Debugging X Applications: Introduction

What this talk is about:

- Sources of Complexity in X11
- Programming in the Face of Complexity
- Defect Avoidance
- Defect Isolation and Repair

What this talk is not about:

- Not about any particular toolkit
- Not a laundry list of bugs and fixes

Instead, we will try to acquire a feel for the kind of bugs that occur, when and why they occur, how to avoid them when you can, and how to find them when you can't.

Generic Issues in GUI Development

GUI ’s Defy Traditional S/E Techniques

- Traditional S/E waterfall looks like this:

Requirements Definition
Software Design
Coding
Unit Test
Integration/System Test
Ship
(Maintenance)

- GUI developments looks like this:

(Requirements Definition)
GUI Design and Prototype
Test Usability and Refine
Test Usability and Refine
...ad infinitum
Ship
(Maintenance)

What to do?

- Component-level design and testing
- Tackle the hard parts (custom widgets) first

Generic Issues in GUI Development

GUI's are Asynchronous and Event-Driven

- Makes black-box testing difficult
- White-box testing is easier, if you know what to look for, but...
- In both cases, the number of possible inputs is large, because the user can do anything from anywhere at any time!

No truly good solutions for this problem, but:

- Programming defensively can help
- Be especially careful with Actions in X11
  (more about them later!)

Generic Issues in GUI Development

GUI Toolkits are Data Oriented instead of Procedural

- Most toolkits use callbacks and callback data

  Control flow isn't apparent from the call graph; it's embedded in the data.

- Callback data passed in by one part of the program, retrieved by another

  Invalid data isn't detected until long after the damage has occurred.

- Hard to test for in QA!

  Too many possible permutations of user actions; bugs can lay dormant for *years.*

Generic Issues in GUI Development

Defensive Programming for Data Oriented Toolkits

- Instrument code to show function entry/exit; tells you if you program is executing the code you think it's executing

- Embed magic values in callback data

  Write macros to test validity of data; if you use struct GraphNode as callback data, write:

  ```
  #define IsGraphNode(gn) ((gn)->magic == GNODE_MAGIC)
  ```

  Use assert() [/usr/include/assert.h] to check validity before passing data into toolkit and after getting it back:

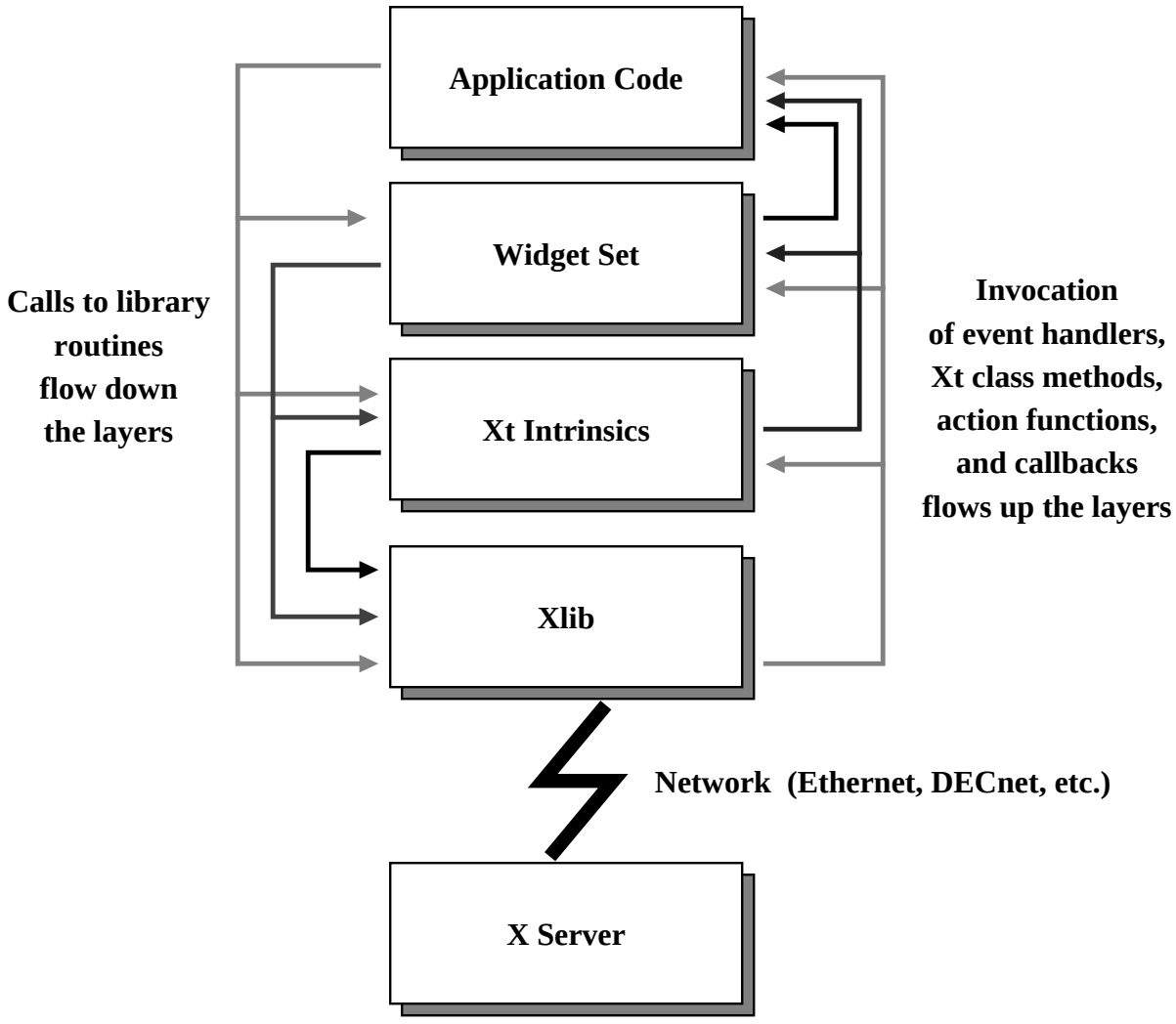
  ```
  void my_callback(..., void *data)
  {
  struct GraphNode *nodep;

  nodep = (GraphNode *)data;
  assert(IsGraphNode(nodep));
  ...
  }
  ```

- Include originator-id's in callback data; helps to figure out where bad data came from

- Use programming tools (CodeCenter/ObjectCenter, Purify) that can find bad pointers at runtime

X11: Layers of Abstraction

X Involves Layers of Implementation & Abstraction



X11: Layers of Abstraction

Many Layers of Abstraction Introduce Complexity:

- Layers obscure the source of errors

  Valid requests at high levels (Toolkit, Intrinsics) can generate errors at low levels (Xlib, server) without any indication where the error originated.

- Many possible vendors/versions of each layer
- Different vendors/versions have different bugs
- Implementation differences can mask bugs

```
Arg     args[3];
short   width, height;   /* Oops! */
int     i = 0;

XtSetArg(args[i], XtNwidth, &width); i++;
XtSetArg(args[i], XtNheight, &height); i++;
XtGetValues(widget, args, i);
```

  This code works on some systems, not on others. Can you tell why?

- Shared libraries can introduce incompatibilities *at runtime*

X11: Layers of Abstraction

Coping With Layers of Abstraction:

- Be clear about what versions you support
- Watch for shared library problems at runtime
    - Widget version mismatch messages

      Warning: Widget class version mismatch:
      widget 11004 vs. intrinsics 11003

    - Shared library version mismatch messages

      ld.so: warning:
      /apps/openwin3.0/lib/libXmu.so.4.0 has
      older revision than expected 10

      ld.so: Undefined symbol: _XmuNewCvtStringToWidget

    - Use ldd to check which libs are used

      <rapier 113>% ldd colors
      -lOI.3 => /tools/OI3.0/sparc/lib/libOI.so.3.0
      -lX11.4 => /apps/openwin3.0/lib/libX11.so.4.3
      -lC.1 => /usr/local/CenterLine/sun4/lib/libC.so.1.1
      -lc.1 => /usr/lib/libc.so.1.6

    - Watch for bad LD_LIBRARY_PATH settings

Unfortunately, there is no good answer to the problem of multiple versions.

X11 Toolkit Issues

Toolkits Introduce New Complexities

- X11 toolkits are virtually all data driven
  (We have already seen what this implies.)
- Toolkits add new "hierarchies of confusion"
  - Widget/Object class hierarchy
  - Instance hierarchy
  - Potentially separate Window hierarchy
- Multiple hierarchies make predicting and understanding behavior nontrivial

Toolkits: Widget/Class Hierarchy

Toolkits usually try to be object oriented, but they are not usually written in object oriented languages.

- Widget hierarchy: like class hierarchy in C++
- Inheritance in the Widget hierarchy influences
    - behavior - callback functionality inherited
    - resource usage
- Inheritance can obscure the where behavior is implemented (i.e., where bugs come from)
- Getting details right should be the widget writer's job (but we know it doesn't always work that way)
- Understanding the Widget hierarchy would benefit from tools like those used with OO languages

Toolkits: Widget/Class Hierarchy

Toolkits: Widget/Class Hierarchy

Tools for Exploring the Widget Hierarchy:

- listres: displays a widget hierarchy

```
<rapier>% listres
asciiSink     Object\TextSink\AsciiSink
asciiSrc      Object\TextSrc\AsciiSrc
box           Object\Rect\Core\Composite\Box
...
```

Listres can also display which resources are used by each class:

```
<rapier 33>% listres label
 WidgetClass      Instance   Class         Type
 -----------      --------   -----         ----
 ...
 Core             background    Background   Pixel
 Core             background    Pixmap       Pixmap
 Label            bitmap        Pixmap       Bitmap
 ...
```

Use -nosuper to filter out inherited resources.

- Debuggers: explore the hierarchy interactively

Tip: Routines in wdiag.c can help expose a widget's inheritance path. (Thanks to Ollie Jones!)

Toolkits: Instance Hierarchy

Toolkits also have Instance Hierarchies

- Functions like instance hierarchy in C++
- Made up of objects/widgets you create
- Instance hierarchy == window hierarchy (usually!)

Instance hierarchy influences:

- Position and size on the screen
  (constraints, geometry, clipping)
- Visibility
- Resource lookup
  (Resource name reflects the instance hierarchy)

Some toolkits separate the instance hierarchy from the window hierarchy

- Adds flexibility for the application writer
- Means you may need to understand the relationship between instance tree and window tree

Toolkits: Instance Hierarchy

Effects of Problems in the Instance Tree

- Size/position/visibility arguments set for an object don't "take", because the parent won't let them.
- User-supplied resource settings can have unexpected and pathological effects on objects the user didn't intend them to apply to. (More about this later...)

Tools for exploring the instance tree:

- xlswins: lists all windows on a display
- xwininfo: lists attributes of a window

  Useful for finding out about depth, colormap, backing store, save unders, map state, and events solicited.

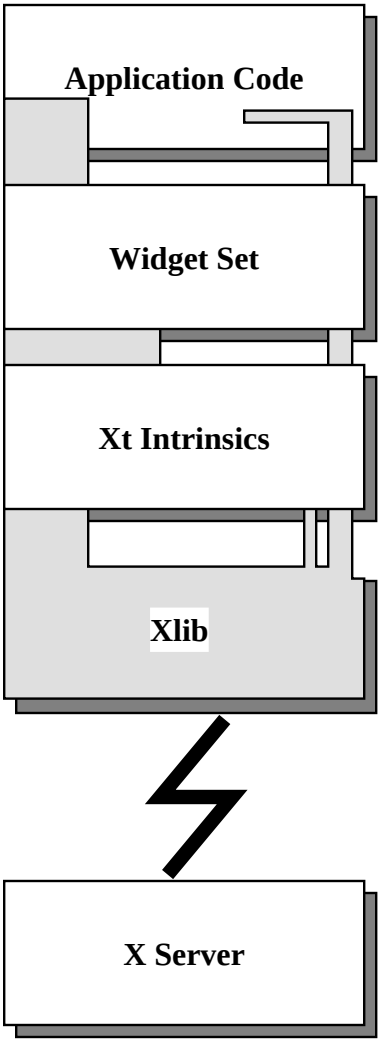
  Tip: Map State "IsUnviewable" means you are mapped, but one of your ancestors isn't.

- xprop: lists properties of a window

  Useful for finding out what the application class is, what WM protocols the program participates in, what private protocols the program participates in (OL/Motif window decoration, for example)

Xlib Issues

Toolkits try to hide Xlib, but none of them really do so. Every toolkit has holes through which Xlib shines:



(The picture makes it look cleaner than it is!)

Xlib Issues

Using Xlib Adds More Complexity

- Xlib programming interface is complex
- Need to understand your toolkit, Xlib, *and how they work together*

  Example: What things can you do to a Window that is "owned" by your toolkit? Can you:

  - Set properties/attributes on it?
  - Change its colormap?
  - Resize it?

  Answers all depend on the toolkit in question.
- Rule of thumb: Don’t use Xlib for anything that has a library call in your toolkit.

Xlib Issues

Xlib programming interface is complex and low-level, leading to many common programming mistakes.

Simple things to watch for:

- Common event processing bugs

  ```
  dpy = XOpenDisplay(...);
  win = XCreateSimpleWindow(...);
  gc = XCreateGC(dpy, win, 0, &gcv);
  XMapWindow(dpy, win);
  XDrawLine(dpy, win, gc, 0, 0, 50, 50);
  ```

  This code has (at least) two bugs in it...

- Order of creation errors

  ```
  widget = XtCreateManagedWidget(...);
  gc = XCreateGC(dpy, XtWindow(widget), 0, &gcv);
  ```

  BadDrawable error: widget has not been realized yet, so XtWindow returns None

- Assumptions about pixel values

  Canonical example is rubber banding or using xor to draw and erase lines. Naive code simply xor's the foreground and background pixels together. Why doesn't this work? (Hint: what happens if BlackPixel is 0 and WhitePixel is 1?)

Xlib Issues

More Common Xlib Problems:

- Image construction/copying errors
    - Confusion about XCopyArea/XCopyPlane leads to code that works on monochrome displays, but not on color displays
    - Confusion about the use of the gc function, plane mask, and pixel values leads to code that draws incorrectly
    - Assuming that XCreatePixmap clears the pixmap leads to applications that draw garbage
    - Too many more to list!
- Assumptions about server depth, colormap type are dangerous:
    - Not all N-bit visuals are color, or have writable colormaps
    - Not all visuals have default colormaps

      Tip: When creating a Window on a pseudocolor visual that is not the default visual of the screen, you must find or create an appropriate colormap.

Xlib Issues

In some sense these are mostly "easy" bugs

- Erroneous code usually fails consistently
- Can usually catch drawing errors by testing on two or three different servers with different depths, display types (monochrome, color), and visual types
- Many common problems will generate X errors from the server

But...

- Asynchronous error reporting in Xlib tends to trap errors in the wrong place

Xlib Issues

Asynchronous Error Reporting in Xlib:

- Xlib buffers X11 requests internally
- Improves efficiency, but delays error reporting

Example of an X protocol error report:

```
X Error of failed request: BadDrawable
  Major opcode of failed request: 55
  Minor opcode of failed request: 0
  Serial number of failed request: 77
  Current serial number in stream: 81
  ...
```

Note the two serial numbers: 4 more requests were made and buffered up in Xlib after the one that actually caused the error.

In normal operation, virtually all X protocol errors will be reported "late".

Xlib Issues

Coping With Asynchronous Error Reporting

Several easy techniques for coping:

- Run application with -synchronize flag, or
- Provide a command line flag that calls XSynchronize at startup.

When running in a debugger:

- Call XSynchronize from the debugger, or
- Set the variable _Xdebug to a nonzero value *before calling XOpenDisplay*
- Install your own X error handler and set a breakpoint on it, or set a breakpoint on the default handler in Xlib

Example (using ObjectCenter)

```
4-> int MyErrorHandler(Display *, XErrorEvent *)
5-> {
6->     centerline_stop("");
7->     return 1;  /* continue execution */
8-> }
9->
9-> (void)XSetErrorHandler(MyErrorHandler);
```

X11 Server Issues

Two Kinds of Server Issues Commonly Arise

- X11 Server Bugs
    - All software has bugs
    - (Almost) all X11 servers are software
    - Therefore, all X11 servers have bugs?

  Not really, but some servers do have bugs that you have to work around. Identify server bugs by testing on other servers and using process of elimination.

  Using xscope to monitor X11 protocol traffic can help elucidate precise server behavior.

- Differences in Server Capabilities

  X11 Protocol specifies that many aspects of behavior are implementation dependent. These are the ones that most often trip programs up.

X11 Server Issues

Differences in X11 Server Capabilities

Four areas in which server implementation decisions and limits become important:

- Maximum Request Size
    - Can cause many requests to fail with server errors

      Tip: XDrawLines and XDrawArcs are cases to watch out for. Drawing too many lines or arcs can generate a BadLength error.

    - Some Xlib routines try to allow for server request size limits, while others don't; this is all implementation-defined, so best not to rely on any assistance here
    - Restrictions are usually undocumented (but O'Reilly books do cover them)
- Backing Store and Save Unders
    - Cannot assume they are available. Even if server offers them, it may withdraw them at any time

X11 Server Issues

More Implementation-Defined Server Capabilities:

- Resource Allocation Limits
  - Servers may run out of memory
  - Servers may place arbitrary restrictions on the number and/or size of Windows, Pixmaps, and other server resources

    Tip: Some servers prohibit you from creating a Pixmap bigger than the physical screen. They are trying to make optimal use of graphics hardware for pixmap cache, but end up making life difficult for application writers.

    If you're a server vendor, *don't do this.*

X11 Server Issues

Even More Implementation-Defined Server Capabilities:

- Handling of Invalid Protocol Requests
    - Some servers are more forgiving than others about bad requests (drawing rectangles with width or height of zero, creating windows with width or height of zero)
    - MIT servers are particularly forgiving, which masks application bugs that will appear with other servers
    - Be sure you test on some vendor supplied servers. OpenWindows and DECWindows servers are both good (or bad) about catching invalid requests

Environment Issues

Almost every X11 program interacts with other programs, whether it wants to or not. Interaction comes from three main areas:

- X servers *(which we have already looked at)*
- Window Managers
- Selection Transfers to/from other programs
  (also known as "cut and paste")

Environment Issues: Window Managers

Window Managers Impact Clients in Several Ways

- Geometry requests and constraints
- Virtual root windows
- Colormap installation
- Keyboard input problems
- Standard I/O and controlling tty issues
- Window manager key/button grabs

Environment Issues: Window Managers

Geometry Requests and Constraints

- Clients must be prepared for whatever sizes and locations they get
- Attempts to draw outside your window are silently dropped - looks to the user like a program bug

Virtual Root Windows

- Most applications don't care, but some do
- Virtual window managers work in different ways:
    - tvtwm/swm use window properties, so programs can cope by using vroot.h
    - HP VUE uses virtual root windows, but programs cannot find out about them
    - olvwm does not use virtual root windows
- If you play games with the root window, and they don't work in swm, tvtwm, olvwm, or VUE, suspect problems with virtual root windows.

Environment Issues: Window Managers

Colormap Installation

- ICCCM specifies how colormaps are to be installed
  - XSetWindowColormap for top level colormap
  - XSetWMColormapWindows allows different colormaps on subwindows.
  - Never call XInstallColormap yourself.
- Compatibility issues: non-ICCCM window managers don't know about mechanisms.

Tools for investigating colormap problems:

- xwininfo: tells what Colormap a window uses
- xprop: look for WM_COLORMAP_WINDOWS
- xshowcmap: display default colormap contents

Environment Issues: Window Managers

Keyboard Input Problems

- Make sure you tell the WM you want input
    - XtNinput resource in Widget sets
    - XSetWMHints w/InputHint for others

  (Use xwininfo -all to check the Input hint)
- WM may ignore hints set after mapping
- WM may grab buttons and keys

Standard I/O and Controlling TTY Issues

- Programs run from window manager menu may have different behavior
    - Environment variables may not be set
    - Standard I/O may be closed
    - Controlling tty may be absent
- Test programs started from shell and programs started from WM menu. Do they work the same way?

Environment Issues: Selections

The Selection Mechanism (Greatly Simplified)

- Selections are named by Atoms; there may be infinitely many selections
- Each selection has at most one owner.
- Requesters ask for the contents of a particular selection, usually in a particular format

Overview of Selection Transfer (Greatly Simplified)

- Program A informs server that it owns a particular selection (e.g. "CLIPBOARD")
- Program B asks for the CLIPBOARD selection

  Asks for a particular selection and format; provides a window and property on which it wants the data reported; continues execution

- Server sends Program A a SelectionRequest event
- Program A responds by placing data in the named property on the named window (may generate a PropertyNotify event)
- Program A sends SelectionNotify when done
- Program B handles SelectionNotify event by retrieving the selection data

Many additional complexities not mentioned here...

Environment Issues: Selections

Traps and pitfalls for the selection owner:

- Server maximum request size
- Allocation errors: server may run out memory storing selection data in property
- Race conditions

Traps and pitfalls for the selection requester:

- Selection failures: no owner of a selection
- Conversion failures: data format not available
- Allocation errors
- Race conditions
- GUI "standard" toolkits use selections differently

Common problem: Using *CurrentTime* in Xlib calls related to selections. (Read the ICCCM!)

Environment Issues: Selections

Symptoms of Incorrect Selection-Handling Code:

- Selection transfer fails (Press the Copy key and nothing happens)
- Selection transfer hangs the requester and/or the selection owner
- Programs that appear to be running correctly (or even quiescent!) crash when some other program tries to paste data
- X11 Server crashes when copying and pasting

Debugging Selection Code:

- Use xscope to monitor protocol traffic
- Watch for Time values of CurrentTime
- Watch for missing/malformed PropertyNotify and SelectionNotify events

Environment Issues: Selections

Example XScope Output (Highly Abridged!)

```
#
#  Here the user made a selection in an xterm, double clicking
#  on the word "Selection"
#
487.01: Client 11 -->    88 bytes
      ............REQUEST: ImageText8
                  drawable: DWB 01600014
                        gc: GXC 01600010
                         x: 2
                         y: 13
                    string: "<rapier 1>% This is the "
      ............REQUEST: ImageText8
                  drawable: DWB 01600014
                        gc: GXC 01600012
                         x: 194
                         y: 13
                    string: "Selection"


#
# The xterm becomes the owner of the PRIMARY selection; note
# how it does SetSelectionOwner followed by GetSelectionOwner
# to make sure it really has the selection
#
487.17: Client 11 -->    24 bytes
      ............REQUEST: SetSelectionOwner
                     owner: WIN 01600014
                 selection: <PRIMARY>
                      time: TIM a3e6af7e
      ............REQUEST: GetSelectionOwner
                 selection: <PRIMARY>
487.18:                                      32 bytes <-- X11 Server 11
                                       ..............REPLY: GetSelectionOwner
                                                    owner: WIN 01600014

#
# Another client has requested the PRIMARY selection, as a STRING,
# be placed in property 0x11d of window 01400014
#
494.74: Client 9 -->    24 bytes
      ............REQUEST: ConvertSelection
                 requester: WIN 01400014
                 selection: <PRIMARY>
                    target: <STRING>
                  property: ATM 0000011d
                      time: TIM a3e6c7cc
```

```
#
# The selection owner is placing the selection data into the
# right property on the right window -- make sure the property
# and window here match the ones in the Convert Selection
# request
#
494.80: Client 11 -->    96 bytes
      ............REQUEST: ChangeProperty
                      mode: Replace
                    window: WIN 01400014
                  property: ATM 0000011d
                      type: <STRING>
                    format: 08
                      data: "Selection"

#
# The selection owner is sending SelectionNotify to tell the
# requester that the transfer is done
#
      ............REQUEST: SendEvent
                  propagate: False
                destination: WIN 01400014
                 event-mask: 0
                      event:                ..............EVENT: SelectionNotify
                                                       time: TIM a3e6c7cc
                                                  requester: WIN 01400014
                                                  selection: <PRIMARY>
                                                     target: <STRING>
                                                   property: ATM 0000011d
494.92:                                              64 bytes <-- X11 Server 9
                                                ..............EVENT: PropertyNotify
                                                       window: WIN 01400014
                                                         atom: ATM 0000011d
                                                         time: TIM a3e6cdb2
                                                        state: NewValue


#
# The requester is doing a GetProperty to fetch the selection data
#

494.96: Client 9 -->    24 bytes
      ............REQUEST: GetProperty
                    delete: False
                    window: WIN 01400014
                  property: ATM 0000011d
                      type: AnyPropertyType
               long-offset: 00000000
495.00:                                              44 bytes <-- X11 Server 9
                                                ..............REPLY: GetProperty
                                                        format: 08
                                                          type: <STRING>
                                                   bytes-after: 00000000
                                                         value: "Selection"

#
# Finally, the requester deletes the property when done with it
#
495.01: Client 9 -->    80 bytes
      ............REQUEST: DeleteProperty
                    window: WIN 01400014
                  property: ATM 0000011d
```

Customization Issues

Resources, Actions, and Translations (RAT)

Customization presents opportunities:

- Resources allow configuration of fonts, colors, etc.
- Translations allow customization of behavior (key bindings, mouse operation, etc.)
- Some toolkits use resources to represent structure and behavior of applications: allows major changes without recompiling any code

Customization also presents problems:

- *Exposing this much of the implementation to naive users is a recipe for disaster!*

  (In effect, changes the number of possible execution environments from "many" to "infinitely many")

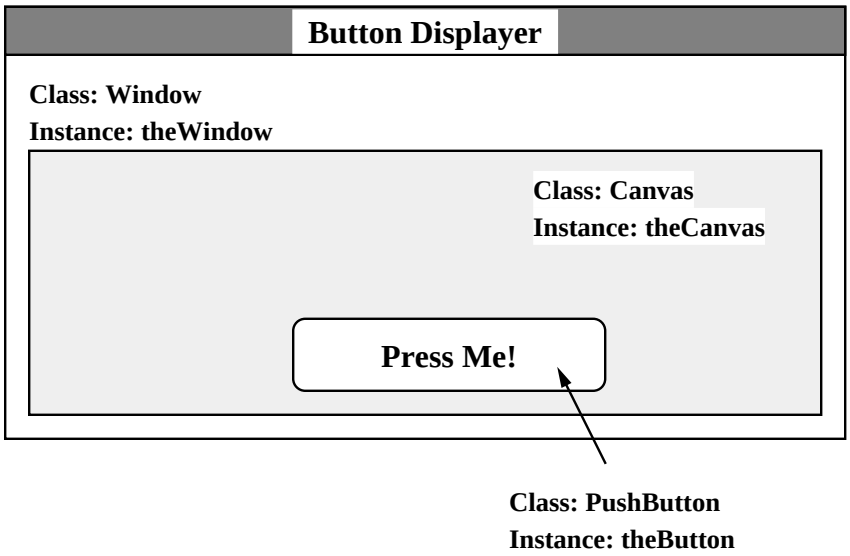
Two sources of confusion in customization:

- Resource naming hierarchy
- Resource file lookup hierarchy

Resource Naming Hierarchy

Resource specification hierarchy

- Resource specifications composed from application class and instance hierarchy for the object in question



  Resource specifications for the button:

  showbtn.theWindow.theCanvas.theButton.foreground
  showbtn.theWindow.theCanvas.theButton.background

- For each node, can use instance name or class name:

  ShowBtn.Window.Canvas.PushButton.Foreground
  ShowBtn.Window.Canvas.theButton.foreground
  showBtn.Window.theCanvas.Button.foreground

- Wildcards and binding rules make it hard to tell which resource specification is going to be applied when finding a resource value

Resource Naming Hierarchy

End users are often confused by resource names:

- The complete specification isn't always obvious
- Wildcards and binding rules make it hard to tell which of several specifications will apply when finding a resource's value

Examples (based on example from previous slide):

ShowBtn*Window*Background: SkyBlue
ShowBtn*PushButton*Background: Green

What color is the button?

ShowBtn*Window*Background: SkyBlue
ShowBtn*theButton*Background: Green

Now what color is the button?

***Window*Background: SkyBlue**
ShowBtn*theButton*Background: Green

Result of confusing rules: Users fumble around until it "looks right", but create unintended and potentially damaging side-effects.

Resource File Hierarchy

X11 Programs Seek Resources in Many Locations

- /usr/lib/X11/app-defaults/$LANG/Class
- /usr/lib/X11/app-defaults/Class
- $XAPPLRESDIR/Class
- xrdb property of the root window (usually set from $HOME/.Xdefaults)
- $XENVIRONMENT,
  or $HOME/.Xdefaults-hostname
- -xrm command line flags
- hard-coded program values

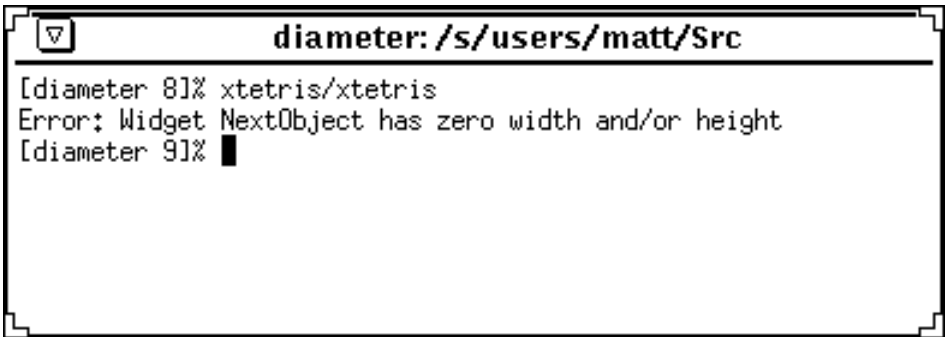
Plethora of locations makes it hard to tell where a resource value was set, or how to change it correctly.

End result: More fumbling around by users, more unintended and pathological side-effects

Detecting Resource Problems

Bad resource settings lead to several common kinds of misbehavior:

- Applications that refuse to run



  Generally implies missing app-defaults file

- Applications that run with wrong appearance

  Wrong colors, fonts, sizes generally implies a bad resource value being fetched because

  - Value specified with wrong format
  - Value specified in wrong file
  - .Xdefaults changed w/o running xrdb

- Applications that run with incorrect behavior

  - Keys/Mouse buttons don’t perform properly
  - Drag/select operations hang or crash

  Often implies that translations are set incorrectly, or are overridden somewhere

Detecting Resource Problems

Tools/Techniques to Examine Resource Settings

- Determine what resource specifications are being used
  - Toolkits with resource debugging support (e.g., OI) can help
  - xlswins to examine instance hierarchies
  - listres to examine Widget hierarchies
- Determine what resource values are being applied
  - appres: list resources used by application class

```
<rapier 127>% appres CodeCenter

CodeCenter*ProjectBrowser.RevisionControl:    centerline
CodeCenter*ProjectBrowser.Cmd1.Label:          Locked Files
CodeCenter*ProjectBrowser.Cmd1.Command:       listlocks ui
*OI*focusPolicy:          click_to_type
*OI*Model:                 openlook
*OI*Motif*background:   #e8b997
```

  - Use appres to look for "promiscuous" resources:

```
<rapier 127>% appres

*OI*Model:                openlook
*borderWidth:             1
```

  - Use "xrdb -query" to check the default resource database
- Scan environment and files to see where values come from

Resources and Defect Avoidance

Resources and translations allow the end user to break the application, so:

- Check for failure to load app-defaults files and print clear error messages.
- Consider hard-coding fallback resources
- Be compulsive when writing action functions

  Tip: Remember that the user can bind any action function to any translation. If action functions are context sensitive (start, drag, end), check for correct context. If action functions except to be bound to keyboard keys, check XEvent type before assuming XKeyEvent

  Many bugs happen because the user doesn't know what assumptions you've made inside the implementation.

Using Debuggers With X11 Programs

Special Problems Arise In Debuggers

- Program name may not be what you expect

  Tip: CodeCenter and ObjectCenter set your program name to "a.out" by default, which can interfere with resource fetch. Use the program_name option to set the name to whatever it would normally be.

- Timing considerations may change behavior

  Most likely to affect window manager interactions, selection behavior, bugs that involve race conditions when mapping windows.

- Debugging code that uses grabs is tricky!

  What happens if you hit a breakpoint in code that is called while the server is grabbed? Application has the server grabbed, but application isn't running because the debugger has control, and you cannot type at the debugger.... you're pretty much stuck.

Debuggers and Grabs

Basic Ways to Deal With Grabs

- Try iconifying application window from the WM
- Turn grabs off in the toolkit if possible (XView -fullscreendebug flag, for example)
- Reimplement XGrab* as no-ops
- Use debugger "tricks"

Debuggers and Grabs

Clever Debugger Trick:

1. Declare a global Display * variable

2. Set it to point at application's Display

3. Write a simple function:

```
void UngrabAll()
{
    XUngrabServer(globalDisplay);
    XUngrabKeyboard(globalDisplay, CurrentTime);
    XUngrabPointer(globalDisplay, CurrentTime);
    XFlush(globalDisplay);     /* Important!!! */
}
```

4. Tell debugger to "display UngrabAll()", so it will be called each time the debugger gains control.

This works for CodeCenter and ObjectCenter, also for dbx-style debuggers. For dbx-style debuggers, must link this code into the application; for CodeCenter and ObjectCenter, can do it all in the workspace.

Caveat:

> While in a grab, events can be channeled to a particular window. Disabling grabs turns this off, and may change application behavior!

Ideally, run on one screen and debug on another

Summing Up

Complexity comes from many sources

- Layers of implementation/abstraction
- Immaturity of underlying layers
- Confusing, low-level API's
- Asynchronous error reporting
- Interaction between processes
- Incorrect customization by users

Going from symptoms to causes can be difficult.

Almost always a process of elimination.

Strive to manage complexity when programming, so you don't have to wade through complexity when debugging.

Summing Up: Defect Avoidance

Programming to manage complexity

- Build firewalls and perform sanity checks when passing data between layers
- Understand the effects of all the hierarchies
    - Implementation (library) hierarchy
    - Widget class hierarchy
    - Window/instance hierarchy
    - Resource specification hierarchy
    - Resource file search hierarchy
- Develop intuition and experience about probable sources of bugs
- Test in as many environments as possible
    - Different servers
    - Different window managers
    - Different resource settings

Summing Up: Defect Isolation

Know Where to Start Looking

- XError messages

  Usually result from incorrect assumptions about server capabilities, drawable depths, visuals types, or colormaps. Sometimes due to "lazy" Window creation in toolkits.

- Display/Redraw problems

  Often result from bugs in processing Expose events.

- Color/Font problems

  Usually the result of bad resource values.

- Copy/Paste problems

  Usually the result of buggy Selection code

- Input/focus problems

  Often the result bad interactions with the window manager, or fights between the application and the toolkit

Defect Isolation Tools: A Bestiary

- xdpyinfo: show display/server characteristics
- xlsfonts: show available fonts
- xlswins: show all known windows
- xwininfo: show attributes of a window
- xprop: show properties of a window
- xshowcmap: show installed colormap entries
- listres: show Widget class hierarchy
- appres: show resource values likely to be used
- xrdb: show/modify default resource database
- xclipboard: check/monitor selection contents
- xscope: monitor X11 protocol traffic
- debuggers and programming environments
- Usenet (comp.windows.x, xpert mailing list)