Extreme GUI Makeover: Lookin' Better

Scott Violet
Shannon Hickey
Romain Guy

Sun Microsystems, Inc.
http://java.sun.com

TS-1548

Copyright © 2006, Sun Microsystems, Inc., All rights reserved.

Goal

Learn to take advantage of advanced Swing and Java 2D™ API to create visually stunning applications

Goal

Learn to take advantage of advanced Swing and Java 2D™ API to create visually stunning applications

And have fun!

Agenda

Introduction

Last year's application, this year's application, inspiration!

Spicing up the Main Window

Shadows, fancy rendering, drag and drop feedback

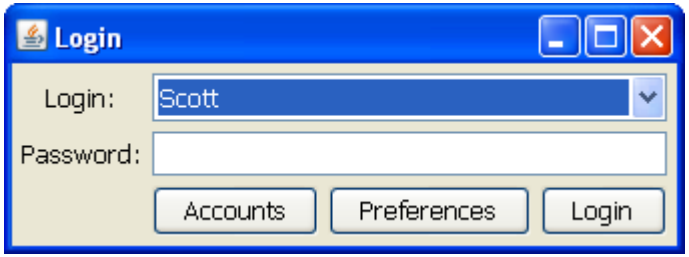
Tables, Lists, and of Course Text

Beautifying table, web style lists, and message folding

Search Dialogs and Sending Messages

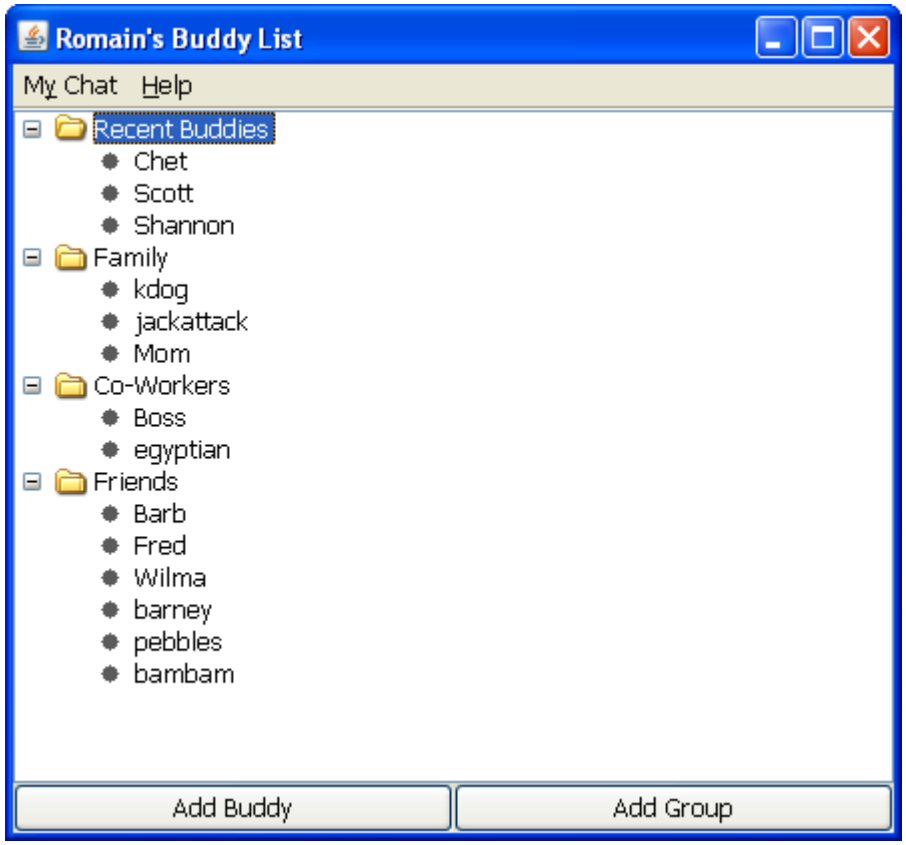
Color mixing, frosting, and animation

Chat: Plain Login



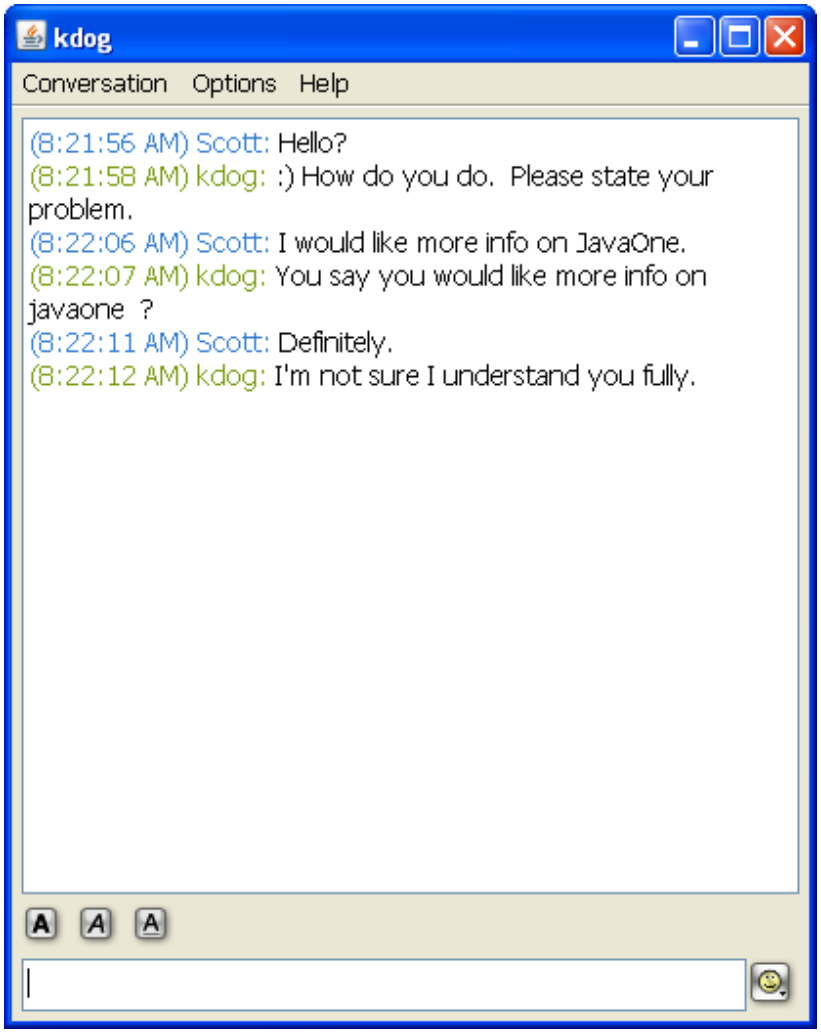
Chat: Extreme Login

Chat: Plain Buddy List

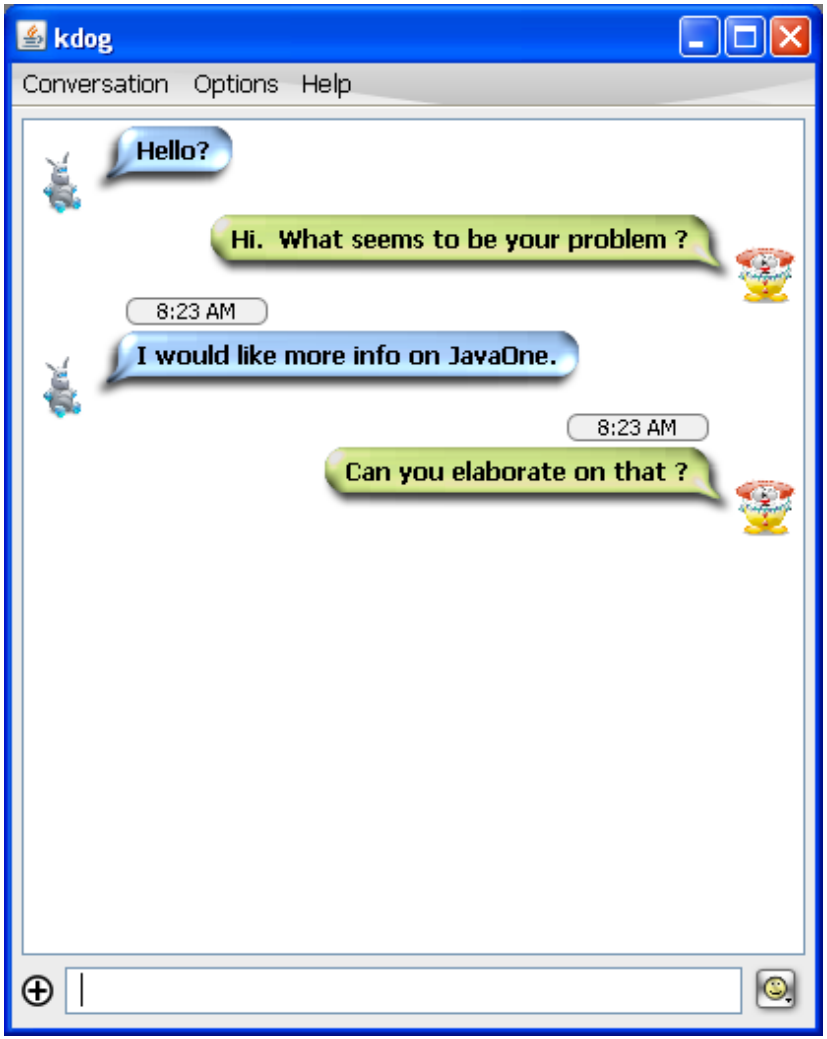


Chat: Extreme Buddy Lists

Chat: Plain Chat Window

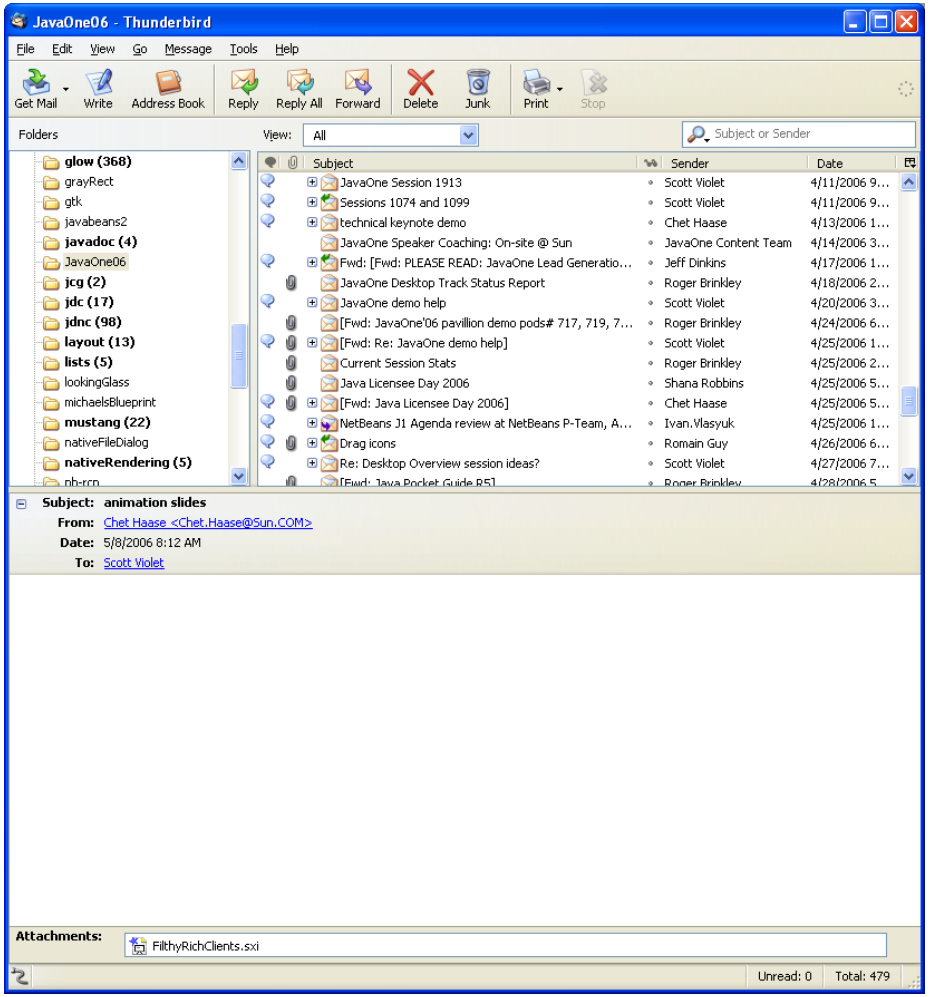


Chat: Extreme Chat Window

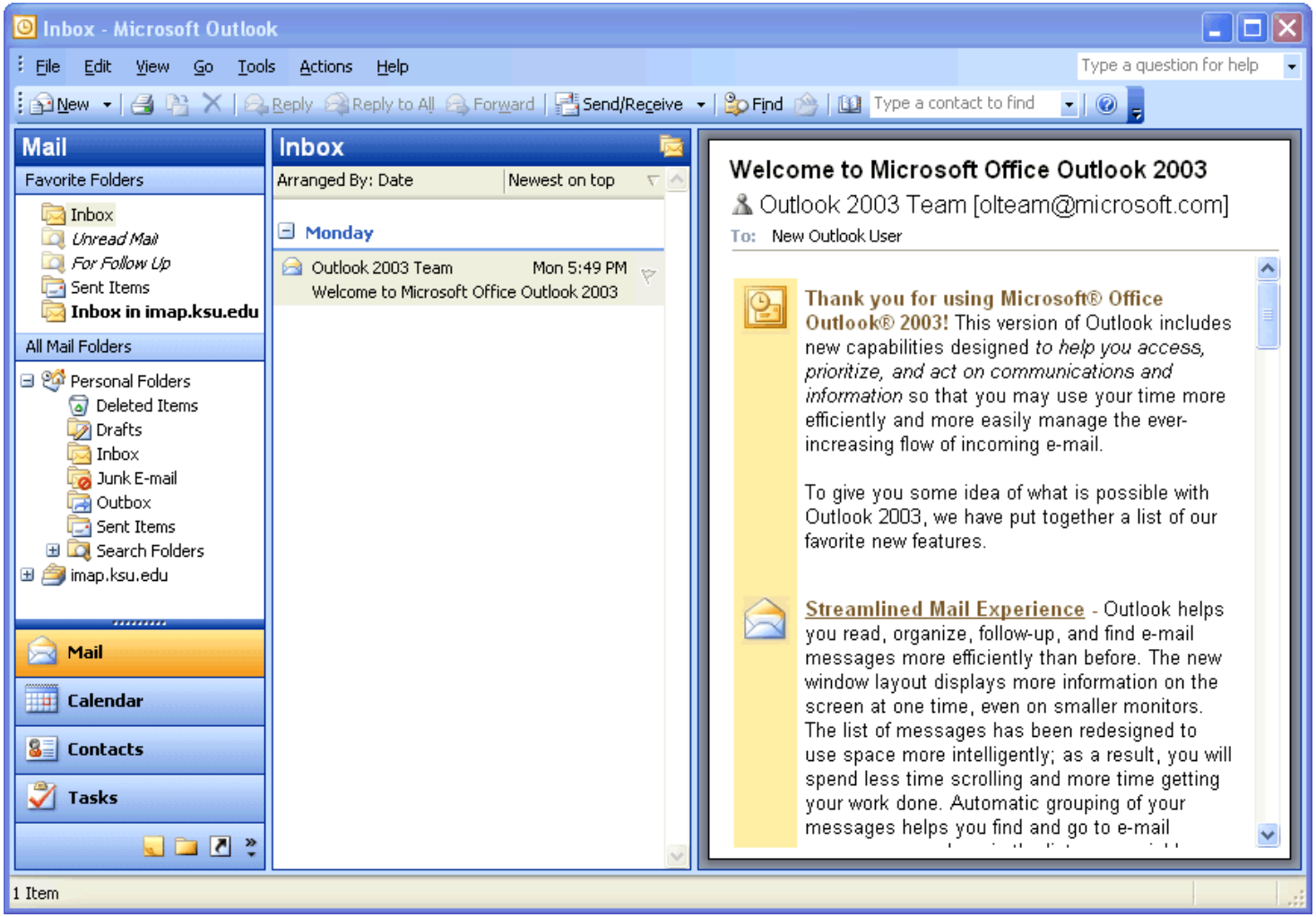


This Year's Makeover Application

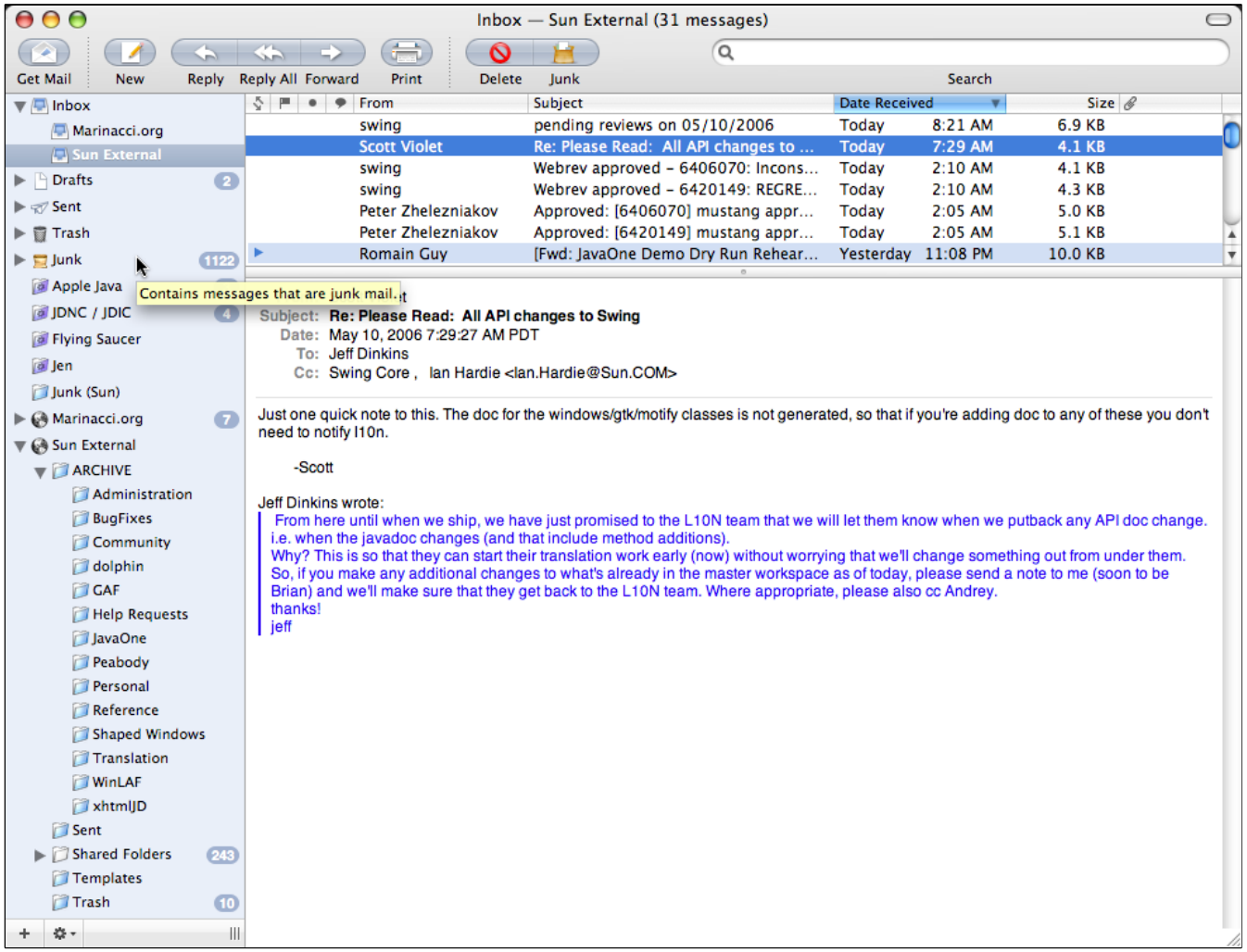
This Year's Makeover Application



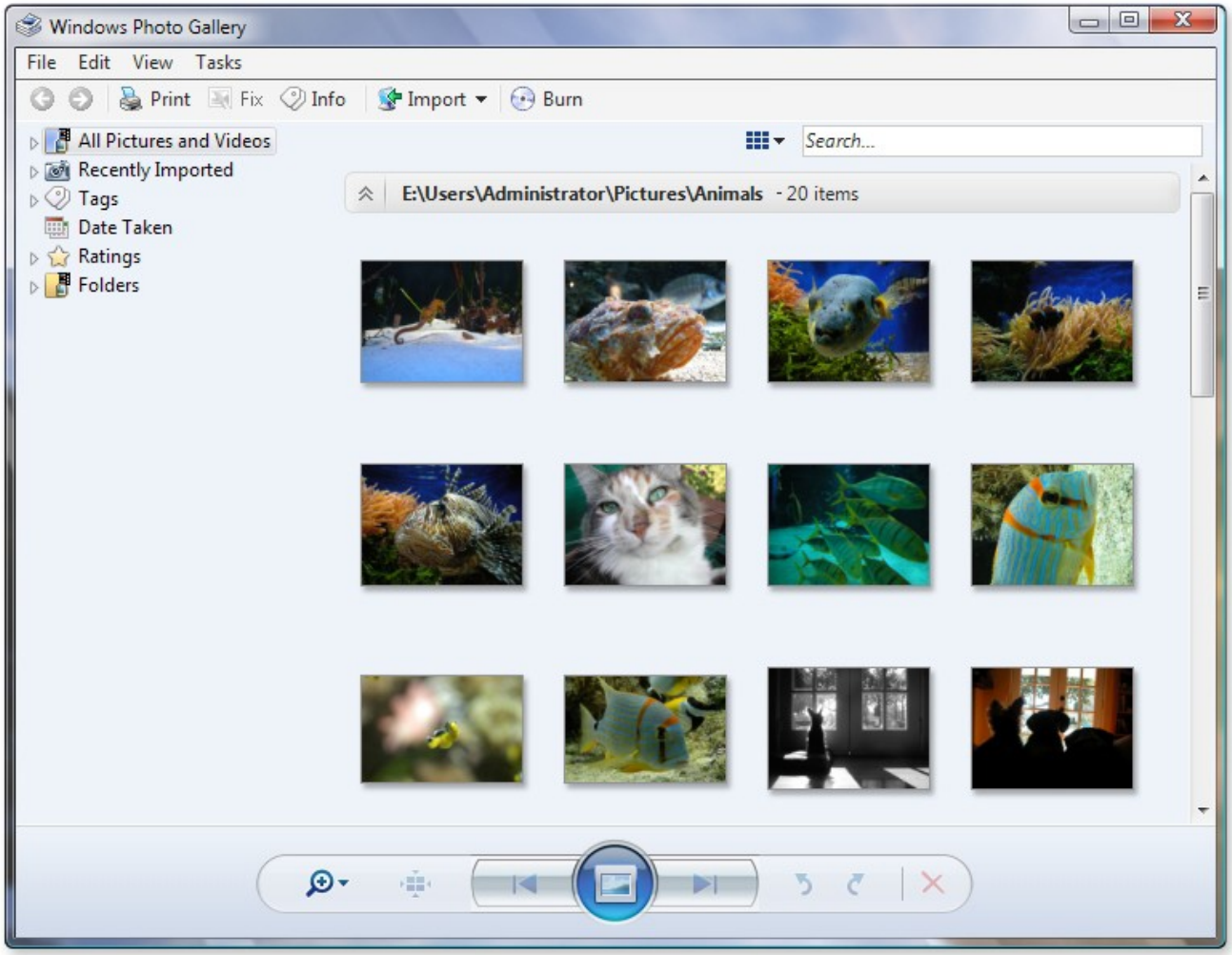
Outlook


Inbox - Microsoft Outlook
File Edit View Go Tools Actions Help
Type a question for help
New Reply Reply to All Forward Send/Receive Find Type a contact to find
Mail
Favorite Folders
Inbox
Unread Mail
For Follow Up
Sent Items
Inbox in imap.ksu.edu
All Mail Folders
Personal Folders
Deleted Items
Drafts
Inbox
Junk E-mail
Outbox
Sent Items
Search Folders
imap.ksu.edu
Mail
Calendar
Contacts
Tasks
Inbox
Arranged By: Date
Newest on top
Monday
Outlook 2003 Team
Mon 5:49 PM
Welcome to Microsoft Office Outlook 2003
Welcome to Microsoft Office Outlook 2003
Outlook 2003 Team [olteam@microsoft.com]
To: New Outlook User
Thank you for using Microsoft® Office Outlook® 2003! This version of Outlook includes new capabilities designed to help you access, prioritize, and act on communications and information so that you may use your time more efficiently and more easily manage the ever-increasing flow of incoming e-mail.
To give you some idea of what is possible with Outlook 2003, we have put together a list of our favorite new features.
Streamlined Mail Experience - Outlook helps you read, organize, follow-up, and find e-mail messages more efficiently than before. The new window layout displays more information on the screen at one time, even on smaller monitors. The list of messages has been redesigned to use space more intelligently; as a result, you will spend less time scrolling and more time getting your work done. Automatic grouping of your messages helps you find and go to e-mail
1 Item

OS X Mail


Inbox — Sun External (31 messages)
Get Mail
New
Reply
Reply All
Forward
Print
Delete
Junk
Search
Inbox
Marinacci.org
Sun External
Drafts 2
Sent
Trash
Junk 1122
Apple Java
JDNC / JDIC 4
Flying Saucer
Jen
Junk (Sun)
Marinacci.org 7
Sun External
ARCHIVE
Administration
BugFixes
Community
dolphin
GAF
Help Requests
JavaOne
Peabody
Personal
Reference
Shaped Windows
Translation
WinLAF
xhtmlJD
Sent
Shared Folders 243
Templates
Trash 10
Contains messages that are junk mail.
From
Subject
Date Received
Size
swing pending reviews on 05/10/2006 Today 8:21 AM 6.9 KB
Scott Violet Re: Please Read: All API changes to ... Today 7:29 AM 4.1 KB
swing Webrev approved - 6406070: Incons... Today 2:10 AM 4.1 KB
swing Webrev approved - 6420149: REGRE... Today 2:10 AM 4.3 KB
Peter Zhelezniakov Approved: [6406070] mustang appr... Today 2:05 AM 5.0 KB
Peter Zhelezniakov Approved: [6420149] mustang appr... Today 2:05 AM 5.1 KB
Romain Guy [Fwd: JavaOne Demo Dry Run Rehear... Yesterday 11:08 PM 10.0 KB
Subject: Re: Please Read: All API changes to Swing
Date: May 10, 2006 7:29:27 AM PDT
To: Jeff Dinkins
Cc: Swing Core , Ian Hardie <Ian.Hardie@Sun.COM>
Just one quick note to this. The doc for the windows/gtk/motify classes is not generated, so that if you're adding doc to any of these you don't need to notify l10n.
-Scott
Jeff Dinkins wrote:
From here until when we ship, we have just promised to the L10N team that we will let them know when we putback any API doc change. i.e. when the javadoc changes (and that include method additions).
Why? This is so that they can start their translation work early (now) without worrying that we'll change something out from under them.
So, if you make any additional changes to what's already in the master workspace as of today, please send a note to me (soon to be Brian) and we'll make sure that they get back to the L10N team. Where appropriate, please also cc Andrey.
thanks!
jeff

Inspiration: Vista



Agenda

Introduction

Last year's application, this year's application, inspiration!

Spicing Up the Main Window

Shadows, fancy rendering, drag and drop feedback

Tables, Lists, and of Course Text

Beautifying table, web style lists, and message folding

Search Dialogs and Sending Messages

Color mixing, frosting, and animation

DEMO

The Shadowy Depths, and a New Look for Buttons

The Shadowy Depths

- Multiple ways to generate shadows
- DropShadowBorder
  - Add a rectangular drop shadow to a component
  - `org.jdesktop.swingx.border.DropShadowBorder`
- ShadowFactory and DropShadowPanel
  - Non-rectangular shadows for opaque content
  - In the SwingLabs incubator project

DropShadowBorder

Constructor

```
public DropShadowBorder(
    Color lineColor, int lineWidth,
    int shadowSize, float shadowOpacity, int cornerSize,
    boolean showTopShadow, boolean showLeftShadow,
    boolean showBottomShadow, boolean showRightShadow)
```

Example

```
Border shadow =
    new DropShadowBorder(Color.BLACK, 0,
                         5, .5f, 12,
                         false, true, true, true);

c.setBorder(new CompoundBorder(shadow, c.getBorder()));
```

ShadowFactory

Constructor

```
public ShadowFactory(int size, float opacity, Color color)
```

Method

```
public BufferedImage createShadow(BufferedImage image)
```

Example—create a shadow for an image

```
ShadowFactory fac = new ShadowFactory(5, 0.2f, BLACK);
BufferedImage shadow = fac.createShadow(myImage);
```

Now the shadow and original image can be overlayed to generate the effect

DropShadowPanel

Extends JPanel—Just Add Your Components to It

Constructor

```
public DropShadowPanel()
```

Methods

```
public void setShadowFactory(ShadowFactory)
public void setAngle(float angle)
public void setDistance(int distance)
```

Example—Shadows behind our toolbar buttons

```
DropShadowPanel ds = new DropShadowPanel();
ds.setLayout(new BoxLayout(ds, BoxLayout.X_AXIS));
dsp.add(getMailButton);
dsp.add(composeButton);
dsp.add(addressBookButton);
```

CoolBar—Extends JToolBar

```
public CoolBar() {
   setBorder(new DropShadowBorder(/* bottom only */));
}

public void paintComponent(Graphics g) {
   Graphics2D g2 = (Graphics2D)g.create();
   int bottom = getHeight() - getInsets().bottom;
   GradientPaint gp =
      new GradientPaint(0, 0, GRAY, 0, bottom, WHITE);
   g2.setPaint(gp);
   g2.fillRect(0, 0, getWidth(), bottom);
   g2.setColor(BLUEISH);
   g2.drawLine(0, bottom - 1, getWidth(), bottom - 1);
   g2.dispose();
}
```

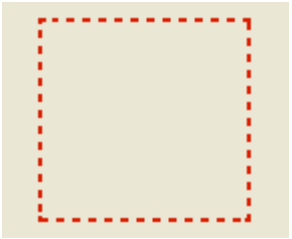
CoolButton—Extends JButton

Override `paintComponent(Graphics g)`

CoolButton—Extends JButton

Override `paintComponent(Graphics g)`

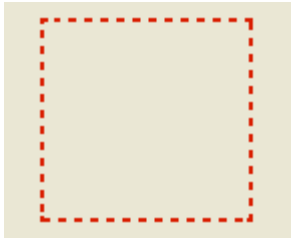
Initial Graphics clip is the full bounds of the component:



CoolButton—Extends JButton

Override `paintComponent(Graphics g)`

Initial Graphics clip is the full bounds of the component:

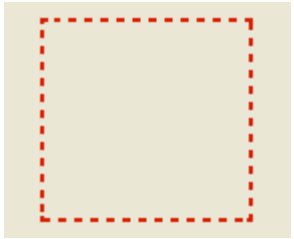


```
// create a rounded clip LARGER than the comp
RoundRectangle2D.Float r2d =
    new RoundRectangle2D.Float(
        0, 0, w + 30, h - 1, 20, 20);
```

CoolButton—Extends JButton

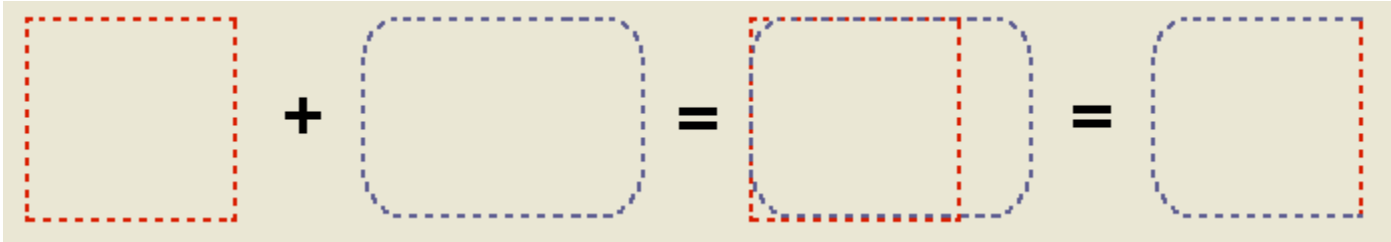
Override `paintComponent(Graphics g)`

Initial Graphics clip is the full bounds of the component:



```
// create a rounded clip LARGER than the comp
RoundRectangle2D.Float r2d =
    new RoundRectangle2D.Float(
        0, 0, w + 30, h - 1, 20, 20);

// intersect this with the existing clip
g2d.clip(r2d);
```



CoolButton—Extends JButton

```
// fill the clipped area
g2d.setPaint(LIGHT_GRADIENT);
g2d.fillRect(0, 0, w, h);
// restore original clip
```

CoolButton—Extends JButton

```
// fill the clipped area
g2d.setPaint(LIGHT_GRADIENT);
g2d.fillRect(0, 0, w, h);
// restore original clip
```

```
// paint outer border
g2d.setPaint(OUTER);
g2d.drawRoundRect(0, 0, w + 30, h - 1, 20, 20);
```

CoolButton—Extends JButton

```
// fill the clipped area
g2d.setPaint(LIGHT_GRADIENT);
g2d.fillRect(0, 0, w, h);
// restore original clip
```

```
// paint outer border
g2d.setPaint(OUTER);
g2d.drawRoundRect(0, 0, w + 30, h - 1, 20, 20);
```

```
// paint inner border
g2d.setPaint(INNER);
g2d.drawRoundRect(1, 1, w + 30, h - 3, 18, 18);
```

CoolButton—Extends JButton

```
// paint right outside border
g2d.setPaint(p1);
g2d.drawLine(w - 1, 1, w - 1, h);
```

CoolButton—Extends JButton

```
// paint right outside border
g2d.setPaint(p1);
g2d.drawLine(w - 1, 1, w - 1, h);
```

```
// paint right inside border
g2d.setPaint(p2);
g2d.drawLine(w - 2, 2, w - 2, h - 1);
```

CoolButton—Extends JButton

```
// paint right outside border
g2d.setPaint(p1);
g2d.drawLine(w - 1, 1, w - 1, h);
```

```
// paint right inside border
g2d.setPaint(p2);
g2d.drawLine(w - 2, 2, w - 2, h - 1);
```

```
// make it translucent
g2d.setComposite(AlphaComposite.getInstance(
    AlphaComposite.SRC_OVER, 0.1f));
```

CoolButton—Extends JButton

```
// paint the text and icon
super.paintComponent(g);
```

CoolButton—Extends JButton

```
// paint the text and icon
super.paintComponent(g);
```


```
// create shadow by adding groups of buttons
// to a DropShadowPanel
```

Fading by Way of `TimingController`

```
TimingController con = new TimingController(200, target);

addMouseListener(new MouseAdapter() {
    public void mouseEntered(MouseEvent me) {
        con.start();
    }
}
```

Fading by Way of `TimingController`

```
TimingTarget target = new TimingTarget() {
    public void timingEvent(long cycleElapsedTime,
                            long totalElapsedTime,
                            float fraction) {

        this.pct = fraction;
        int r = BLUE.getRed() + (int)
            ((WHITE.getRed() - BLUE.getRed()) * pct);
        int g = BLUE.getGreen() + (int)
            ((WHITE.getGreen() - BLUE.getGreen()) * pct);
        int b = BLUE.getBlue() + (int)
             ((WHITE.getBlue() - BLUE.getBlue()) * pct);
        setForeground(new Color(r, g, b));
        repaint();
}};
```

Fading by Way of `TimingController`

And of course, in the **`paintComponent`** method:

```
// base transparency is 0.1f
// fade towards fully opaque, based on pct
float tran = 0.1f + pct * 0.9f;

g2d.setComposite(AlphaComposite.getInstance
                 (AlphaComposite.SRC_OVER, tran));
```

DEMO

Tree Spice + DnD Spice

{and all things nice}

Custom Tree

- Nodes are drawn by a TreeCellRenderer with code similar to **`CoolButton`**
- Renderer also responsible for drawing stars to show new message count
- TimingController used to fade in these stars
- Custom UI class to relocate +/- control, draw connection lines, and give drag-over feedback

Relocate +/- Control

Subclass WindowsTreeUI

```
void paintExpandControl(..., Rectangle bounds, ...) {
    Rectangle transBounds = new Rectangle(bounds);
    transBounds.x += 44;
    transBounds.y += 10;
    super.paintExpandControl(..., transBounds, ...);
}

boolean isLocationInExpandControl(..., int mouseX,
                                       int mouseY) {
    return super.isLocationInExpandControl(...,
                                           mouseX - 44,
                                           mouseY - 10);
}
```

Relocate +/- Control

Subclass WindowsTreeUI

```
void selectPathForEvent(TreePath path, MouseEvent evt) {
    if (isLocationInExpandControl(path,
                                  event.getX(),
                                  event.getY())) {
        return;
    }

    super.selectPathForEvent(path, event);
}
```

Drag-Over Feedback for the Tree

Subclass WindowsTreeUI

```
protected void installListeners() {
    super.installListeners();
    tree.addPropertyChangeListener("dropLocation", this);
}

public void propertyChange(PropertyChangeEvent pce) {
    DropLocation dl = tree.getDropLocation();
    repaintFor(this.dropPath);
    this.dropPath = (dl == null) ? null : dl.getPath();
    repaintFor(this.dropPath);
}
```

Drag-Over Feedback for the Tree

Subclass WindowsTreeUI

```
public void paint(Graphics g, JComponent c) {
    super.paint(g, c);
    if (dropPath == null) { return; }

    int row = tree.getRowForPath(dropPath);
    Rectangle bounds = getRowBounds(row);
    bounds.x -= 8;
    bounds.y -= 8;
    bounds.width += 16;
    bounds.height += 16;
    paintRow(g, (Rectangle)g.getClip(),
             getInsets(), bounds,
             dropPath, row, true, true, true);
}
```

Drag and Drop Visual Representation

- Custom GlassPane is responsible for showing an image at a given location
- Custom TransferHandler: controls the visibility of this GlassPane, the image it shows, and placement of the image

First, The Custom GlassPane

```
class DragOverGlassPane extends JPanel {
    private Point p;
    private Image image;

    public DragOverGlassPane() {
        setOpaque(false);
        setVisible(false);
    }

    public void showIt(Image image, Point p) {
        this.image = image;
        setVisible(true);
        moveImageTo(p);
    }
```

First, The Custom GlassPane

```
    public void moveImageTo(Point p) {
        // point is in SCREEN co-ordinates
        SwingUtilities.convertPointFromScreen(p, this);
        repaintFor(this.p);
        this.p = p;
        repaintFor(this.p);
    }

    public void hideIt() { setVisible(false); }

    public void paintComponent(Graphics g) {
        if (image != null) {
            g.drawImage(image, p.x, p.y, null);
        }
    }
}
```

Install the Custom GlassPane

```
DragOverGlassPane gp = new DragOverGlassPane();
mainFrame.setGlassPane(gp);
```

The Custom TransferHandler

Uses the GlassPane to Show Drop Location

```
public class CoolTransferHandler extends TransferHandler
                        implements DragSourceMotionListener {

    public Transferable createTransferable(JComponent c) {
        DragSource.getDefaultDragSource()
                  .addDragSourceMotionListener(this);
        Point p = MouseInfo.getPointerInfo().getLocation();
        gp.showIt(icon, p);

        // then create and return Transferable
    }
```

The Custom TransferHandler

Uses the GlassPane to Show Drop Location

```
    public void dragMouseMoved(DragSourceDragEvent dsde) {
        gp.moveImageTo(dsde.getLocation());
    }

    public void exportDone(JComponent source,
                           Transferable data, int action) {
        DragSource.getDefaultDragSource().
            removeDragSourceMotionListener(this);
        gp.hideIt();
    }
}
```

Or, to Shrink and Fade on a Drop

```
public void exportDone(JComponent source,
                       Transferable data, int action) {
    DragSource.getDefaultDragSource().
        removeDragSourceMotionListener(this);

    new TimingController(300, this).start();
}

public void timingEvent(long l, long l0, float f) {
    gp.reduceItTo(1.0f - f);
}
```

Agenda

Introduction

Last year's application, this year's application, inspiration!

Spicing Up the Main Window

Shadows, fancy rendering, drag and drop feedback

Tables, Lists, and of Course Text

Beautifying table, web style lists, and message folding

Search Dialogs and Sending Messages

Color mixing, frosting, and animation

DEMO

Beautifying Table

Beautifying Table

- Grid lines are good for spread sheets, not so good for mail clients
  - `setShowHorizontalLines(false)`, `setShowVerticalLines(false)`
- White space is your friend
  - Default row height is 16, regardless of font
    - Up to a more reasonable value via `setRowHeight()`
  - Default renderer has 1 pixel border surrounding content—way too small
    - Provide a custom renderer for more space

Table Striping

- Makes it easier to follow wide rows
- Numerous approaches
  - Only a couple of lines of code with SwingX's JXTable
  - Make JTable non-opaque, override **`paintComponent`** and draw stripes
    - Making components non-opaque will result in needing to paint more, especially when scrolling
  - Create custom renderer that sets background color as appropriate
    - You'll have to replace all renderers to do this
    - If table smaller than viewport, striping will abruptly stop

Table Striping

Make the Table Non-Opaque

```
table.setOpaque(false);
```

Table Striping

Override paintComponent and Fill the Background

```
table.setOpaque(false);

protected void paintComponent(Graphics g) {
  Rectangle clip = g.getClipBounds();
  // Fill in the background
  g.setColor(getBackground());
  g.fillRect(clip.x, clip.y, clip.width, clip.height);
```

Table Striping

Draw the Stripes

```
protected void paintComponent(Graphics g) {
  // Code from last slide removed for more space
  // Draw the stripes.
  // For simplicity this assumes a uniform height
  int startRow = clip.y / rowHeight;
  int endRow = (clip.y + clip.height) / rowHeight + 1;
  g.setColor(STRIPE_COLOR);
  for (int row = startRow / 2 * 2; row < endRow;
       row += 2) {
    g.fillRect(0, rowHeight * row, w, rowHeight);
  }
```

Table Striping

Invoke super to Paint the Content

```
protected void paintComponent(Graphics g) {
  // Code from last slide removed for more space.
  // Draw the stripes.
  int startRow = clip.y / rowHeight;
  int endRow = (clip.y + clip.height) / rowHeight + 1;
  g.setColor(STRIPE_COLOR);
  for (int row = startRow / 2 * 2; row < endRow;
       row += 2) {
    g.fillRect(0, rh * row, w, rh);
  }
  super.paintComponent(g);
}
```

Highlighting Rows

- Similar approach to that of striping
  - Override **`paintComponent`**, and fill in background for necessary rows
- Painting done with a gradient and round rectangle

```
g.setPaint(new GradientPaint(
                x, y, color1, x, y + h, color2));
g.fillRoundRect(x, y, w, h, arcSize, 12);
```

Beautifying Table

- Looks better, but hardly extreme
- Many apps moving toward a richer, Web-like look
  - No more table
  - List with content wrapped in a cell
  - Works well when you can use images

DEMO

Extreme List

Extreme List

- Custom ListCellRenderer
  - ImagePanel for showing image
  - JLabels for subject, date, content, sender
- Use your LayoutManager(s) of choice
  - Grid based layout managers work well for this
  - GroupLayout works too
- Make sure you use prototype cell value, or fixed cell height, else performance will be impacted

```
list.setPrototypeCellValue(...);
```

Extreme List

- Only trick is in flowing the text
- JLabel doesn't wrap for you
- Implemented as two JLabels
  - Iterate through characters using **`Graphics.charsWidth`** to determine wrap location
  - Once wrap location is determined, set text on JLabel

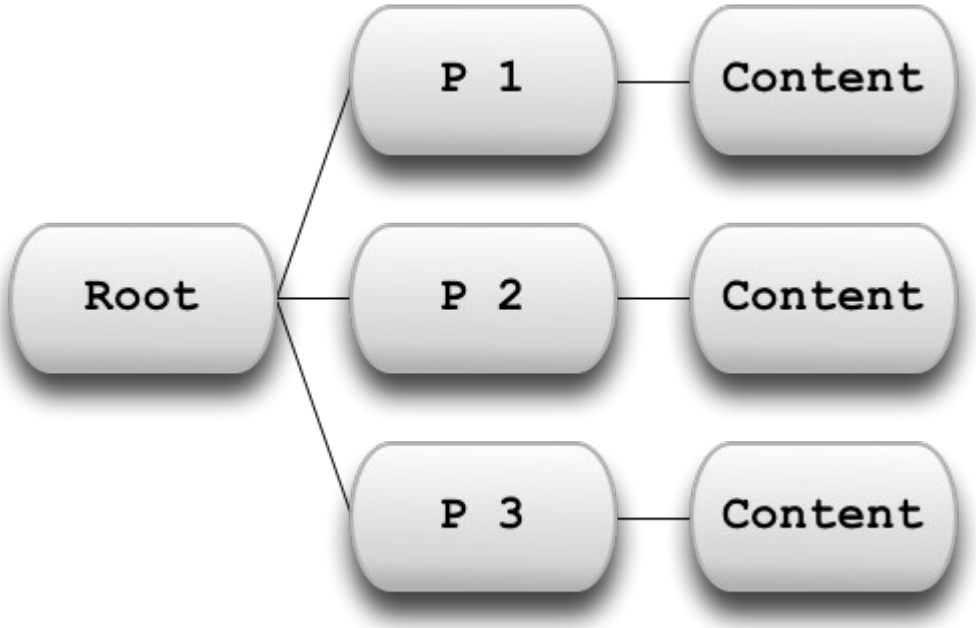
DEMO

Message Folding

Message Folding

- JTextPane subclass
- Custom document structure to represent messages
- Custom View to render indentation level
  - Ability to hide/show folded message
  - Animates using timing framework
- Provides message path property that is updated based on visible location
  - Separate component listens for changes and displays path appropriately

Default Styled Text Element Structure

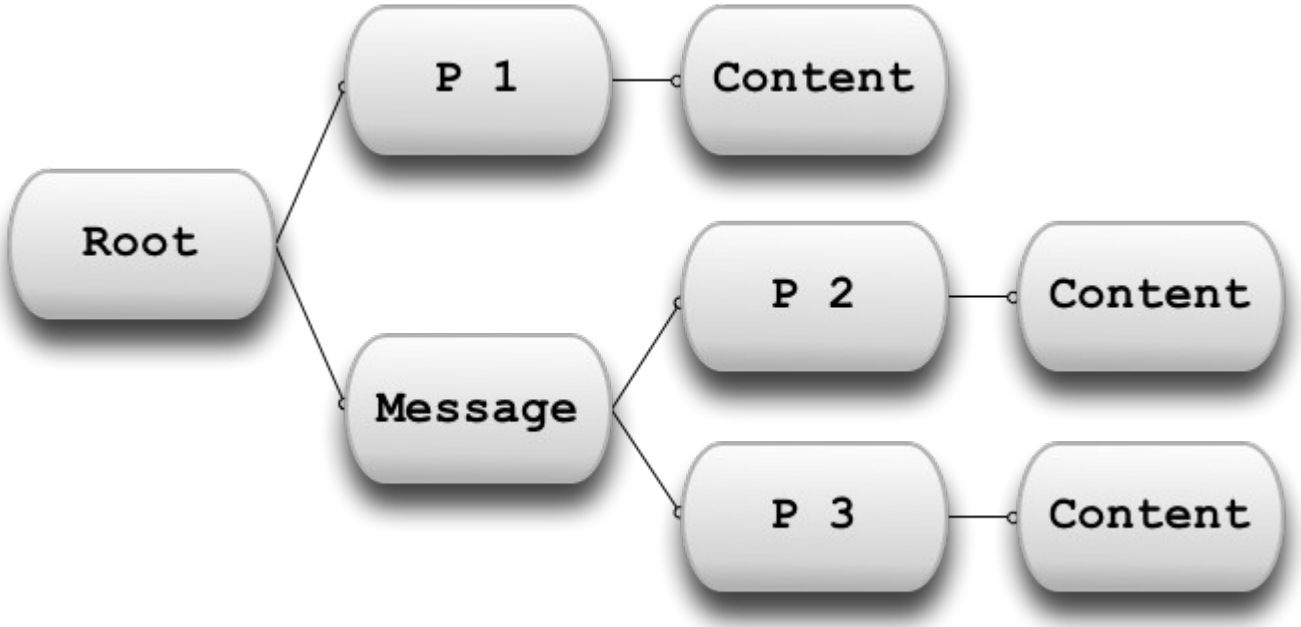


First paragraph

Second paragraph

Third paragraph

Message Folding Element Structure

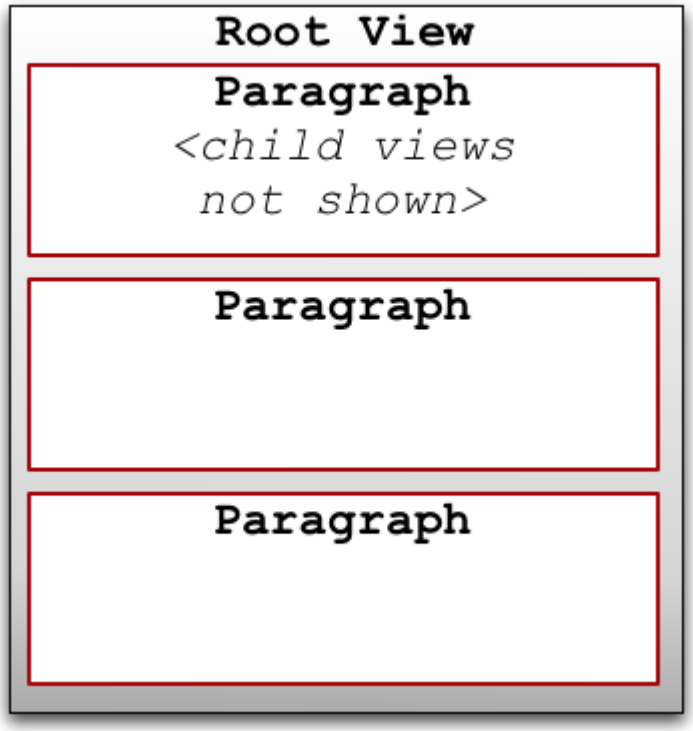


First paragraph

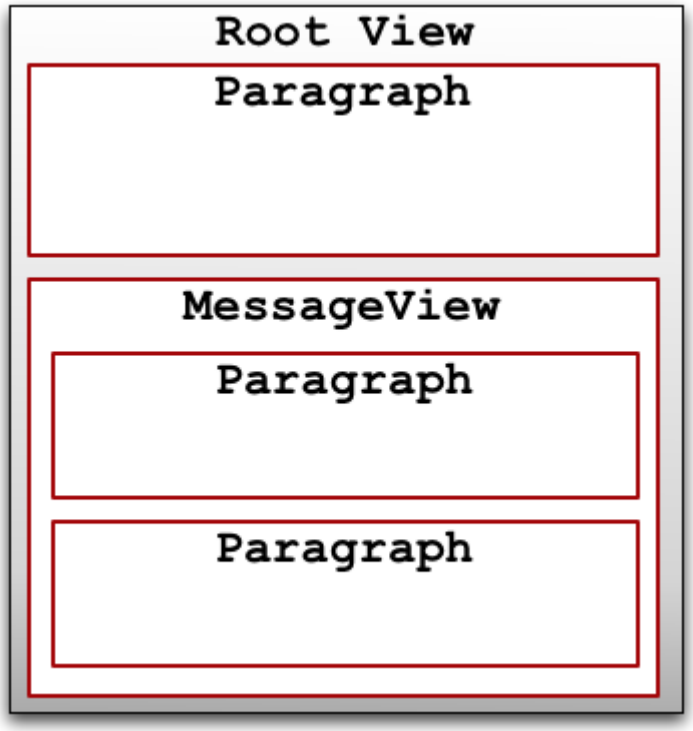
> message

> message

Default Styled Text View Structure



Message Folding View Structure



Text View

- View is very similar to Container
- Has methods for obtaining preferred, minimum and maximum size
- Views are responsible for painting themselves, and their children
- Views are responsible for positioning and sizing any children Views

MessageView

- Extends BoxView
  - Superclass methods still used for layout and calculating expanded size
- Paints expand/collapse icon
- Paints the line along the left side of the View
- Indents children
- Overrides sizing methods to return a fixed value when collapsed
  - When animating, size is varied over time

MessageView: Painting Icon and Line

```
// The paint method is defined by View. The paint
// method is passed the bounds of the View.
public void paint(Graphics g, Shape bounds) {
  // Paint the icon
  Icon icon = getIcon();
  Point iconLoc = getIconLocation(bounds);
  icon.paintIcon(null, g, iconLoc.x, iconLoc.y);

  // Fill in the line
  Color lineColor = getLineColor();
  Rectangle lb = getLineBounds(bounds);
  g.setColor(lineColor);
  g.fillRect(lb.x, lb.y, lb.width, lb.height);
```

MessageView: Painting Children

```
public void paint(Graphics g, Shape bounds) {
  // Code for painting line and icon on previous slide
  if (isCollapsed()) {
    // If collapsed, constrain the clip to avoid
    // hidden views painting.
    Graphics childG = g.create();
    childG.clipRect(bounds.x, bounds.y, bounds.width,
                    fixedHeight);
    super.paint(childG, bounds);
    childG.dispose();
  } else {
    super.paint(g, bounds);
  }
}
```

MessageView: Constraining Size

```
// getPreferredSpan is the equivalent of
// getPreferredSize, along a particular axis.
public float getPreferredSpan(int axis) {
  if (axis == Y_AXIS && isCollapsed()) {
    // User has collapsed this view; return a fixed
    // height. When animating this value will vary.
    return height;
  }
  return super.getPreferredSpan(axis);
}

// getMinimumSpan and getMaximumSpan overriden in
// same manner.
```

Agenda

Introduction

Last year's application, this year's application, inspiration!

Spicing up the Main Window

Shadows, fancy rendering, drag and drop feedback

Tables, Lists, and of Course Text

Beautifying table, web style lists, and message folding

Search Dialogs and Sending Messages

Color mixing, frosting, and animation

Search Made Extreme

- Find Dialogs are common to many applications
- Very simple GUI
  - Two minutes with NetBeans™ IDE GUI builder
- Located atop the search area
  - Search results are hidden
- Very distracting
  - Pop-ups are an annoyance
  - Where's the text!
- Very boring

DEMO

Classic Search Dialog

Collapsible Search Bar

- Located at the bottom of the search area
  - Search results are visible
- Easy to implement
  - SwingX search components
- Easy to spot
- User's work flow is not disrupted
  - Search can be performed on a keystroke
- Animated arrival

DEMO

Collapsible Search Bar

Collapsible Search Bar, the Code 1/2

```
search = new JXSearchPanel();
collapser = new JXCollapsiblePane();
collapser.getContentPane().add(search);

frame.add(messageView, BorderLayout.CENTER);
frame.add(collapser, BorderLayout.SOUTH);
```

Collapsible Search Bar, the Code 2/2

```
// or keyTyped(), etc.
public void actionPerformed(ActionEvent e) {
  boolean collapsed = collapser.isCollapsed();
  collapser.setCollapsed(!collapsed);
}
```

Extreme Search Dialog

- Animated open and close
  - Fade in/out
- Translucent
  - The user knows what is behind
  - Frosty background improves readability
- Drop shadow
- Glowing title
- Inspired by Windows Vista

DEMO

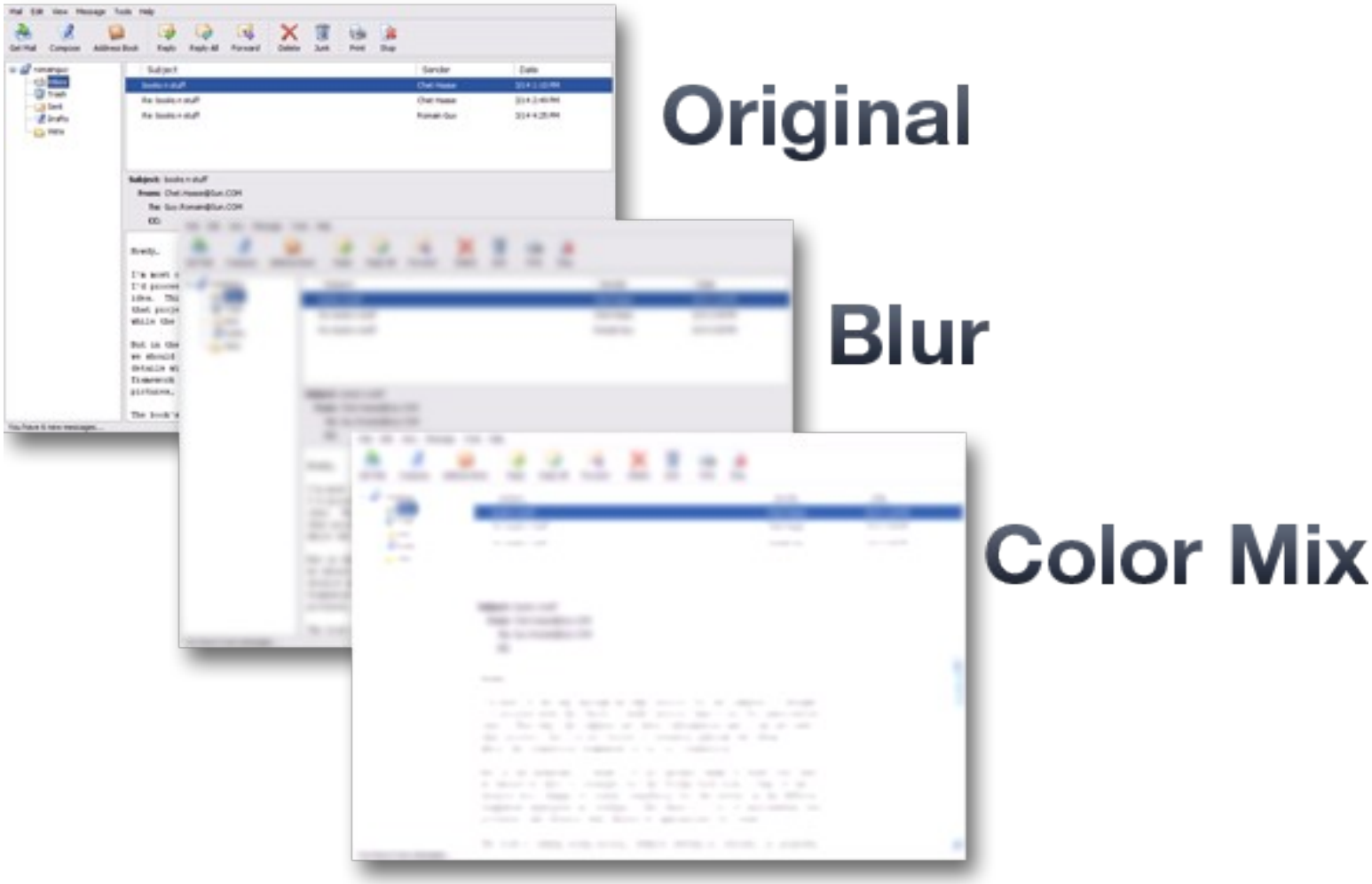
Extreme Search Dialog

Extreme Search Dialog, Hierarchy

Extreme Search Dialog, Hierarchy

```
private void install(JComponent component) {
  layeredPane = getRootPane().getLayeredPane();
  layeredPane.add(component,
      JLayeredPane.PALETTE_LAYER, 20);
  Dimension size = component.getPreferredSize();
  component.setSize(size);
  component.setLocation(
      (getWidth() - size.width) / 2,
      (getHeight() - size.height) / 2);
  component.setVisible(true);
}
```

Extreme Search Dialog, the Frosting



Extreme Search Dialog, the Frosting

- Blur
  - ConvolveOp and kernel
  - Pre-built filters
    - JH Labs, http://www.jhlabs.com/ip/filters/
  - Very expensive operation
    - Optimization by image scaling
- Color mix
  - Mix factor (e.g., 40% of white is added)
  - newColor = (1 - factor) * oldColor + factor * mixColor

Extreme Search Dialog, the Frosting

```
Container content = frame.getRootPane();
int width = content.getWidth() + 2 * BLUR_SIZE;
int height = content.getHeight() + 2 * BLUR_SIZE;
image = createImage(width, height);

Graphics2D g2 = image.createGraphics();
g2.translate(BLUR_SIZE, BLUR_SIZE);
content.paint(g2);
g2.translate(-BLUR_SIZE, -BLUR_SIZE);
g2.dispose();
```

Extreme Search Dialog, the Frosting

```
int width = image.getWidth();
image = createThumbnail(image, width / 2);
gaussian = new GaussianFilter(BLUR_SIZE)
image = gaussian.filter(image, null);

colorMix = new ColorMixerFilter(
    Color.WHITE, 0.4f);
image = colorMix.filter(image, null);
image = createThumbnail(image, width * 2);
```

Extreme Search Dialog, Painting

```
protected void paintComponent(Graphics g) {
  Point location = getLocation();
  location.x = (int) (-location.x - BLUR_SIZE);
  location.y = (int) (-location.y - BLUR_SIZE);

  Insets insets = getInsets();
  Shape oldClip = g.getClip();
  g.setClip(insets.left, insets.top,
      getWidth() - insets.left - insets.right,
      getHeight() - insets.top - insets.bottom);
  g.drawImage(image, location.x, location.y, null);
  g.setClip(oldClip);
}
```

Agenda

Introduction

Last year's application, this year's application, inspiration!

Spicing up the Main Window

Shadows, fancy rendering, drag and drop feedback

Tables, Lists, and of Course Text

Beautifying table, web style lists, and message folding

Search Dialogs and **Sending Messages**

Color mixing, frosting, and animation

Animated Sending

- Mail clients spend a lot of time sending mail
  - Some show a progress bar … boring!
  - Some show status text … even more boring!
  - Some work in the background … frustrating!
- Show what the application is doing
  - Showing progress is dull
  - But showing the mail being sent is … extreme!
- Animated graphics are the solution

DEMO

Animated Sending

Animated Sending

- Pure Java 2D API
- Relies heavily on the Timing Framework
  - Attend our presentation yesterday (TS-1297)
- Divide to conquer
  - Scale down the message
  - Fold the message
  - Put the message in the envelope
  - Fold the envelope
  - Seal the envelope
  - Send the envelope

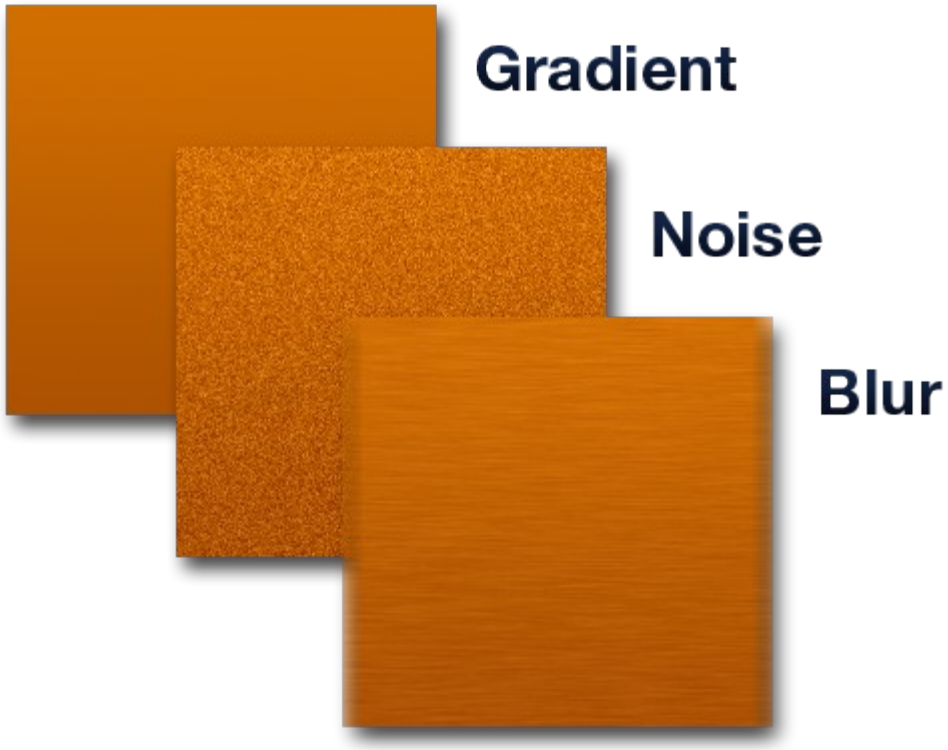
Animated Sending, Message Folding

```
range = PropertyRange.createPropertyRangeFloat(
    "folded", 0.0f, 1.0f);
ObjectModifier target = new ObjectModifier(
    this, range);
controller = new TimingController(
    cycle, envelope, target);
controller.addTimingListener(new TimingListener() {
  public void timerStarted(TimingEvent e) { }
  public void timerStopped(TimingEvent e) {
    foldEnvelope();
  }
  public void timerRepeated(TimingEvent e) { }
});
controller.start();
```

Animated Sending, Message Folding

```
Shape oldClip = g2.getClip();
int clipHeight = (int) (folded * height / 2.0f);
if (folded > 0.0f) {
  g2.setClip(new Rectangle(x, y + clipHeight,
                           width, height));
}
g2.drawImage(image, x, y, width, height, null);
if (folded > 0.0f) {
  g2.setClip(oldClip);
  g2.setPaint(foldGradient);
  g2.fillRect(x, y + clipHeight - 1,
              width, clipHeight + 1);
  g2.setPaint(oldPaint);
}
```

Animated Sending, Wood Background



Animated Sending, Wood Background

```
noise = new NoiseFilter();
noise.setDistribution(NoiseFilter.GAUSSIAN);
noise.setMonochrome(true);
blur = new WrappedBoxBlurFilter();

gradient = new BasicGradientPainter(woodGradient);
gradient.setEffects(new ImageEffect(noise),
                    new ImageEffect(blur));
gradient.setUseCache(true);

// using SwingX JXPanel
messageBackground.setBackgroundPainter(gradient);
```

Summary

- Take advantage of the power of Swing and Java 2D™API to create visually stunning and rich applications
- Timing framework and SwingLabs make it easier to add effects to your application
  - It's only going to get easier going forward
- Don't be boring—make your application fun!

For More Information

- SwingLabs
  - http://swinglabs.org/
- Timing Framework
  - http://timingframework.dev.java.net
- JavaDesktop
  - http://javadesktop.org
- Blogs
  - http://www.jroller.com/page/gfx
  - http://weblogs.java.net/blog/gfx
  - http://weblogs.java.net/blog/shan_man
  - http://weblogs.java.net/blog/zixle/

Q&A

Extreme GUI Makeover: Lookin' Better

Scott Violet
Shannon Hickey
Romain Guy

Sun Microsystems, Inc.
http://java.sun.com

TS-1548