Company Overview
Established in 1996, offices in New York, Boston, Chicago, Denver and Houston

Industries Served:
Automotive
Bio-medical
Chemical and Food Processing
Defense
Electronics/Semiconductor
Fuel Cells/Alternative Energy
Hydraulics
Laboratory Testing

Machine Tool
Material Handling
Medical Devices
Packaging
Pharmaceutical
Printing & Textiles
Certifications
Goal

Explore how to make LabVIEW UIs look:
- Comfortable
- Easy to use
- Intuitive
- Familiar
- Attractive
Alternate Title: “Click Better! The LabVIEW UI Story”
Modern User Interface Design
Typical LabVIEW User Interface
Typical LabVIEW User Interface
Agenda

- Definitions and principles
- Better controls/indicators
- Typedefs – strictly speaking
- Screen navigation
- Dialogs – Dia-dos and dia-don’ts
- Resizable interfaces
- Technical considerations
- Style tips and tricks
Definitions

User Interface = the things you click on or look at to work with your software

User Experience = the rage you feel when your software is hard to use
UI Principles

- Don’t overcrowd screens
- Keep fonts, colors, and styles consistent
- Don’t use colors that burn the user’s face off
- No gradients
- Keep text readable
- Align controls/indicators

Think about it ahead of time... don’t wing it. Plan it like you plan the program’s business logic.
Better Controls/Indicators
Classic is the New Modern

Customizing classic controls makes them look awesome:

[Diagram showing two sets of 'Cycle Stop' controls before and after customization]
Change background/foreground colors

Convoluted process to customize the control:
1. Place a classic button on the front panel
2. Paint the control using the paint brush tool
3. Using the paintbrush tool, right-click near the border of the control, then press SPACEBAR
4. Click the transparent “T” in the upper right corner
5. Repeat for the other Boolean state
Add custom graphics

1. Right click on the button and select Advanced ➔ Customize.

2. Go to Edit ➔ Import Picture from Clipboard and select and image to import.

3. Right click the button and select import picture from clipboard (True, False, Decal).

4. To reposition the graphic, change to Customize Mode (Ctrl – M)
An Aside on System Buttons

- System buttons are the only style that supports hovering
- Colors can be changed by customizing the button as shown previously.
Customizing a System Button

1. Make a new control and add a system button
2. Click the wrench button to customize
3. Right-click the button ➔ Picture item

1: FALSE, not hovering  2: TRUE, not hovering  3: FALSE, button down
4: TRUE, button down  5: FALSE, hovering  6: TRUE, hovering

Menu Bars

Create a clean, Windows-style UI using menu bars

Caveat: not appropriate for a touchscreen
How to add Static Menu Bars

1. Go to Edit ➔ Run Time Menu
2. Change drop down to “Custom”
3. Add your own structure and options for the menu.
4. Save .rtm file in the same directory as the VI.
5. Handle events from the menu items in an event structure.
How to add Dynamic Menu Bars

1. Obtain a reference to the vi’s menu bar.

2. Use “Insert Menu Items” to dynamically add menu items and sub menu items.
Menu Bar Demo
Typedefs – Strictly Speaking
Types of Controls

- Control
- Type Def
- Strict Type Def

All are saved as *.ctl files
Control

- Good for pretty buttons that you want to be able to reuse
- Each one is its own unique snowflake
  - Can be customized independently
  - Only the data type is important
- Won’t update globally
Type Def

- Only the data type is important
- Each instance does update when the data type changes
- The visible properties do not update
Strict Type Def

- Data type and visible properties are important
- All instances change when the typedef changes, including graphical properties
- Every instance has to look the same
- Visual properties cannot be changed programmatically
  - Value can be set
  - Cannot change colors, etc.
Strict Typedef Limitations

- Strict typedefs can be very limiting
- May cause issues for RT systems, since it has UI info embedded

<table>
<thead>
<tr>
<th>Typedef</th>
<th>Strict Typedef</th>
</tr>
</thead>
</table>

- `Visible`: Appears when selected, or not when deselected.
- `Visible`: Also appears when selected, or not when deselected.
- `Enabled`: Appears when selected, or not when deselected.
- `Enabled`: Also appears when selected, or not when deselected.

<table>
<thead>
<tr>
<th>Appearance</th>
<th>Documentation</th>
<th>Data Binding</th>
<th>Key Navigation</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Caption</td>
<td>Size</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visible</td>
<td>Visible</td>
<td>235 Height</td>
<td>259 Width</td>
<td></td>
</tr>
<tr>
<td>Enabled</td>
<td>Enabled</td>
<td>406 Height</td>
<td>648 Width</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Appearance</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled</td>
<td>Visible</td>
</tr>
<tr>
<td>Disabled</td>
<td>Visible</td>
</tr>
<tr>
<td>Disabled &amp; grayed</td>
<td>Visible</td>
</tr>
</tbody>
</table>
Propagating Typedef Changes

Say you have several instances of a typedef and your code changes something programmatically. You can still propagate changes to all instances:

1. Change it to a strict typedef (your VI will be broken)
2. Make your changes to the strict typedef
3. Apply changes, so all instances get the new look
4. Change back to a typedef (VI should be OK)
Multi-Screen Applications
Multi-Screen (View) Challenges

- How will the user navigate?
  - Provide clear, intuitive direction
- How do you keep the UI responsive?
- How do you lay out the screens?
- How do you manage the increased code complexity?
Interface Specification Outline

- Pre-Run
  - Live Display
  - Post-Run
- Data Viewer
- System Settings
  - Hardware Configuration
  - Channel Configuration
- Users
Option 1: Tab Controls

Pros:
- Screens can be separated by tab
- Single supporting thread can assist with navigation
- Tabs have useful sizing/snapping features

Cons:
- Does not scale well beyond 5 (or so) screens
- Supporting code can be very monolithic, results in huge complex event structures, hard to add new features
Example: Using Tabs
Option 2: Subpanels

Pros:
- Allow for more modular code, loading individual VIs into supervisory interface
- Allows for more feature rich screens, without monolithic thread support

Cons:
- Requires careful application architecture to implement
- Must be very aware of VI state and reentrancy settings, can leave threads running in background inadvertently
- Requires well defined communication design pattern to allow effective sequencing of screens.
Example, Using Subpanels

Add Subpanel to Front Panel
Example, Using Subpanels

Subpanel property node is added to block diagram automatically:
Example, Using Subpanels

Simply connect a reference to the VI you want in the subpanel.
Example, Using Subpanels

Add a mechanism for switching between different VIs
Subpanel Demo
Subpanel Recommendations

- Use a tab control that is set to the same size as the subpanel
  - Helps lay out the subpaneled VI
- Programmatically align the front panel of the subpaneled VI with the origin
Intermission
Dialogs – Dia-dos and Dia-don’ts
Dialog Boxes

- Used to:
  - Communicate important information to the user
  - Get the user’s attention
  - Collect data from the user
This template is for the Producer/Consumer design pattern.

This loop is the producer loop.

Dialog Message

Source

status

Releasing the queue stops the consumer loop(s).

This loop is a consumer loop.

Error

Return true in any case that should stop the loop.
Floating vs. Modal

- Modal dialogs will...
  - Stay on top of all other windows
  - Be the only window with which the user can interact
- Floating dialogs will...
  - Stay on top of all other non-floating windows
  - Allow the user to interact with all visible windows
- Context help window is floating
Dia-dos

- Make sure you need a dialog
- Put all blocking dialogs into a UI loop
- Provide a mechanism for data from dialog to be used
- Think about how it communicates with the rest of the application
- Use a reasonable timeout if the response is time-sensitive
- Make your own modal dialogs to match application style
- Output a Boolean to indicate whether or not the user canceled
Dia-don’ts

- Block execution of your control or data acquisition loop
- Create modal dialogs that won’t close
- Create dialogs that write to global variables
- Make a one-button dialog without considering other ways to display the same status
- Use menus in dialogs
Window Behaviors

- Window Behavior
  - Default
  - Floating
  - Modal

- Execution mode
  - Asynchronous
  - Synchronous
Resizing Interfaces
Committing to a Monitor Size Forever

- Ideally, avoid resizing interfaces
- Often, it’s simplest to figure out what the monitor resolution is, then size things appropriately

This is a man committing to his monitor. FOREVER.
If you absolutely MUST allow resizing, try panes

- Use splitters to create a hierarchy of panes
- Right-click splitters to set how the panes move
- "Splitter Sticks Right" ➔ The splitter moves and sticks with the pane on the right
- Set controls to be fit to their pane or to scale with their pane
- Set a minimum VI size
Pane Demo
Front Panel Layout Tool

- Available from the NI Community/VIPM
- Save the front panel state as-is and associates it with a keyword
- Define several layouts for different monitor sizes
- At runtime, decide which configuration to use
Technical Considerations
UI Loop

- What is the UI loop?
  - Common architectural component
  - Event structure in a while loop

- Why should I have one?
  - This is the loop that SHOULD wait on UI events, NOT your main processing loop
  - Don’t do any other processing here... keep the UI responsive
  - Main processing loops should continue executing independent of user interaction
Event Structures

- Has multiple cases to handle direct interactions from the user interface
- Only one event case can execute for a single event
- A single event case can handle multiple events
Fonts

- Default system fonts can and will change between operating systems or versions of Windows
- Be wary of the default Windows zoom level (125%)
- Explicitly set the fonts you use on the front panel, instead of relying on the “System”, “Dialog” or “Application” fonts
- It’s easy to use third-party fonts, BUT it’s hard to include in an installer
Be Careful with Fonts
When running an EXE, no one wants to see the LabVIEW “Run” or “Abort” buttons. Hide them!
Setting Toolbars Programmatically

- Set it so the toolbars are visible in the development environment
- Hide them if the App.Kind property is executable
Closing the Application

With an EXE, close labview at the end of the program so the VI doesn’t just sit there.
XControls

- XControls have dynamic run-time and edit-time behavior that is defined by VIs that run in the background
- That sounds great...but is it?
  - Seems cool, really brittle
  - Hard to debug
  - Need a really good reason to use it
Style Tips and Tricks
Color Palette Basics

- Easy inspiration online
- Think clean design
- Use readable text colors with good contrast

Key colors:
Red = bad/stop
Green = OK/continue
Align/Distribute Tools

Please use these to organize, space and align objects.

Sometimes have to move or hide the label before trying to align items for a clean look.
Control Parts Window

- Right-click a control/indicator ➔ Advanced ➔ Customize
- Select Window ➔ Show Parts Window
- Cycle through each customizable part of the control
- Enjoy the terrible UI on a useful UI tool!
Control-M
- Allows you to preview how a VI will look when it runs
- Removes overlay shadows
- Removes grid lines
- Mouse and keyboard navigation work as if it’s running
User Feedback

- It’s comforting to the user to know that something is happening
- Using the cursor set/unset busy VIs can help
- Have a plan to unset in the event of an error or other problem!
Touchscreen Design

- Touchscreens present unique challenges
  - Can be slow or unresponsive (especially resistive types)
  - Less dexterity than a mouse
Touchscreen Design

But if they really do want one...

- Make buttons large enough for fat fingers
- Plan for user entry popups
- Remember: no tactile feedback

NO. Just don’t...
Working with a graphic designer?

- Be clear about what is realistic in LabVIEW
  - Available controls/indicators
  - Available shapes and animations
- They should focus on user needs and clarity
- They will be using their professional design tools (Illustrator, Inkscape, etc...), not LabVIEW
  - Cannot use vector graphics
  - PNGs with transparent backgrounds
Additional Resources

- NI Community UI Interest Group at https://decibel.ni.com/content/groups/ui
- Creating Quality UIs from NI Developer Days at https://decibel.ni.com/content/groups/ui/blog/2010/04/29/creating-quality-uis-with-ni-labview--developer-days-2010-presentation