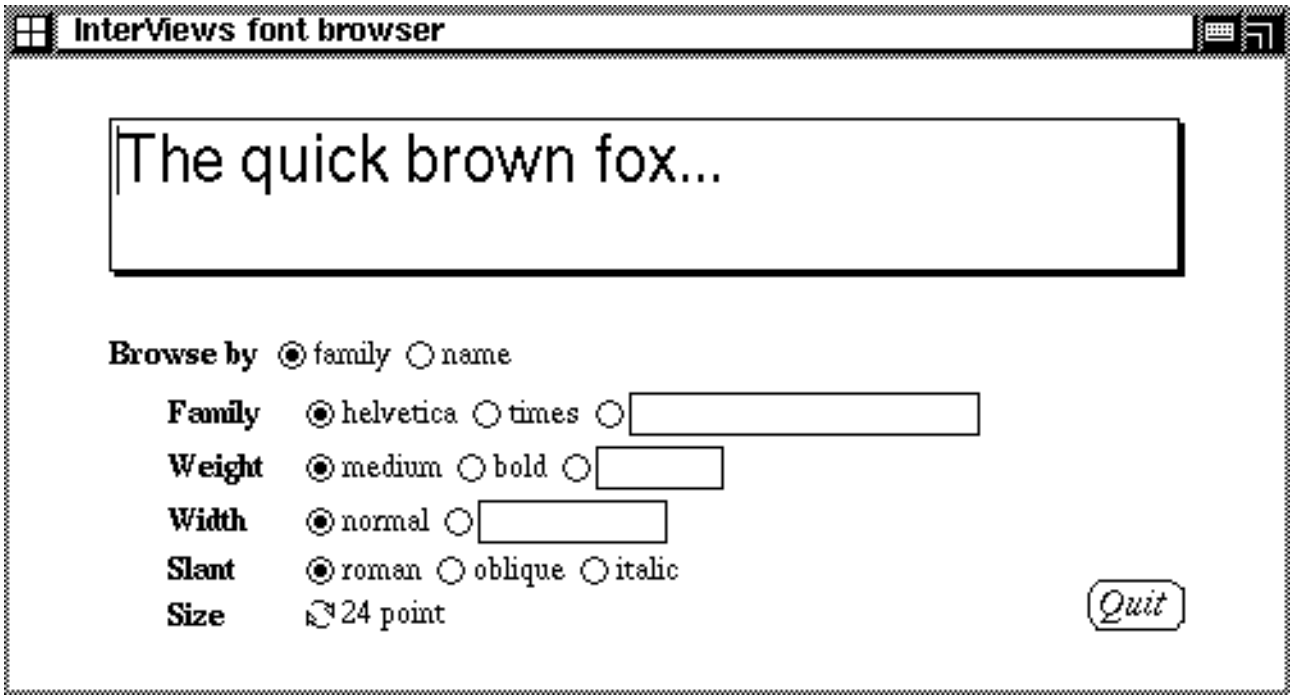
```
FontSample::FontSample(
  const char* sample
) : (2, strlen(sample), 8, Reversed) {
  Init(sample);
}

FontSample::~FontSample () {
  delete text;
  delete sample;
  delete fontname;
  delete 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

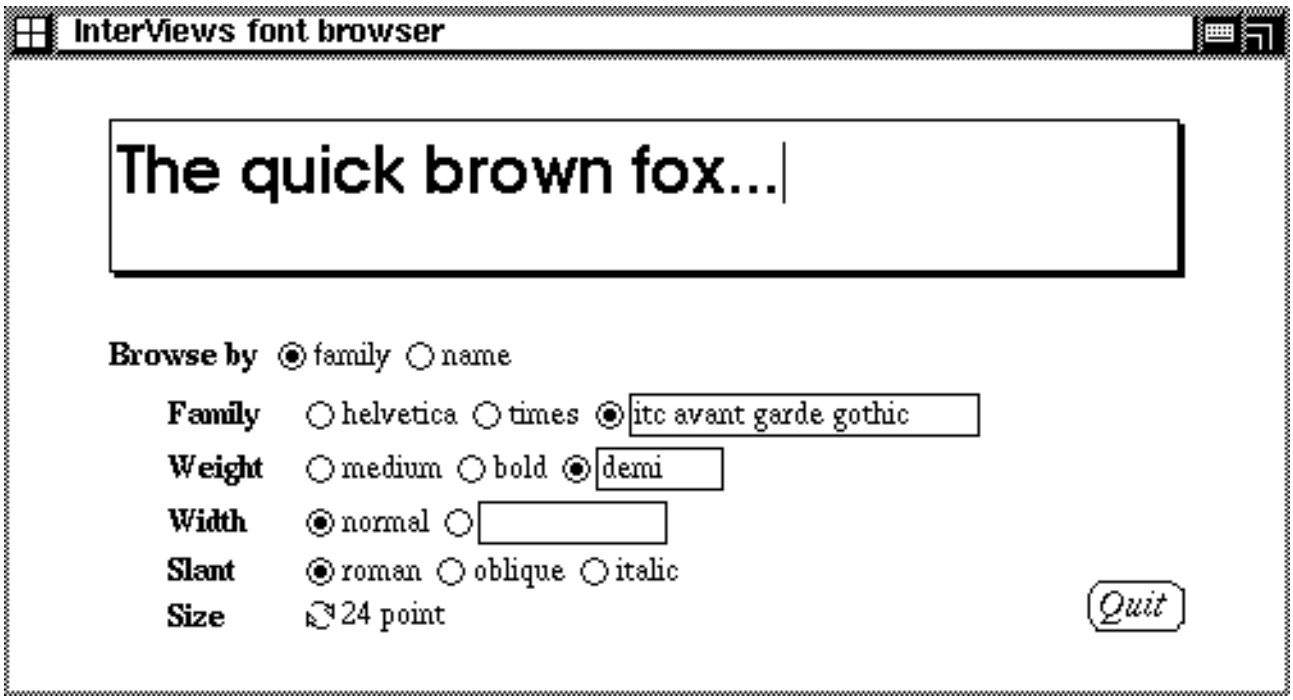
```
void FontSample::Reconfig () {
  ...
  TextEditor::Reconfig();
  shape->vshrink = shape->height - shape->vunits;
}

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;
  }
}
```



Editable buttons...



...for unusual 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:
  void Init();
private:
  char* text;
  TextDisplay* display;
  int offset;
};
```

Construction and destruction

```
StringButton::StringButton (
  ButtonState* s, const char* sample
) : (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

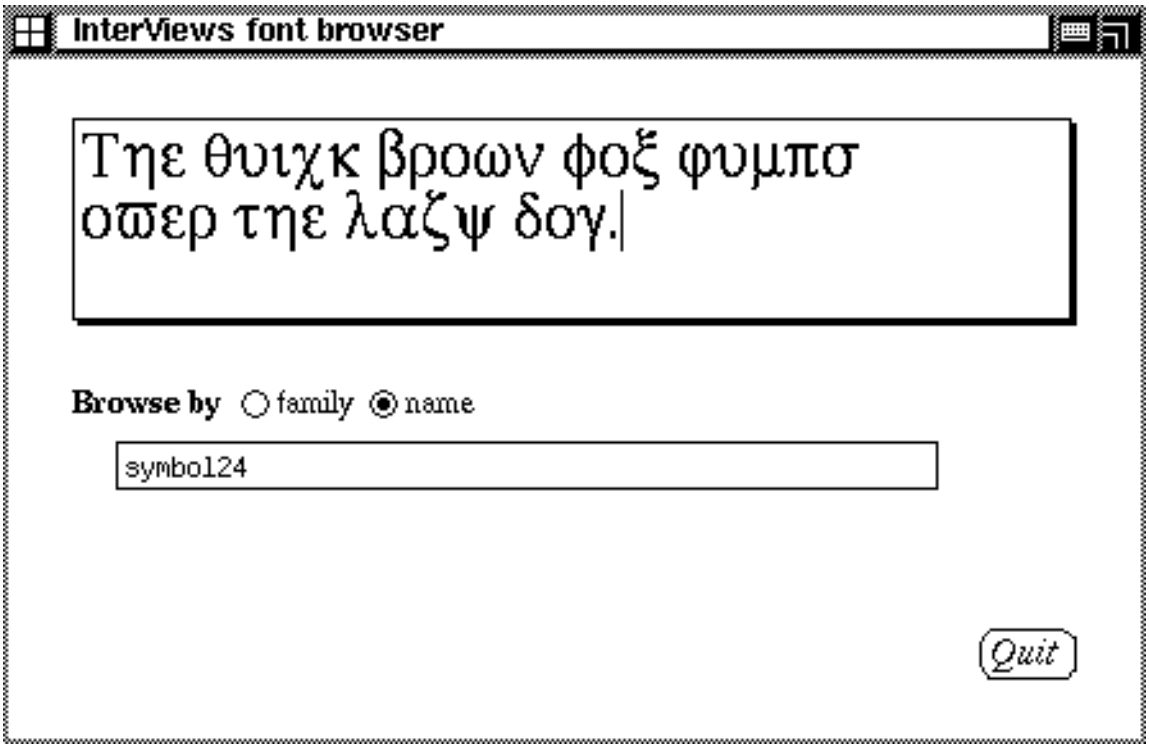
```
void StringButton::Reconfig () {
  RadioButton::Reconfig();
  shape->width += 4;
  shape->height += 4;
}

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';
  }
}
...
```



Browsing by font name, too.

Two browsing modes

```
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,
    ...
  );
}
```

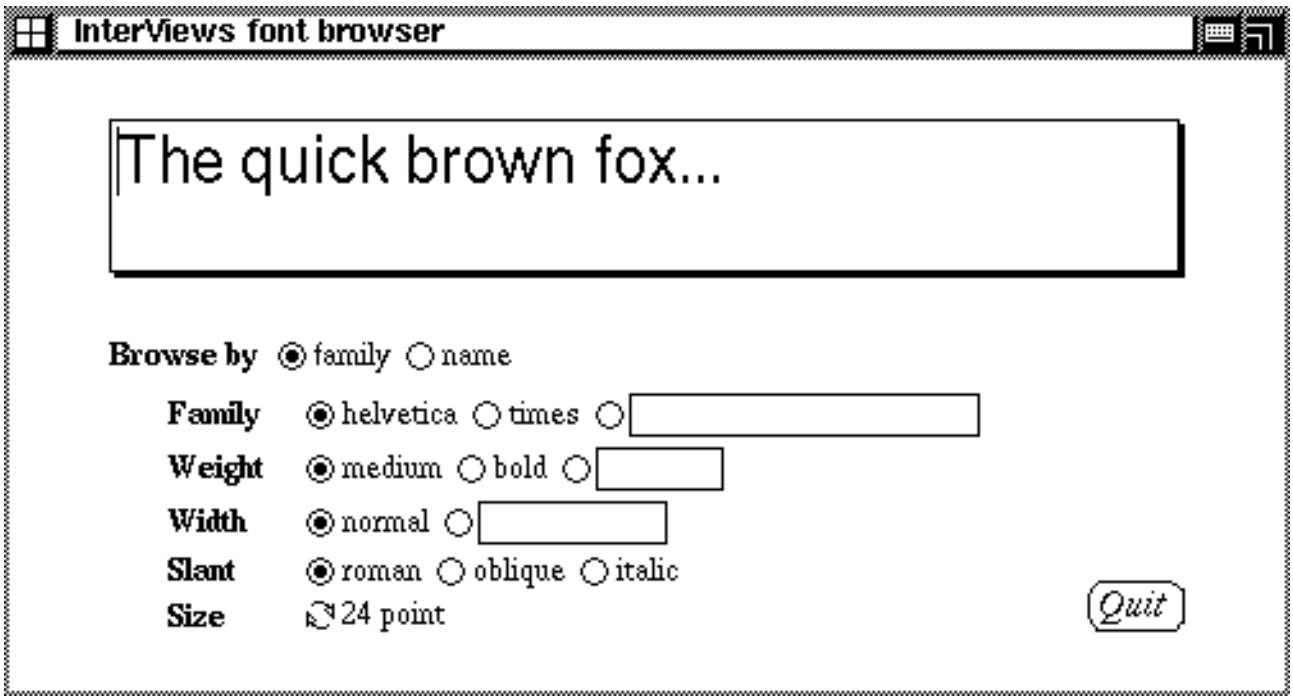
Flipping the Deck

```
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 font name

```
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);
  }
  ...
}
```

Code listings for the font browser



An InterViews font browser application