OAK RIDGE National Laboratory

Display Builder

Kay Kasemir

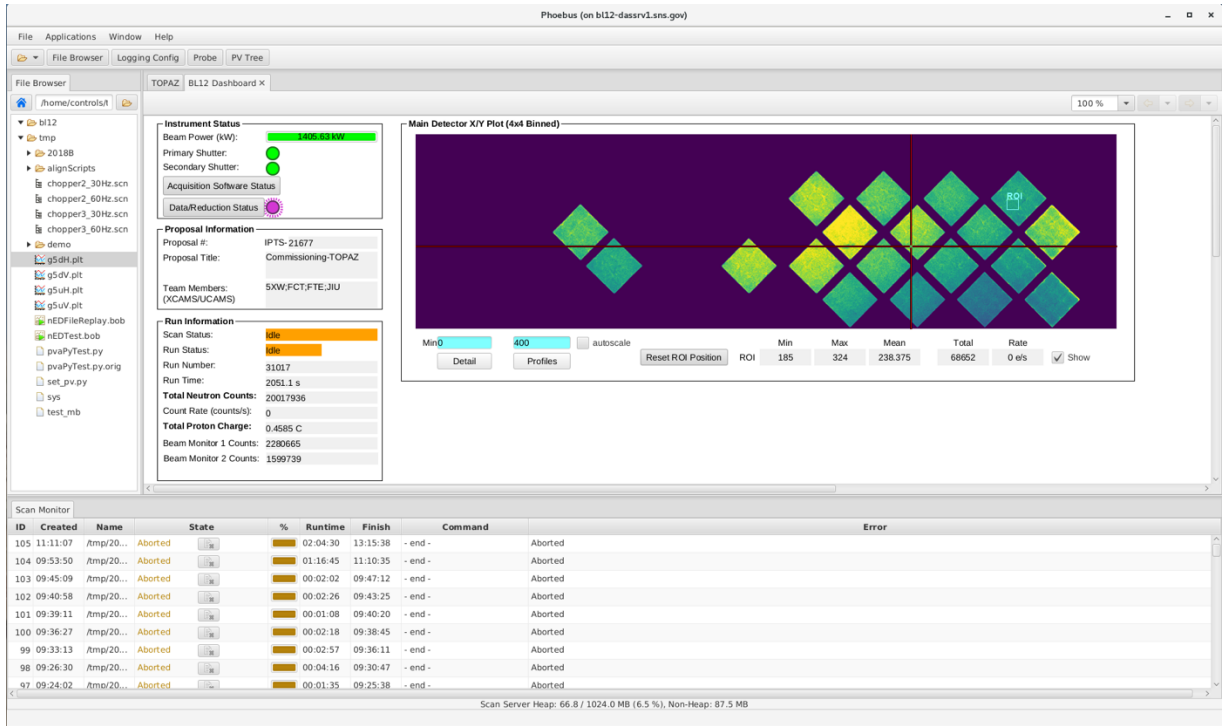
July 2026

ORNL is managed by UT-Battelle, LLC for the US Department of Energy

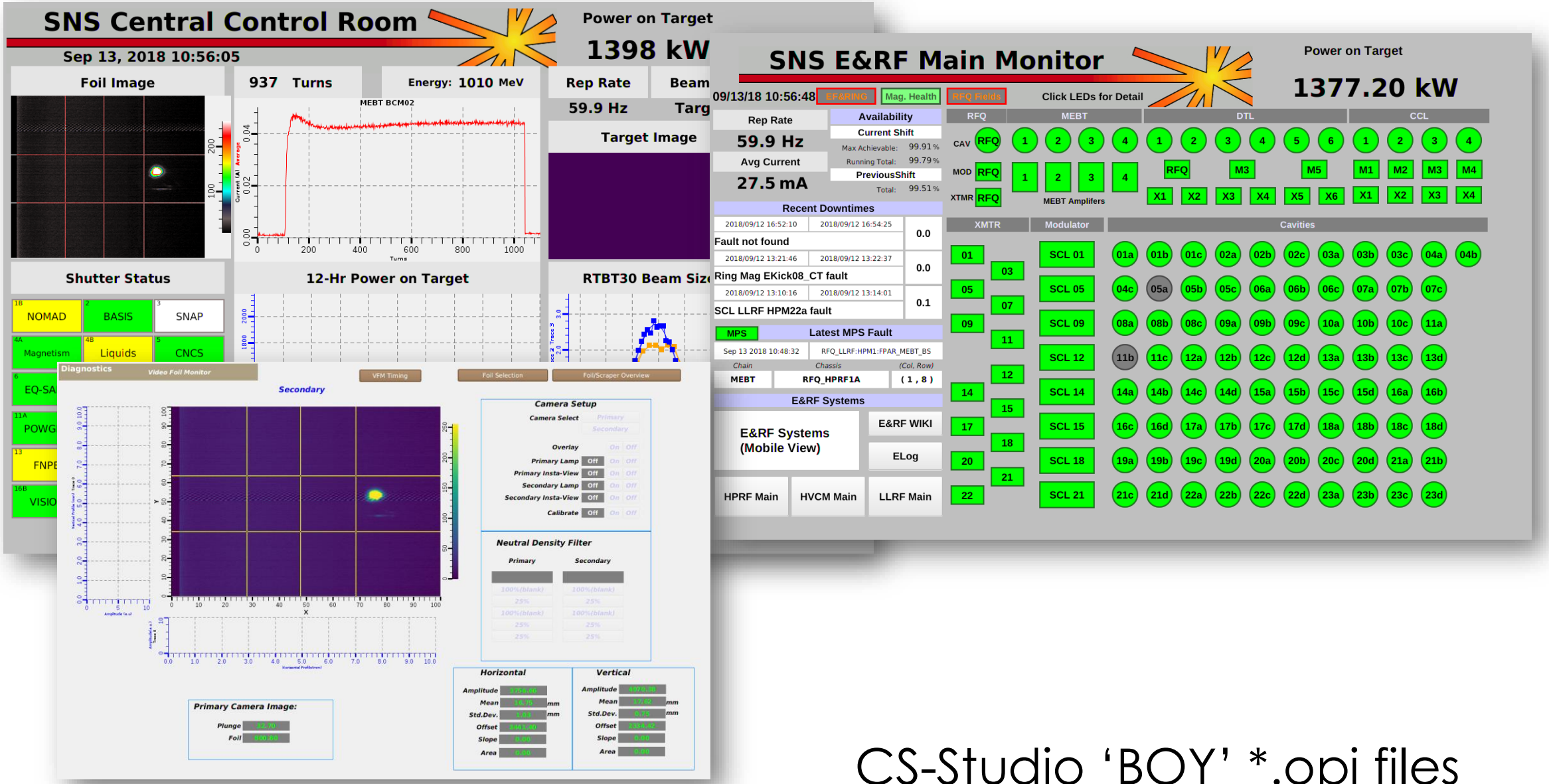
U.S. DEPARTMENT OF ENERGY

Display Builder

- Operator Interface Editor and Runtime
- Builds on ideas from EPICS edd/dm, medm, edm (~2000), ..
- Very compatible with CS-Studio 'BOY' from ~2009
- Started ~2015 in CS-Studio/Eclipse, now in CS-Studio/Phoebus
- Since 2019 also available as Web Runtime

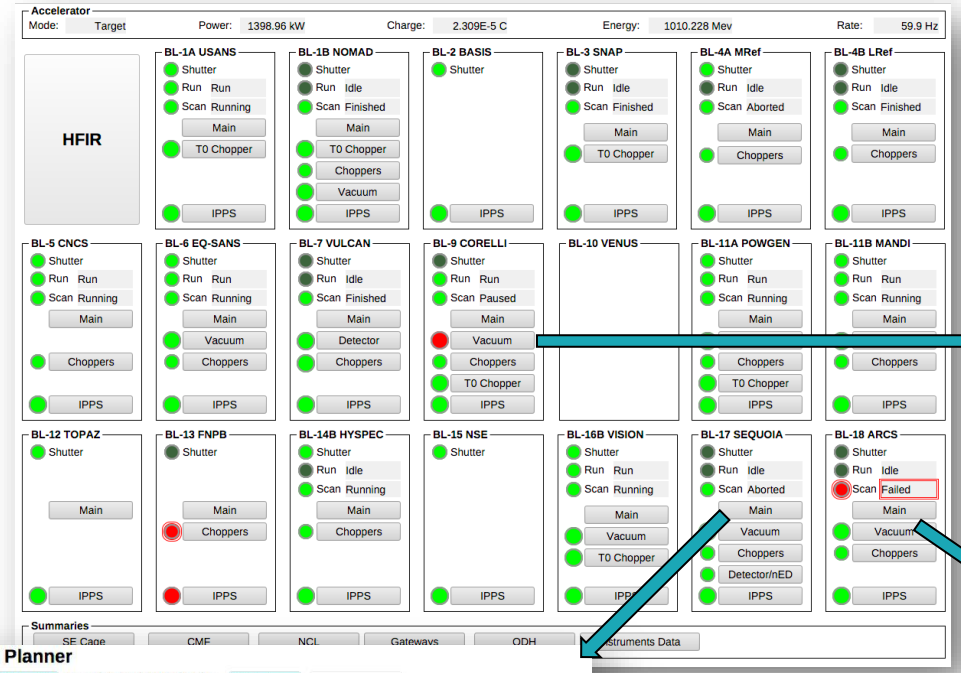


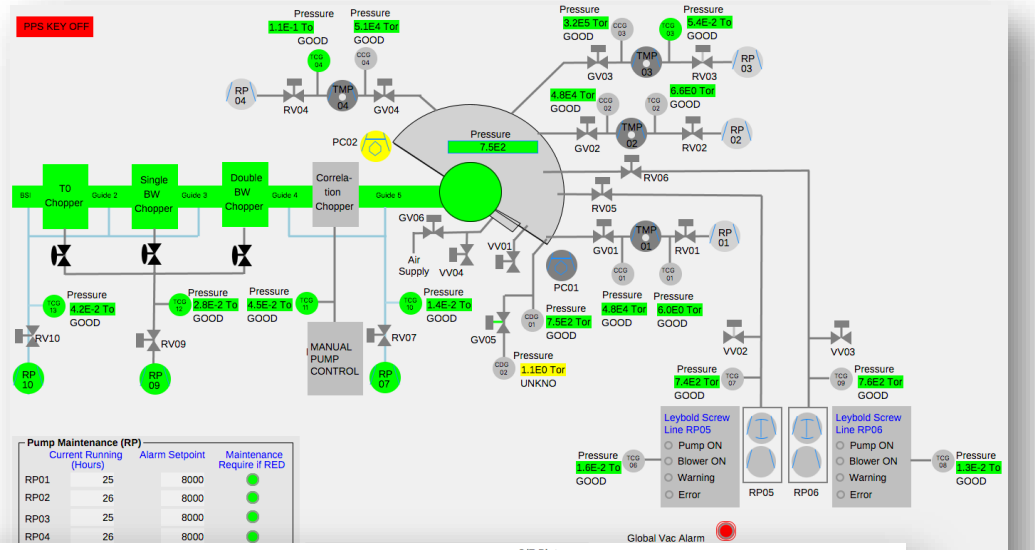
Examples: SNS Accelerator

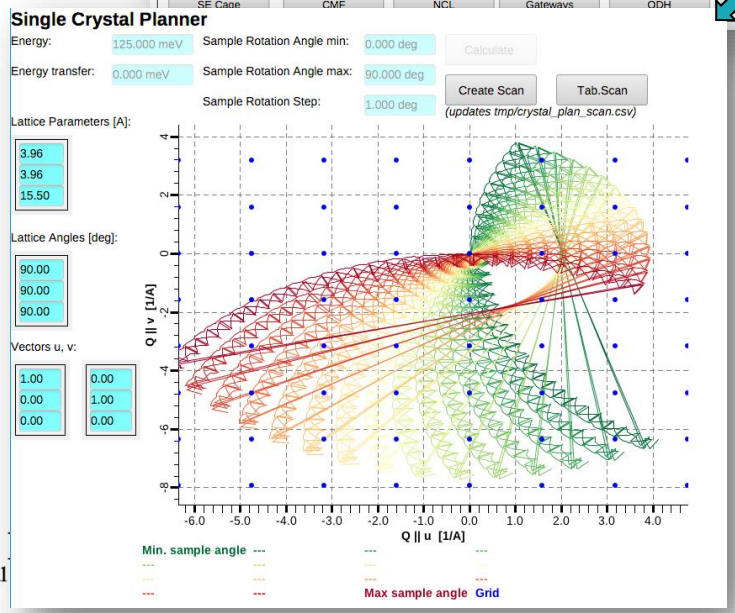


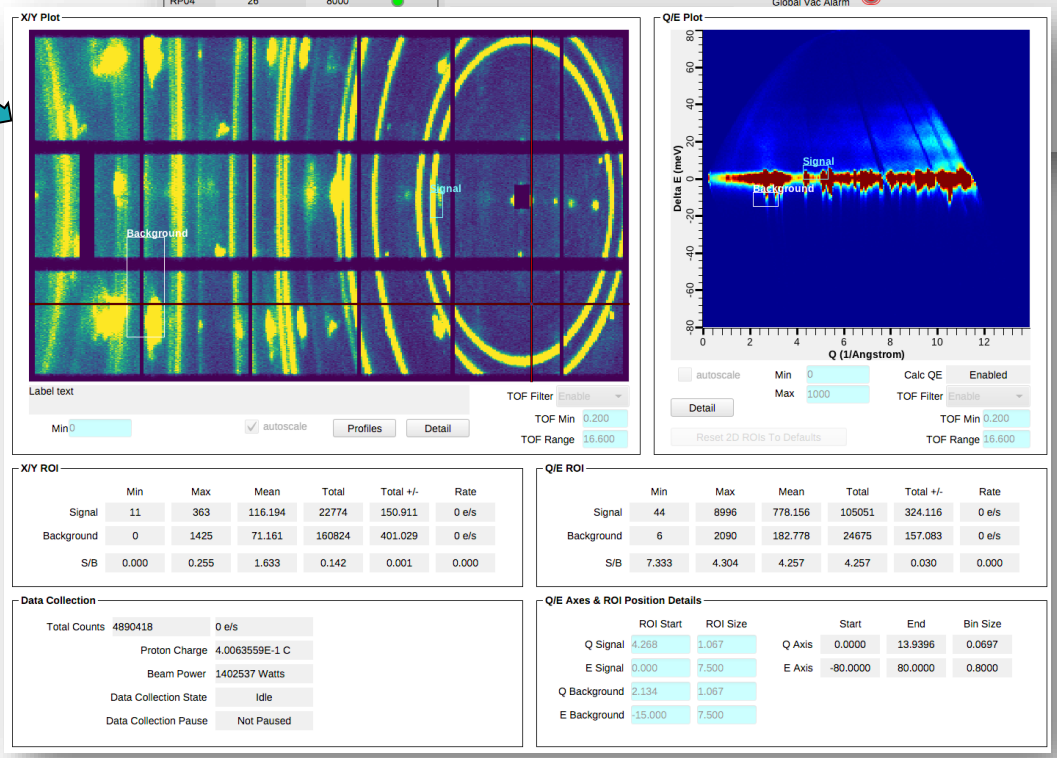
CS-Studio 'BOY' *.opi files

Examples: SNS Beam Lines







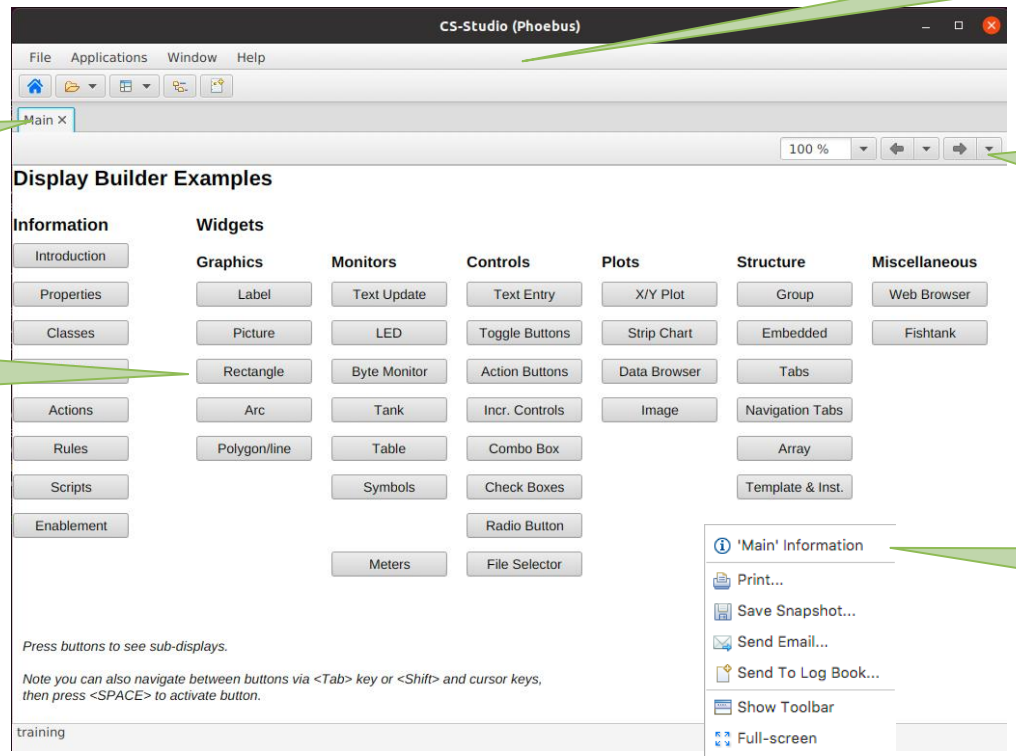


Browse the Examples

- Start CSS/Phoebus
- Your setup might have a menu entry
  - File, Top Resources, Examples
- If not, or if you'd like to inspect and edit the examples
  - Applications, Display, Examples, Install Example Displays

Main Application Toolbar
Menu Window, Show Toolbar

Tab
Hover mouse,
open Context Menu,
Close



Display Runtime Toolbar
Context Menu Window,
Show / Hide Toolbar

Navigate back/forward
also via Alt-Left, Alt-Right cursor keys

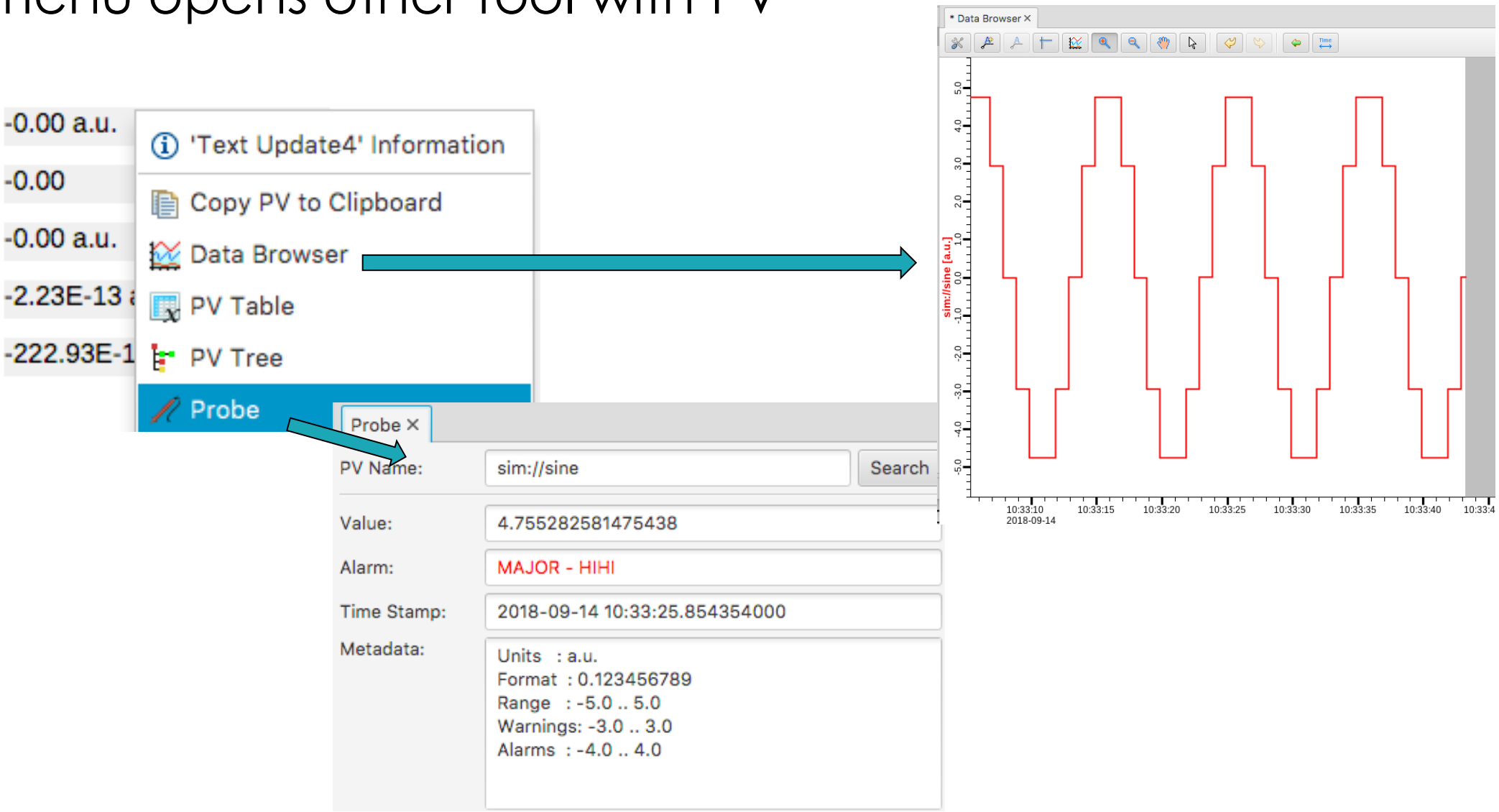
Zoom
to view large control room displays on office computer

Example Display
Push any of the buttons

Context Menu
Details change with widget
on which menu was invoked

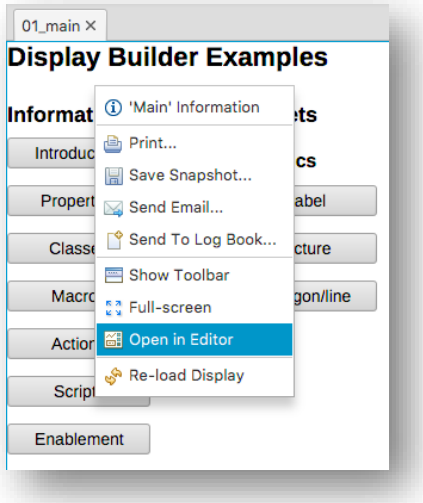
Send PV to other Tools

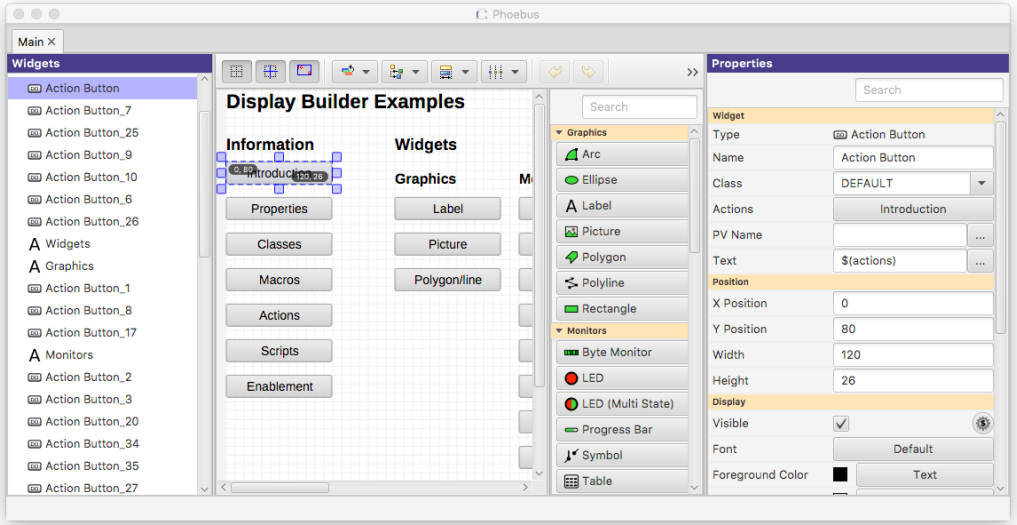
Context menu opens other tool with PV



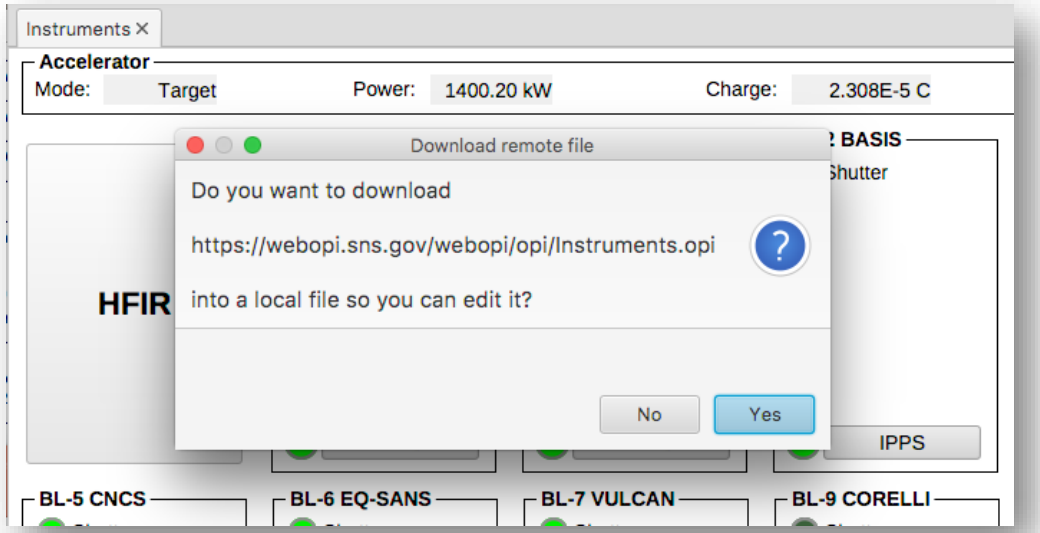
Open Existing Display In Editor

- Context menu can open any display in Editor





- Downloads remote files



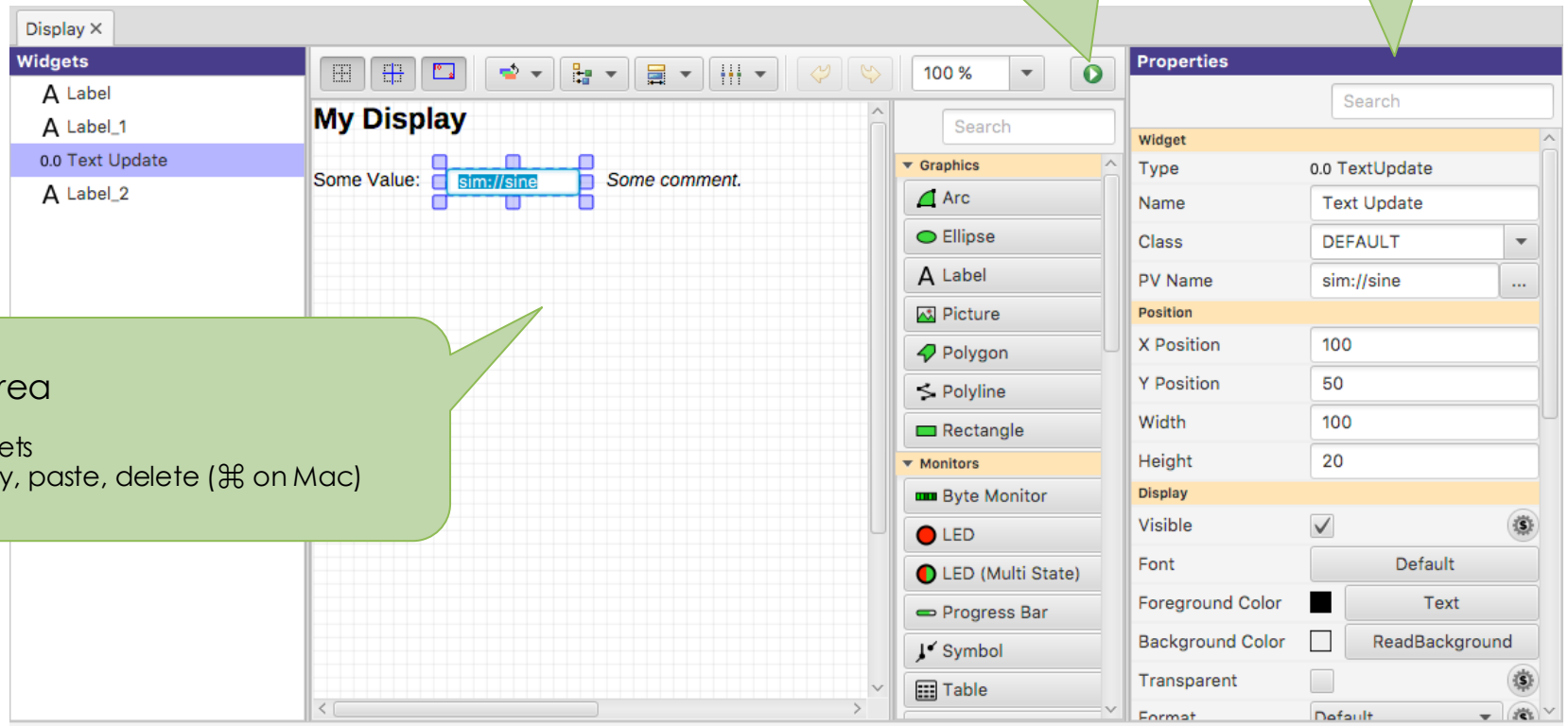
Create New Display

Menu Applications, Display, New Display

- Enter a name with .bob file extension

Save & Execute the Display

Property Panel
Edit properties of selected widgets



Main Editor Area
Select Widgets
Move, resize widgets
Ctrl-C, V, X to copy, paste, delete (⌘ on Mac)

Editing a Display

Selecting Widgets

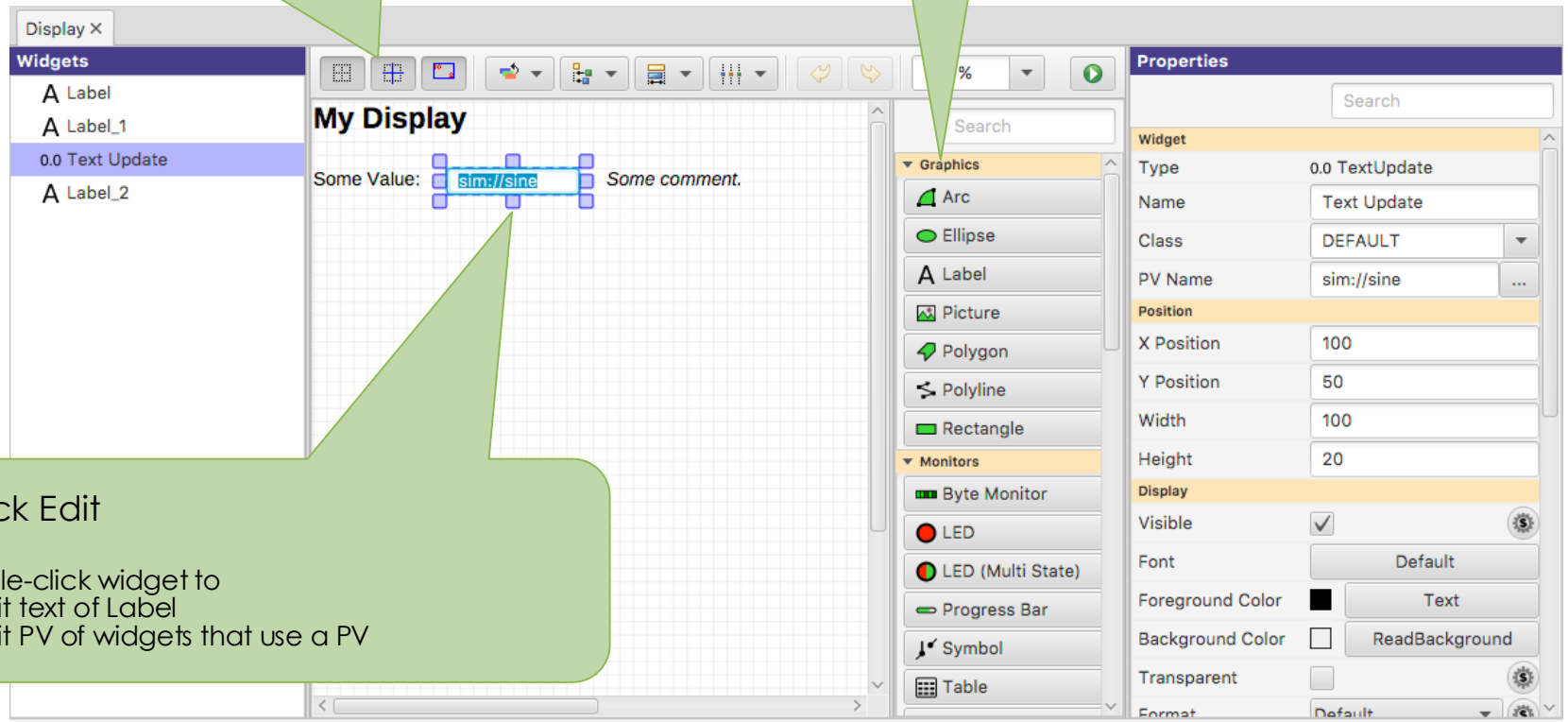
a) Click single widget
b) Ctrl-click to add widget (⌘ on Mac)
c) Drag 'rubberband' around widgets
d) Click or Ctrl/ ⌘ click in widget list

Widget Palette

Drag widget into editor

- or -

1) Select Widget Type
2) Draw rectangular area in display

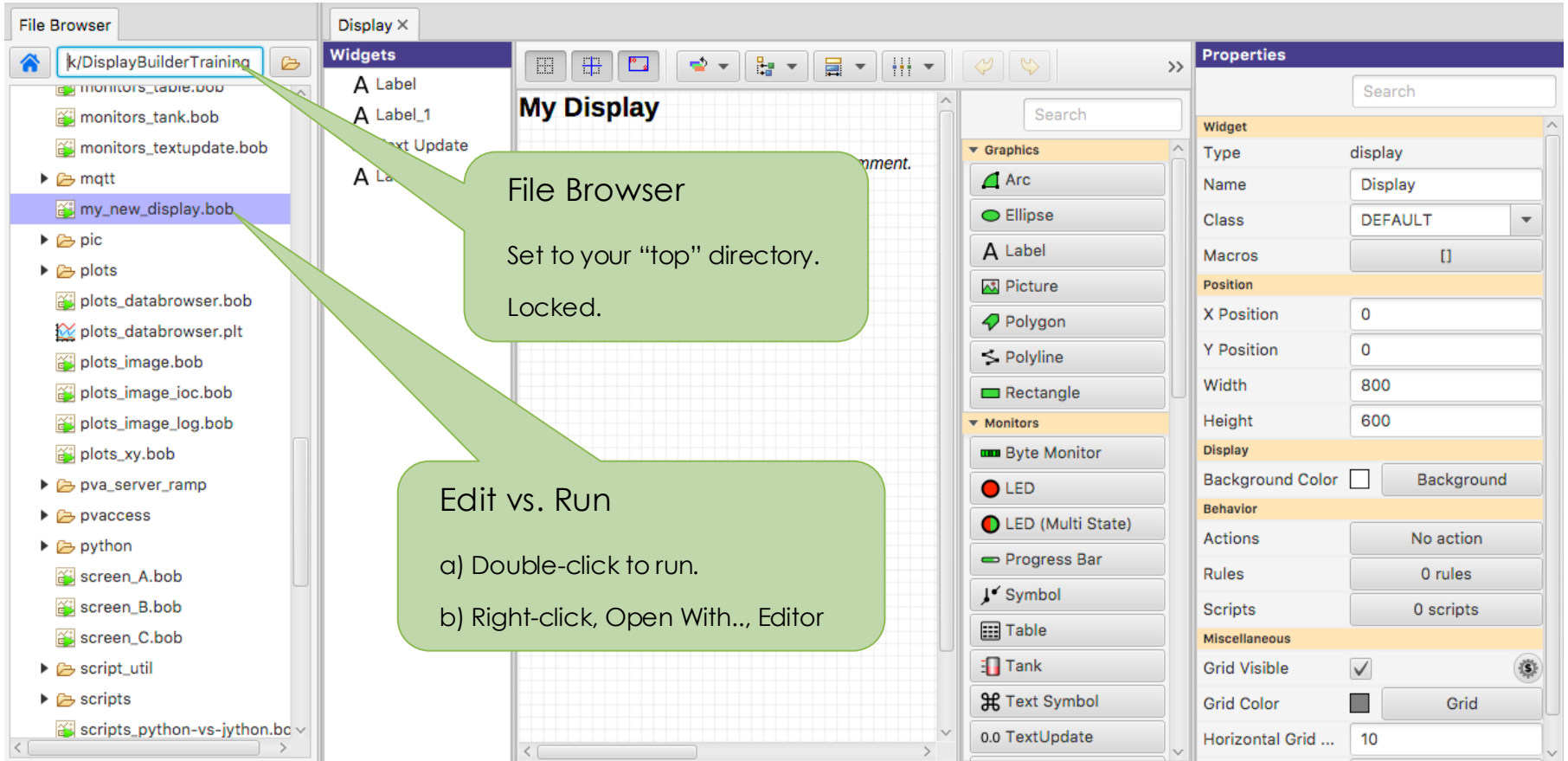


Quick Edit

Double-click widget to
a) Edit text of Label
b) Edit PV of widgets that use a PV

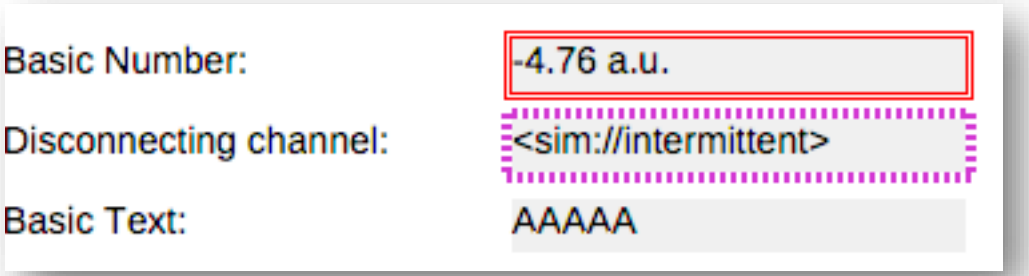
Suggested Setup for Editing

- Pick a top directory, for example where you installed the example files
- Open Applications, Utility, File Browser
  - Set it to your top directory
  - On file browser tab, open context menu, “Split Horizontally”, then “Lock Pane”
- Menu Window, Save Layout As..
  - “Editing”
- Menu Applications, Display, New Display
  - Create new file in your top directory



Keep It Simple

1. Add a Widget
2. Enter Label's Text or Widget's PV Name
3. Done



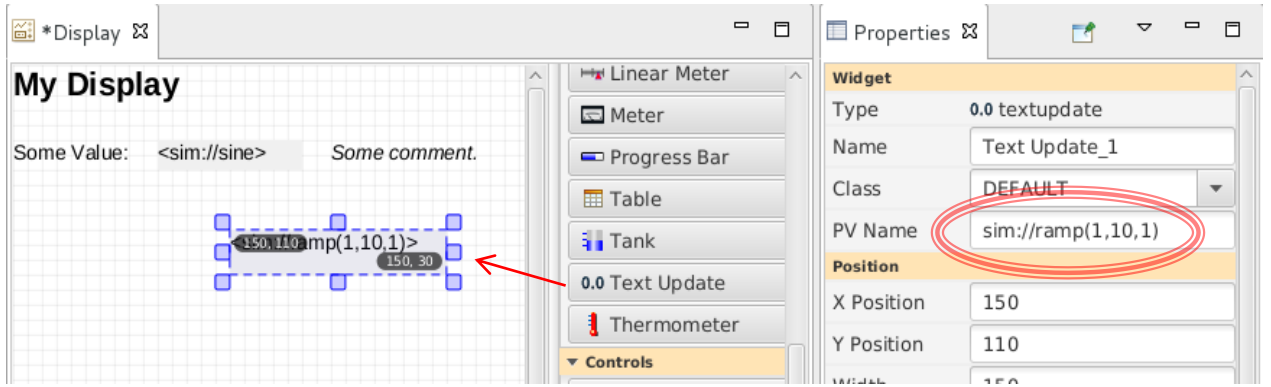
At Runtime, widget will

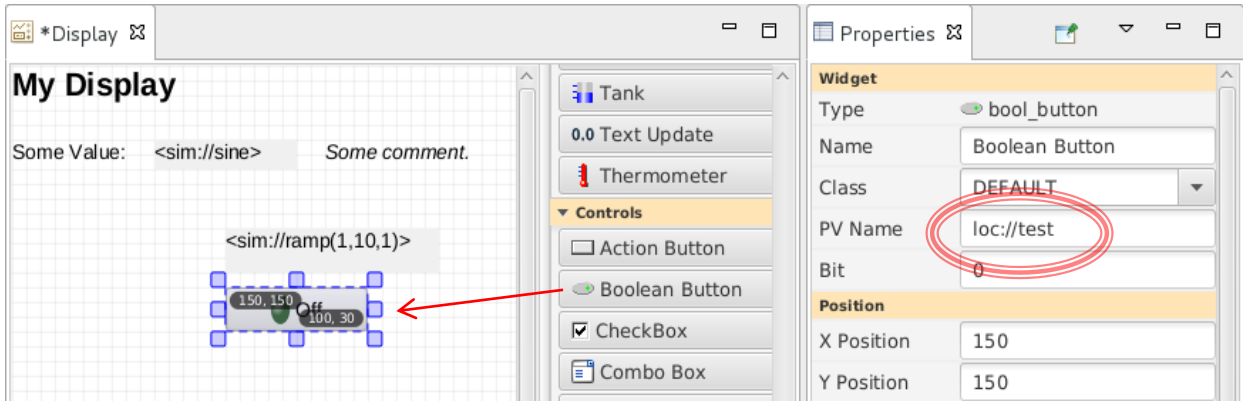
- ✓ Show PV's value, formatted, with units
- ✓ Indicate alarm, disconnect
- ✓ Show tool-tip with PV name and value
- ✓ Combo options read from Enum PV, slider range from numeric PV
- ✓ Disabled when 'control' widget has no PV write access

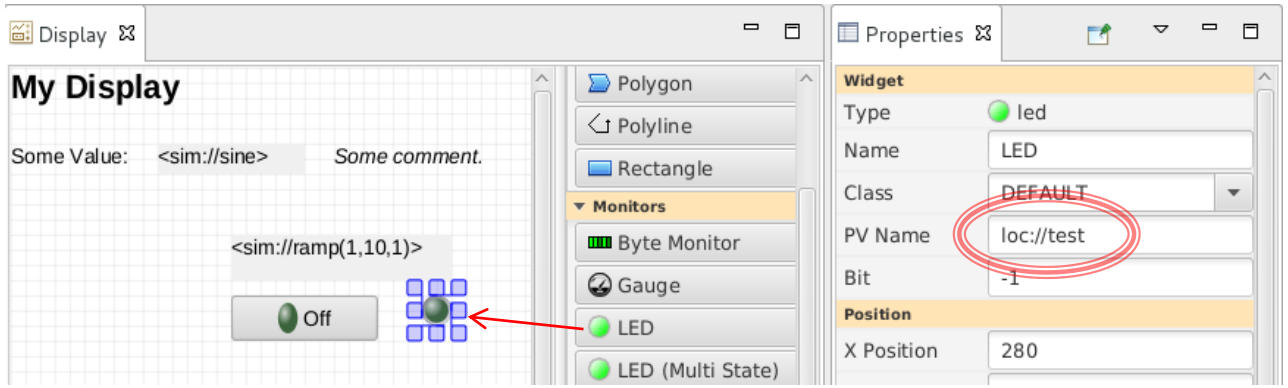
Extend the First Display

- Drag a "Text Update" from the palette
  - Enter PV name "sim://ramp(1, 10, 1)". Note PV name auto-completion popup.

- Add "Boolean Button"
  - PV name "loc://test"
- Add "LED"
  - PV name "loc://test". Note name in PV History.
- Execute the display
  - Toolbar Button or Context Menu





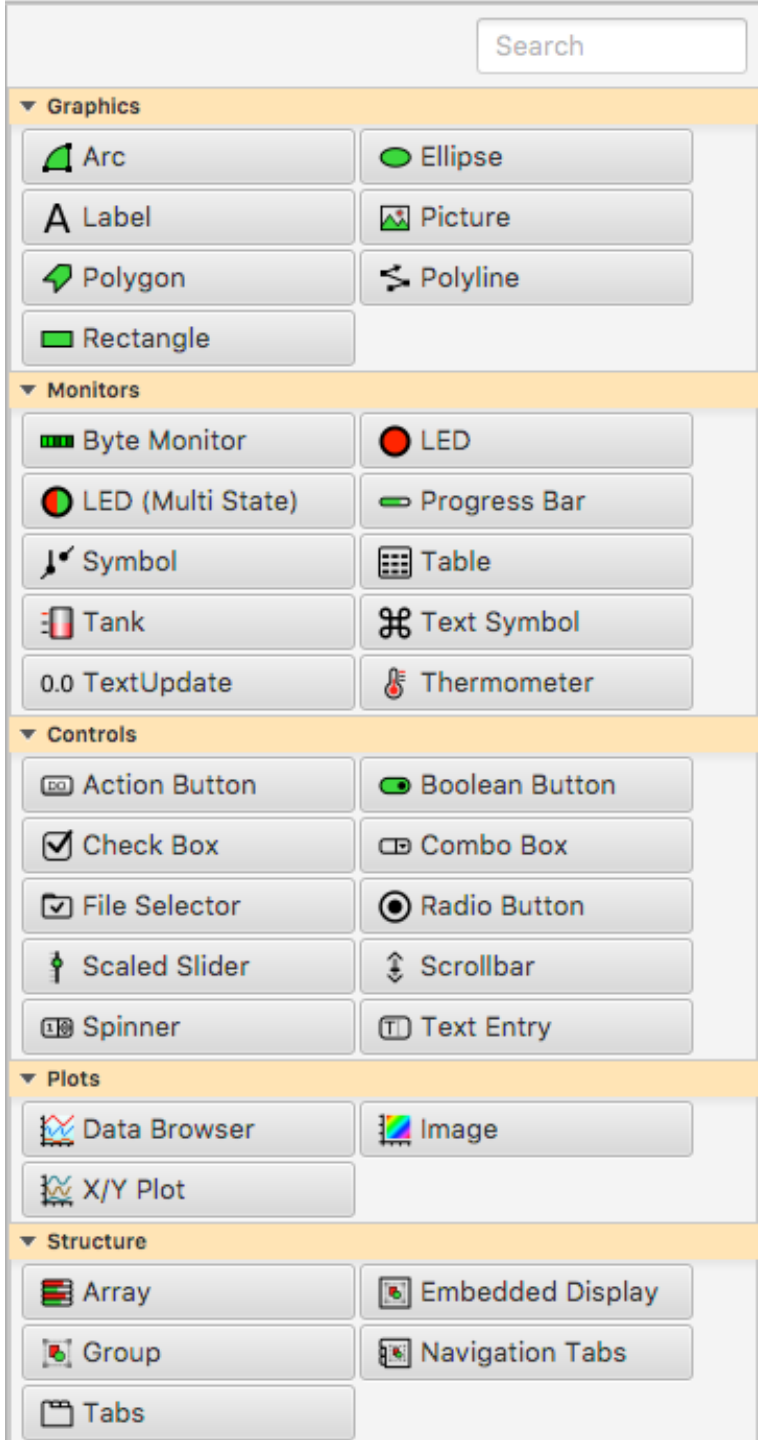


PV Names

- `ca://some_pv_name`
  - EPICS Channel Access PV
- `some_pv_name`
  - Typically same, since “ca://” is the default
  - Eventually, “pva://” may become the default
- `sim://sine`
  - Simulated PV. See auto-completion hints
- `loc://x(4)`
  - Local PV. See auto-completion hints
- `pva://x`
  - EPICS PV Access

Widget Palette

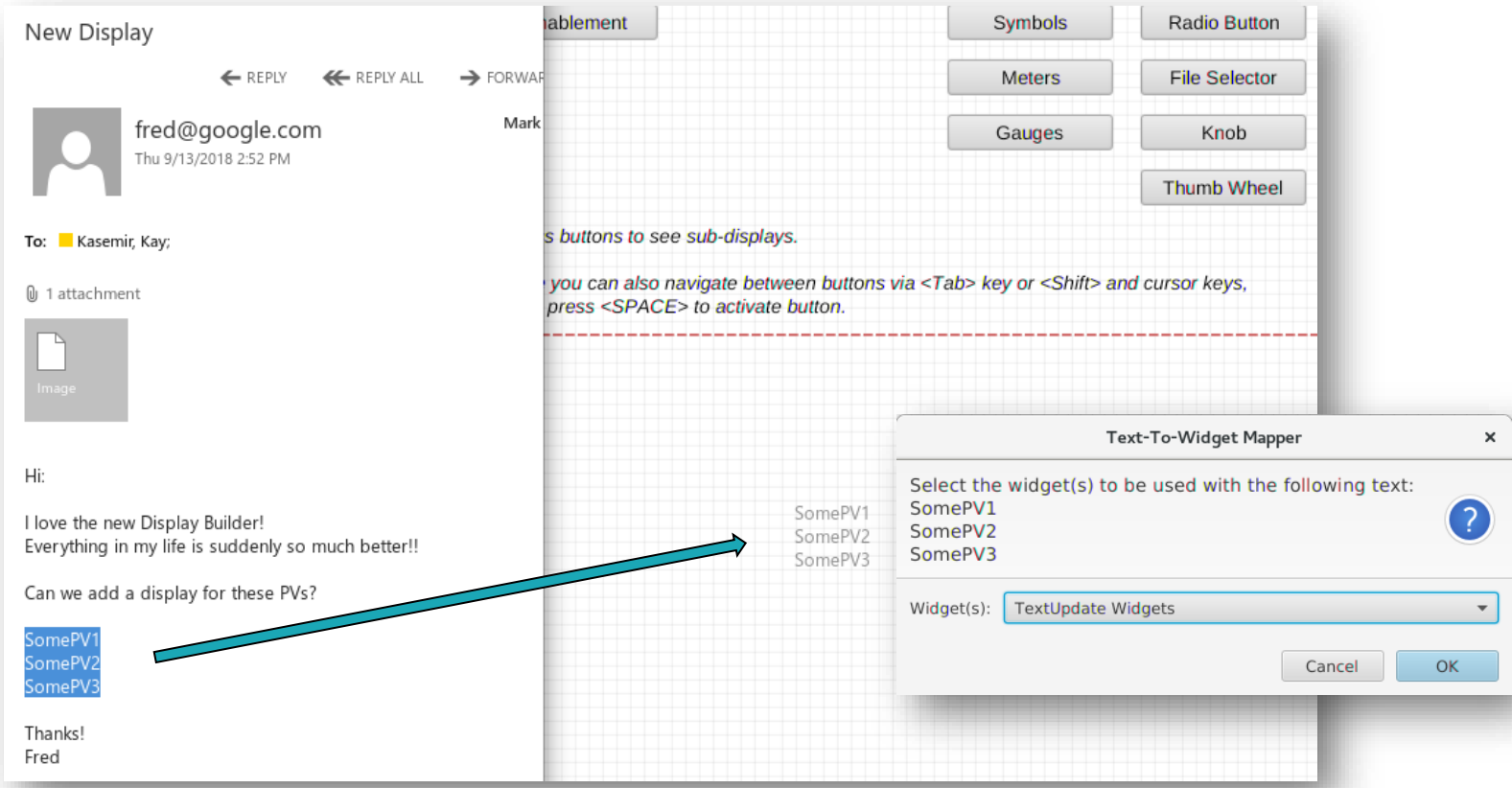
- Shows all available widgets
  - Enter name for "Search"
  - Hover mouse for description
  - Drag *-or-* Select & Rubberband
- Categories
  - Graphics show static label, picture, ..
  - Monitors update based on reading a PV
  - Controls read a PV and can write to the PV
  - Plots tend to read from one or more (waveform) PVs
  - Structures group widgets, embed sub-displays



Create Widgets via Drag/Drop from other Apps

Email with list of PVs?

- Drag that text into Display Editor
- Select widget type



Supported:

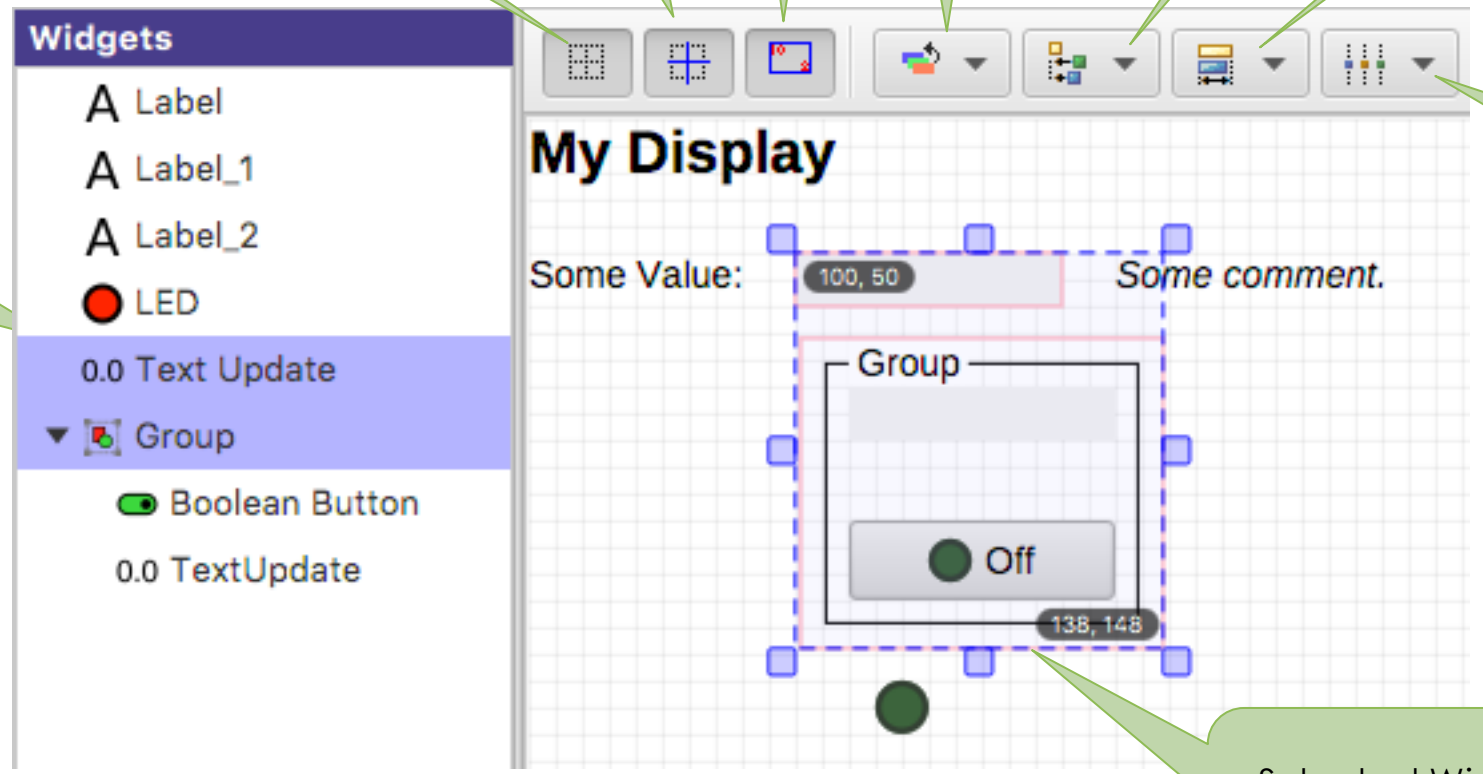
Text → Label

Text → PV Widget

Image File → Picture Widget

*.bob File → Embedded Display Widget

Manipulating Widgets



Show Coordinates

Snap to Grid

.. Other Widgets

Order Front/Back

Align

Size

Distribute

Widget List

Select widgets

Rename Widgets

View/change their order.

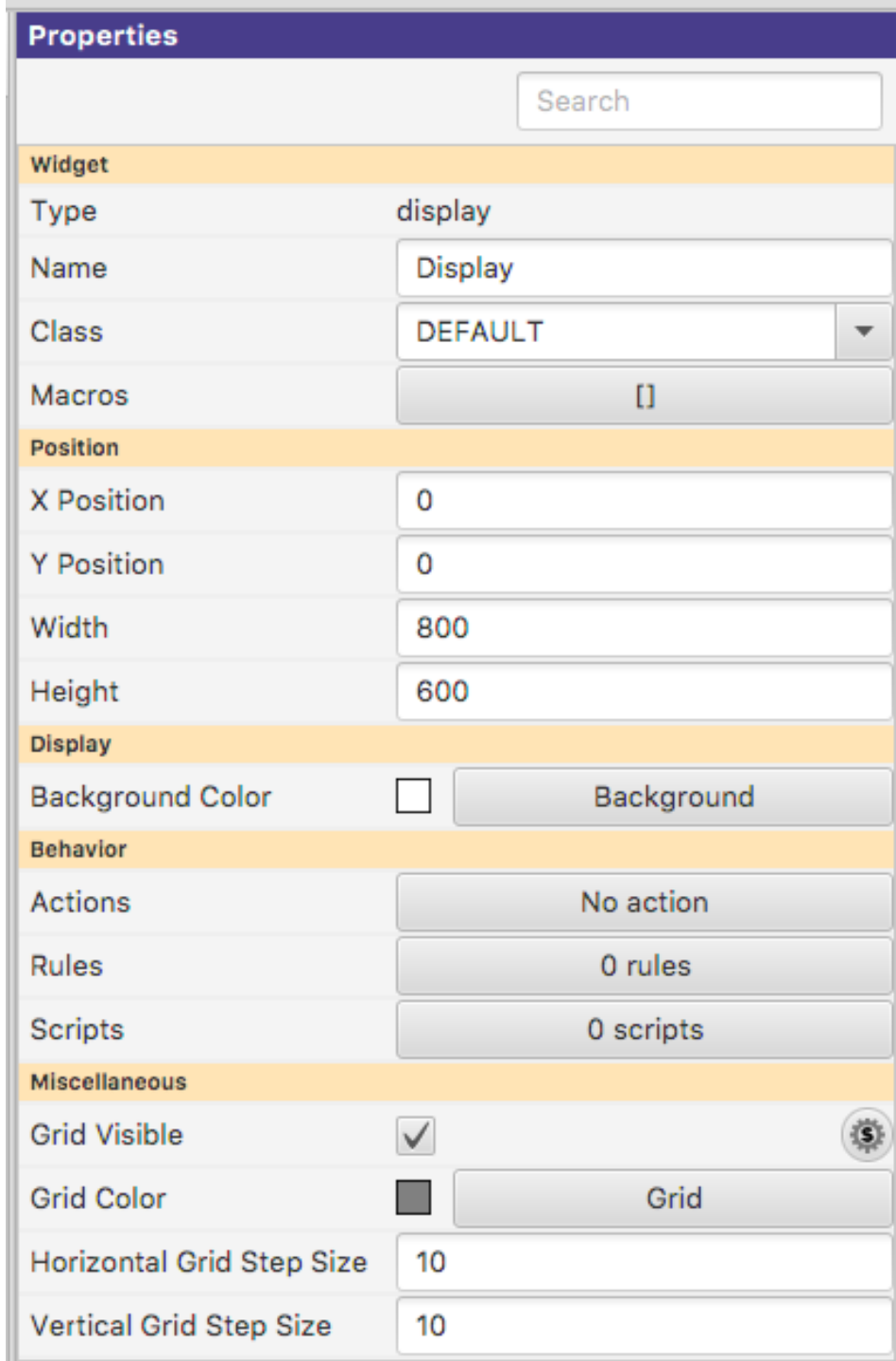
Selected Widgets Tracker

Move or resize selected widgets

Display Properties

Click on display background to select no widget for editing overall display properties

- Name
  - Shown in Tab
- Macros
  - Used by all widgets in this display
- Grid size
  - Can aid with placing widgets

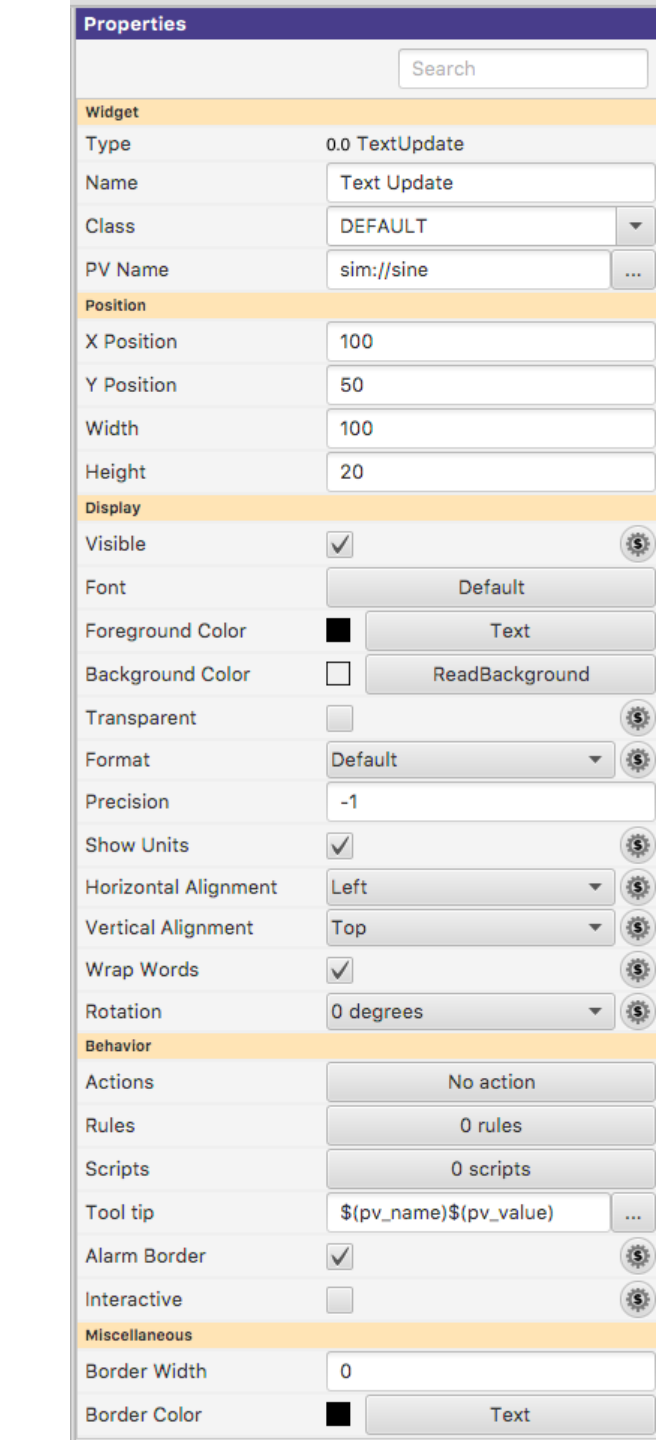


Widget Properties

Select one (or more) widgets to edit their (common) properties

- Search
  - To find desired property
- PV Name
  - Most important property for most widgets

Details depend on the widget type



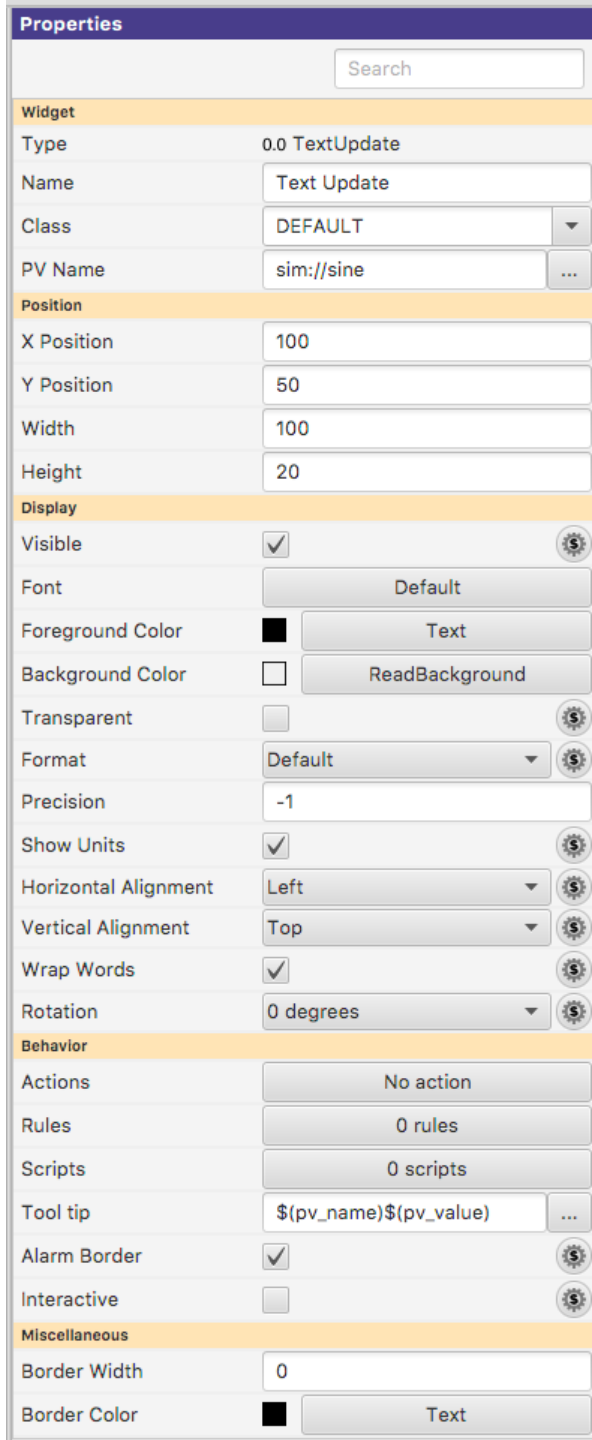
Common Widget Properties

Defaults tend to be reasonable:

- Format with precision set by PV
- Show units provided by PV
- Alarm-sensitive Border
- Fetch Items (Combo, …) from PV

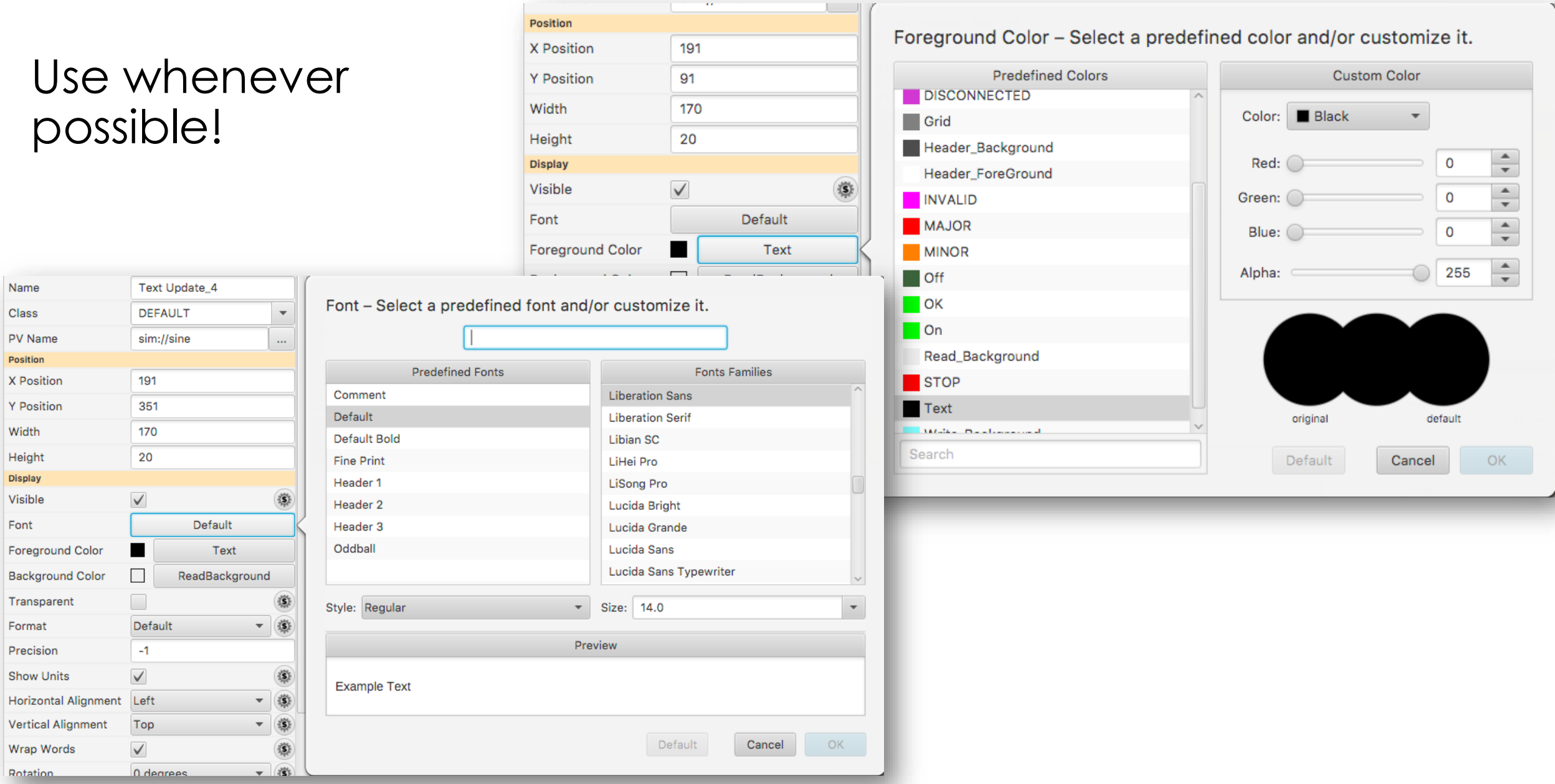
Instead of changing them,
maybe the PV needs to be updated?

Still, can be adjusted as needed for the display.



Predefined “Named” Colors and Fonts

Use whenever possible!



Configuring Named Colors, Fonts

```
# ----------------------------------------
# Package org.csstudio.display.builder.model
# ----------------------------------------


# Widget classes
# One or more *.bcf files, separated by ';'
# Defaults to built-in copy of examples/classes.bcf
class_files=examples:classes.bcf

# Named colors
# One or more *.def files, separated by ';'
# Defaults to built-in copy of examples/color.def
color_files=examples:color.def

# Named fonts
# One or more *.def files, separated by ';'
# Defaults to built-in copy of examples/font.def
font_files=examples:font.def
```

color.def

```
# Named colors
#
# Format:
#   NameOfColor = red, green, blue [, alpha ] |
PreviouslyDefinedNameOfColor
# with values in 0..255 range.
#
# Whenever possible, use named colors in displays
# instead of arbitrary red/green/blue values.

# ------- Predefined colors ----------------
# May be overridden in here

# Alarm related
OK = 0, 255, 0
MINOR = 255, 128, 0
MAJOR = 255, 0, 0
INVALID = 255, 0, 255
DISCONNECTED = 200, 0, 200, 200

# Default color for text
Text=0,0,0

# Default color for 'active' text that's being edited
ActiveText=255, 255, 0

# Display background
Background = 255, 255, 255

# .. for widgets that read/write a value
Read_Background = 240, 240, 240
Write_Background = 128, 255, 255

# .. for buttons
Button_Background = 210, 210, 210

# ------- Examples for additional colors
----------------
# Also show ideas for site-specific guidelines that
# are required to make sense of the color names.

# Styling
Header_Background=77,77,77
Header_ForeGround=255,255,255

# Use alarm colors only when you mean to indicate an
```

font.def

```
// Named font definitions
//
// Entries in this file are read in sequence.
// A later entry in the file can override
// an earlier entry in the file.
//
// In a production setup, this file may be constructed
// by concatenating a generic file with a more specific file,
// and the specific entries would then override generic entries
// of the same name.

// Format:
//
//   NamedFont['(' OS ')'] = Family '-' Style '-' Size | '@'PreviouslyDefinedNamedFont
//
// Family: Font family name "Liberation Sans", "Liberation Mono", "Liberation Serif"
//  Style: "regular", "bold", "italic", "bold italic"
//   Size: Font height in pixels
//     OS: "windows", "linux", "macosx"
//
// Leading/trailing spaces around each element are OK, but if the font family
// is "Liberation Sans", it has to be typed with just that one space between
// "Liberation" and "Sans"
//
// Examples of named fonts
//
//   Default      = Liberation Sans - regular - 14
//   Default Bold = Liberation Sans - bold    - 14
//   Header 1     = @Default Bold
//
// Speaking of "Liberation Sans":
// The display builder includes the "Liberation" fonts
// from https://fedorahosted.org/liberation-fonts.
// Their use is encouraged because the resulting displays
// will always render correctly.
// When using other fonts, for example "Arial" on Windows,
// the font might not be available to a display builder
// runtime that is executing on Mac OS or Linux.

// Predefined fonts that this file could re-define
Default      = Liberation Sans - regular - 14
Default Bold = Liberation Sans - bold    - 14
Header 1     = Liberation Sans - bold    - 22
Header 2     = Liberation Sans - bold    - 18
Header 3     = Liberation Sans - bold    - 16
Comment      = Liberation Sans - italic  - 14
```

Ideally set at start of project

Widget Notes

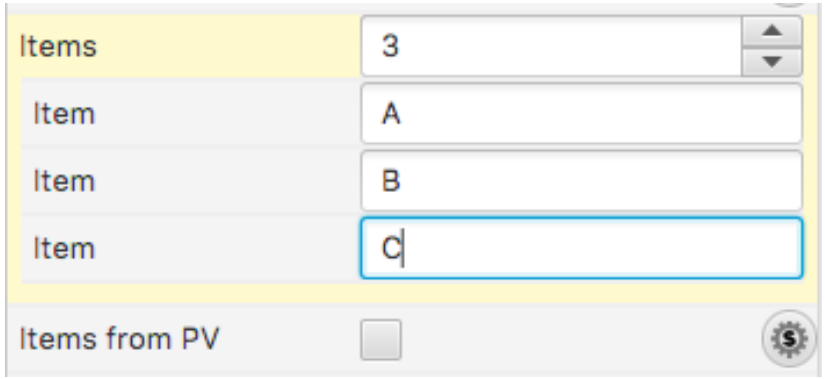
- Text Entry, Text Update:
  - Set Format = String for “long string” waveforms. Default will show array.

- LED, Boolean Button, Checkbox
  - Boolean PV
  - Numeric PV 0 or not 0 (when ”Bit” set to default of -1)
  - Bit in a numeric PV (when ”Bit” set to 0, 1, 2, …)

- Multi-State LED
  - Enumerated or numeric PVs
  - Defaults to using state values 0, 1, 2, 3, ..

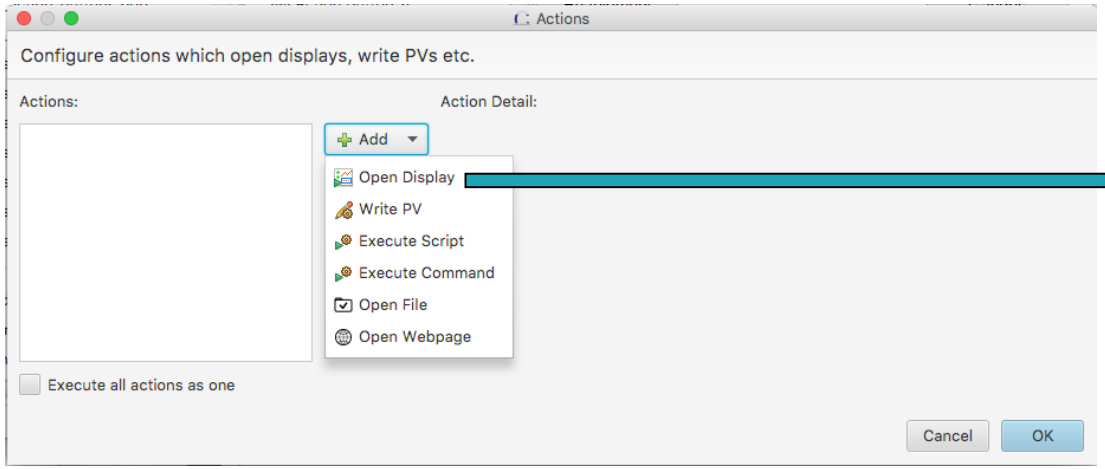
Widget Notes

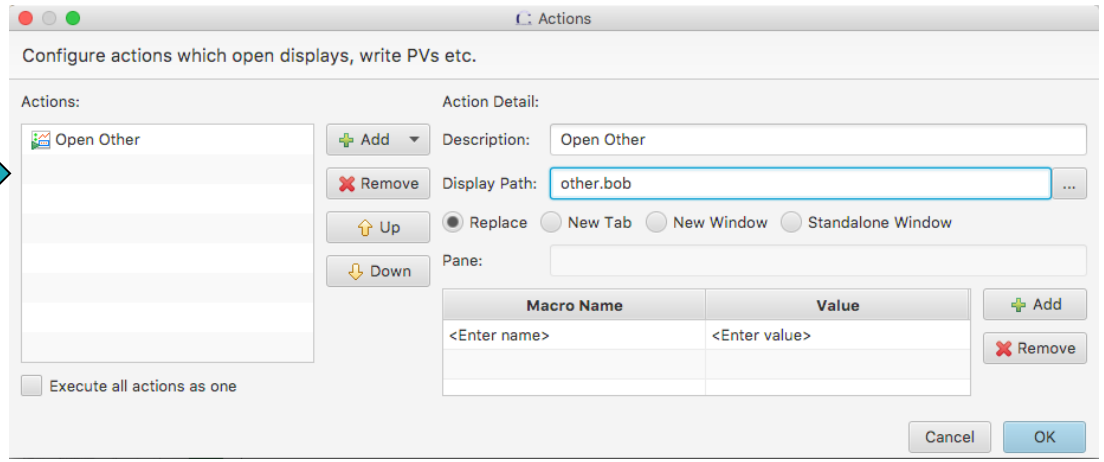
- Combo Box, Radio Button:
  - Best for enumerated PV: Enter PV name, done
  - Alternatively, un-check “Items from PV” and enter items



Action Button

1. Add ActionButton
2. Configure “Actions” property, add “Open Display”

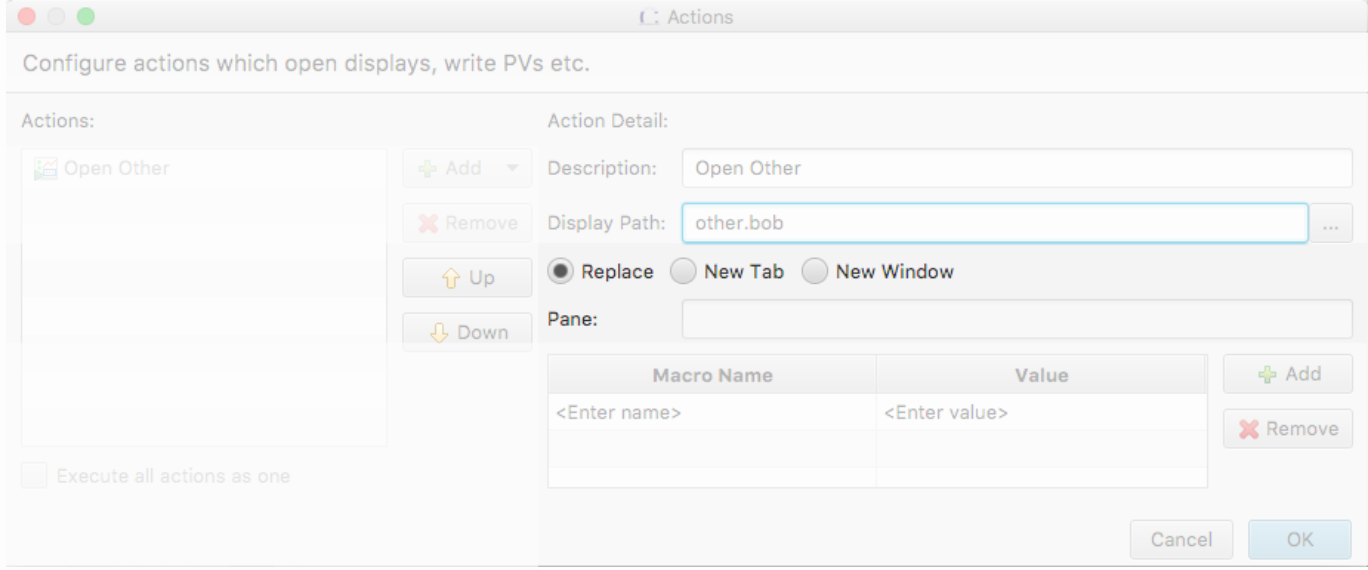




3. Run: Clicking button opens the “other” display.

In principle, any widget can have ‘Actions’.
They appear in the widget’s runtime context menu.
But it’s not obvious to end users that for example a Label will have actions.

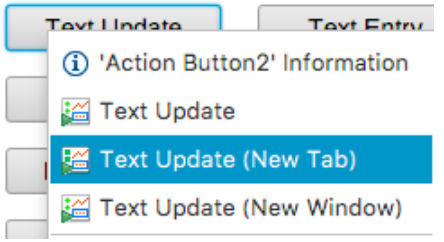
Screen Navigation



- Replace
  - Suggested default.
  - Allows back/forward navigation as in web browser

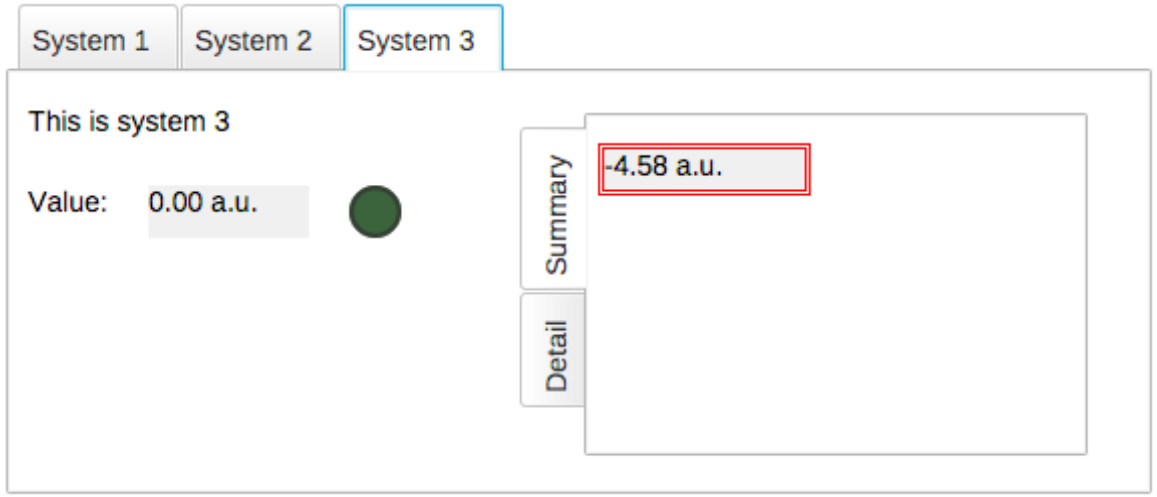
  - Minimizes number of open screens
- New Tab
  - Opens in new tab
  - Allows specific Pane name
- New Window
  - Opens in new window

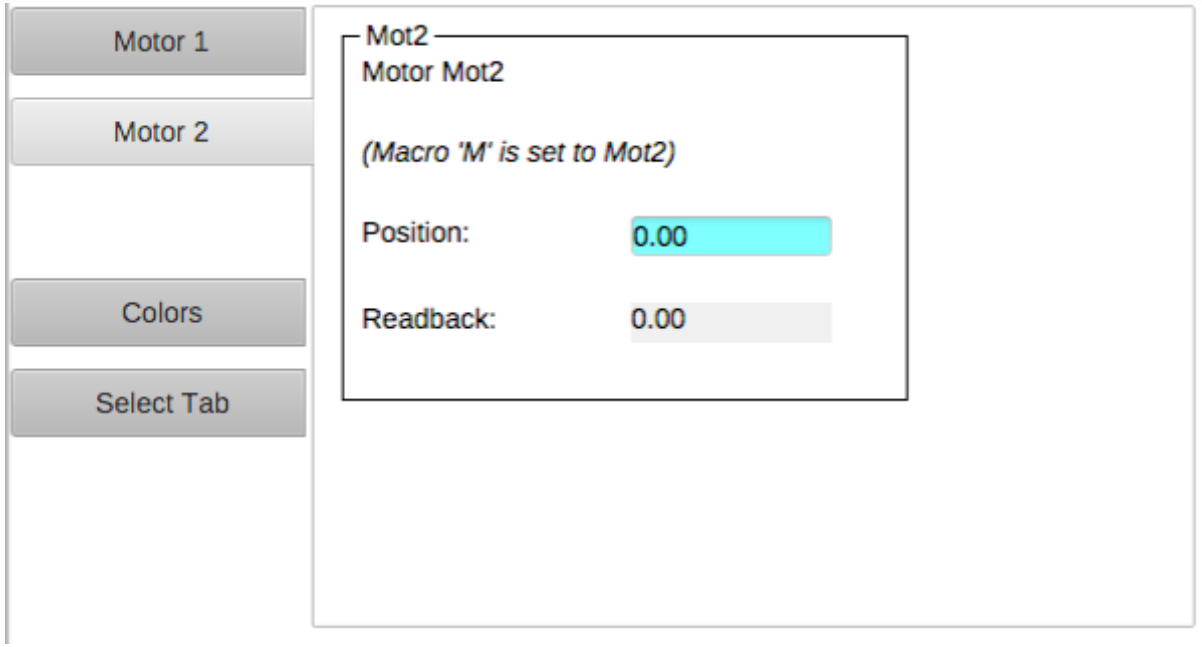
- With "Replace", can still use
  - Context menu



  - Control (⌘ on Mac) for tab
  - Shift-Control for window

Screen Navigation: Tabs





Tabs

Each tab is in-memory, same *.bob

- Appears immediately when selected
- Uses CPU and memory when hidden

Navigation Tabs

Tab is loaded from separate *.bob when selected

- May need a little time to load
- No CPU and memory when hidden

Macros

- Macros are passed into displays from
  1. Enclosing Group or Tab Widget
  2. Display
  3. Embedded widget container or Action that loaded the display
  4. Phoebus preferences
  5. Environment Variables
- To use: $(NameOfMacro)
- Examples:

PV Name:	$(PV)	with PV=TheFullPVName
PV Name:	Motor$(N)	with N=1, 2, 3, …
Width:	$(WID)	with WID=200
Visible:	$(SHOW)	with SHOW=true

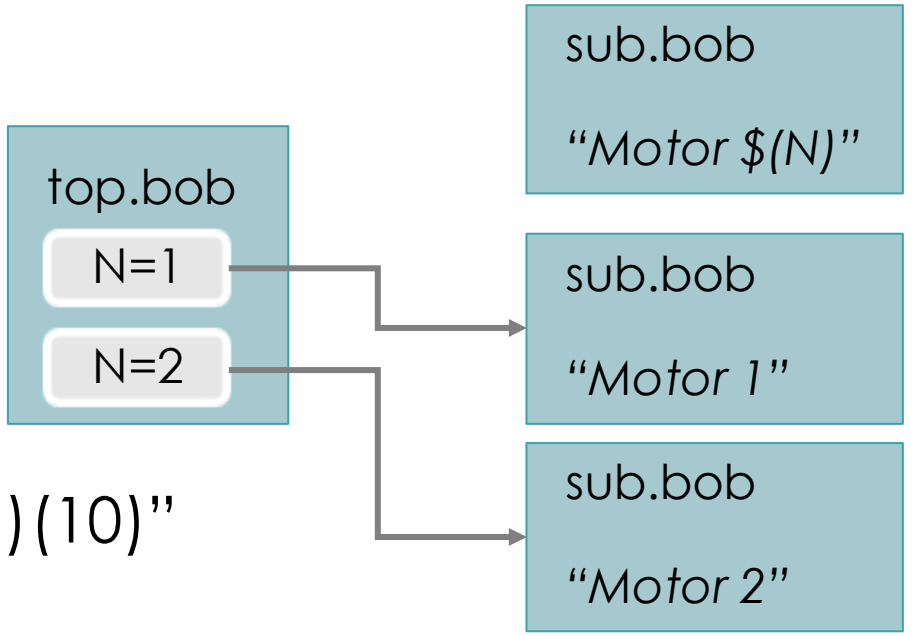
.. or ${NameOfMacro}.
EPICS *.db files use $(xx),
SNL and shell use ${xx},
so we support both conventions.

Macro Example

1. Create display “sub.bob”
   - Label with text “Motor $(N)”
   - TextUpdate with PV “loc://pos$(N)(10)”
   - ActionButton with PV Name “loc://pos$(N)(10)” and Action to “Write PV” value 20
   - Copy that button, update to set PV to 30

2. Create display “top.bob”
   - ActionButton with Action to open sub.bob with N=1
   - Copy/paste the button, update to N=2

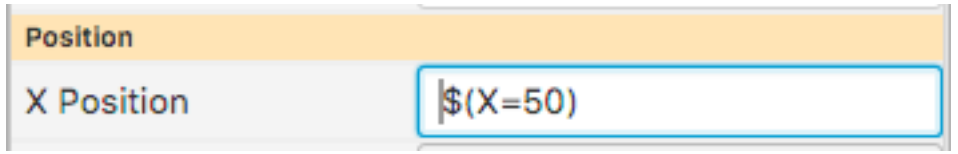
3. Execute top.bob, press buttons



Macros

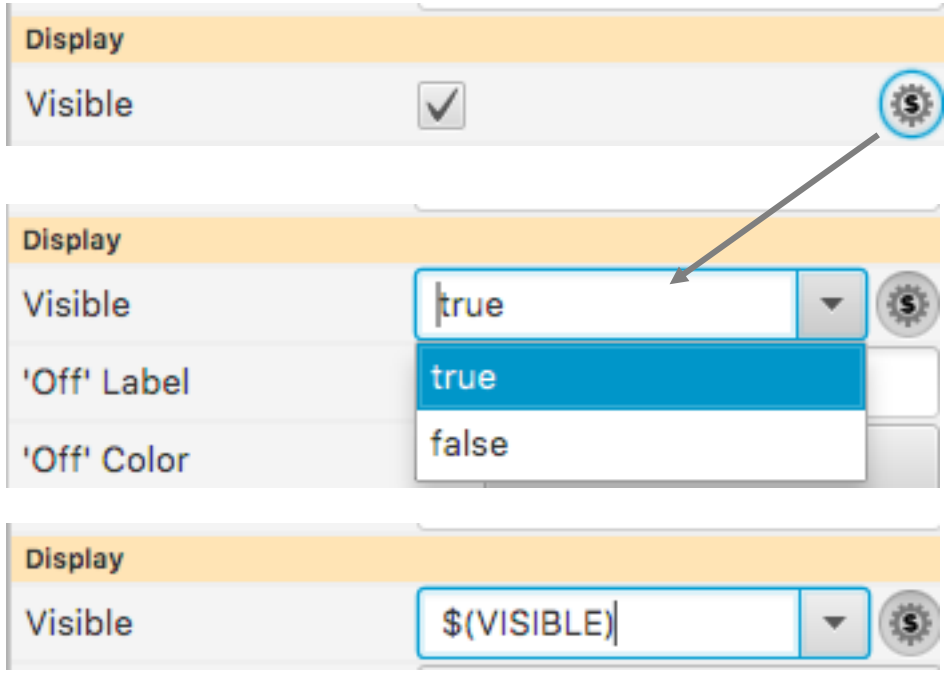
- Default values: $(MACRO=default)

  Allows standalone testing without passing values into display



- To enter macro for Boolean

  Press the "$" macro button



  Select valid option from drop-down …

  .. or enter a macro

Macro Fallbacks

When macro is not defined, falls back to

- Widget Properties
  - Uses the internal property name shown in tool-tip of Properties view
  - Note how tooltip is usually preset to "$(pv_name)\n$(pv_value)"
  - Action Button has PV Name property.
    It's not used directly as in other widgets with PV name,
    but in "Write PV" the PV name is preset to $(pv_name)
  - Action Button text is preset to "$(actions)"
- Java Properties
  - $(os.name)
- Environment Variables
  - $(HOME), $(USER)

Predefined Macros

$(DID): Unique display identifier, useful for per-display PVs
loc://x$(DID)(10)

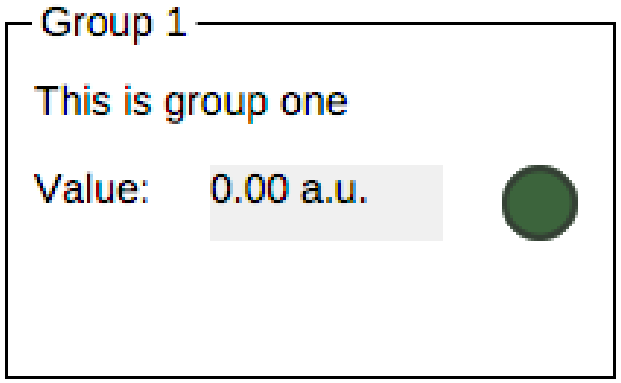
$(DNAME): Display Name

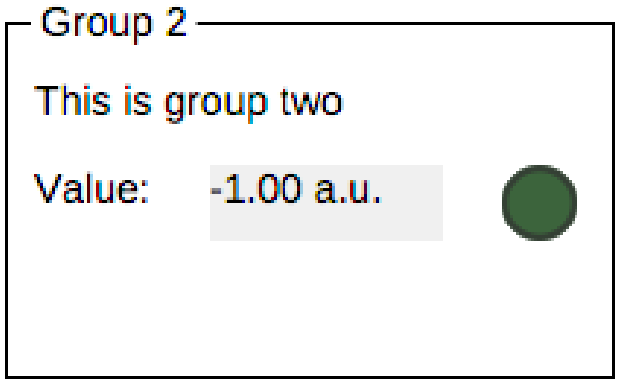
Group Widget

Contains other widgets

Visual Effect:

- Border, Name

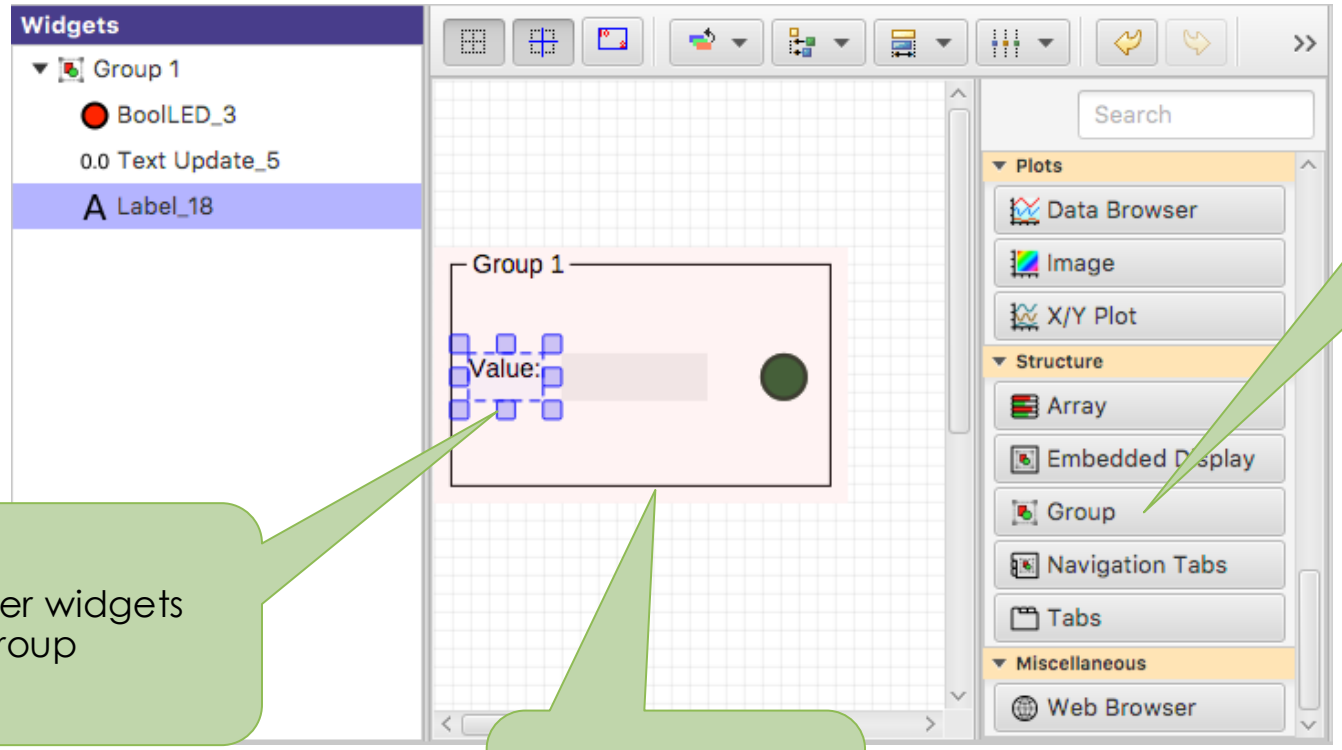




Practical Effect:

- Group can define macros for contained widgets
- Group can be moved, copied/pasted as one unit in editor

Group Widget



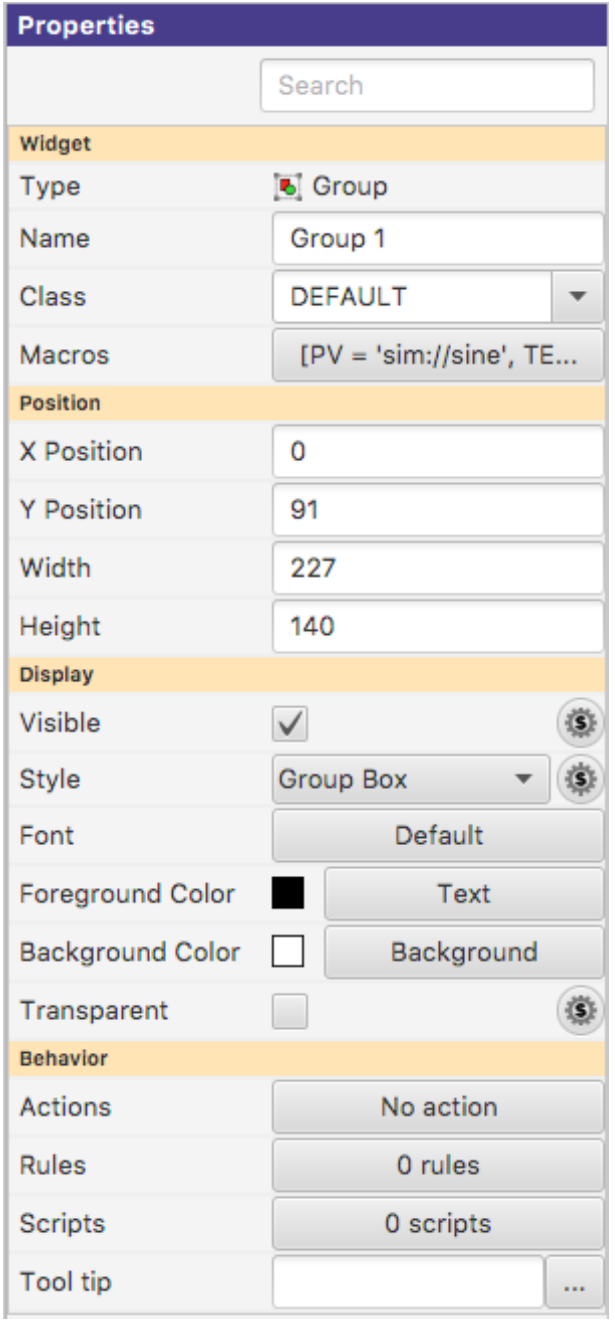
1) Add Group Widget

2) Move other widgets inside the Group

Active Group is highlighted

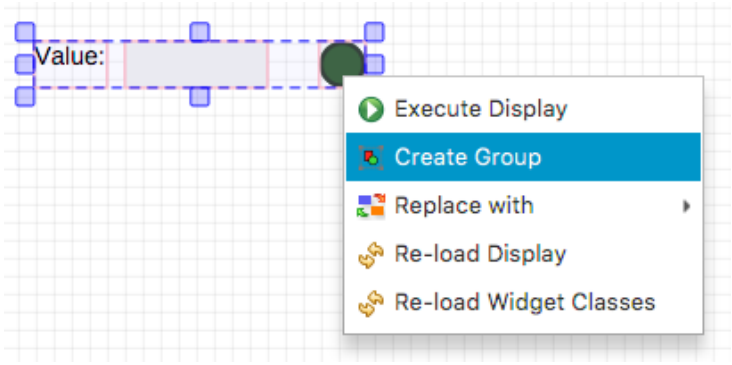
Group Properties

- Name:
  Shown in border
- Style:
  “Group Box” for named border
- Macros:
  Passed to contained widgets

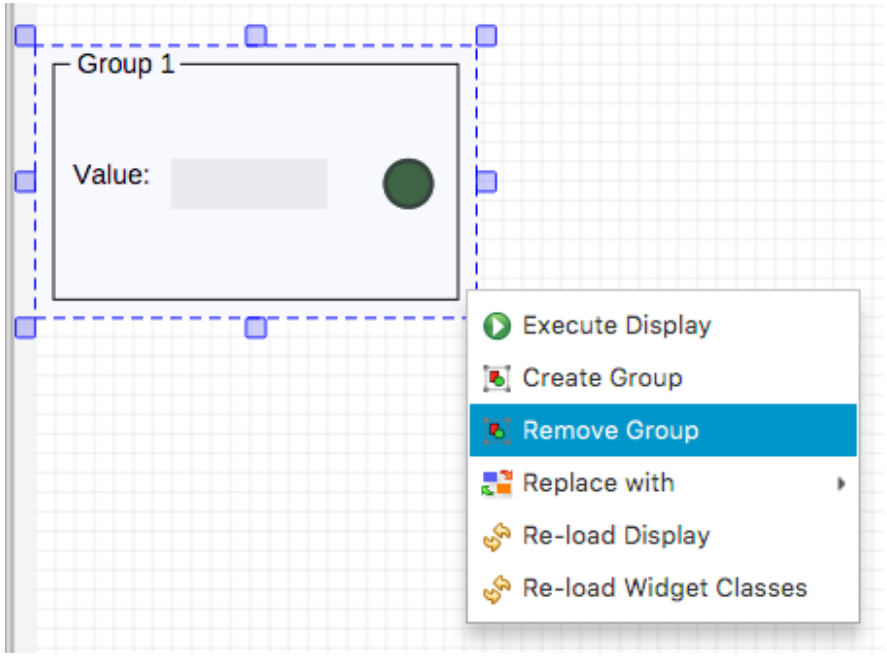


Group Editing Shortcuts

1. Select Widgets
2. Context menu “Create ..”



1. Select Group
2. Context Menu “Remove..”



Embedded Display

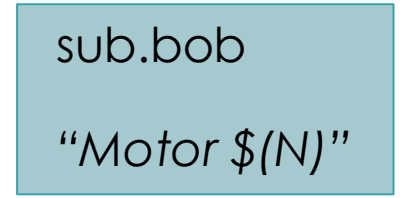
Hosts a complete *.bob file within a widget

Allows composing higher-level displays from smaller displays:

- Per-device *.bob
- Show multiple devices in one display

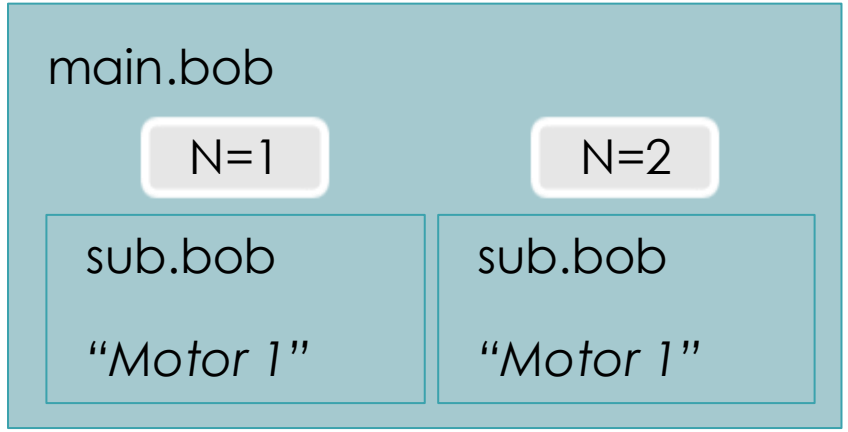
Embedded Display Example

1. Create display "sub.bob" (or use the one created earlier)
   - Label with text "Motor $(N)"
   - TextUpdate with PV "loc://pos$(N)(10)"



2. Create display "main.bob"
   - Embedded Display, File sub.bob, Macros N=1
   - Copy/paste the Embedded Display, update to N=2

3. Execute main.bob



Embedded Display Sizes

a) Embedded Display Size
   - Size of the widget that will host the *.bob
   - Defined by the widget Width and Height properties

b) Content Size
   - Size of the *.bob
   - Defined by that Display Width and Height properties

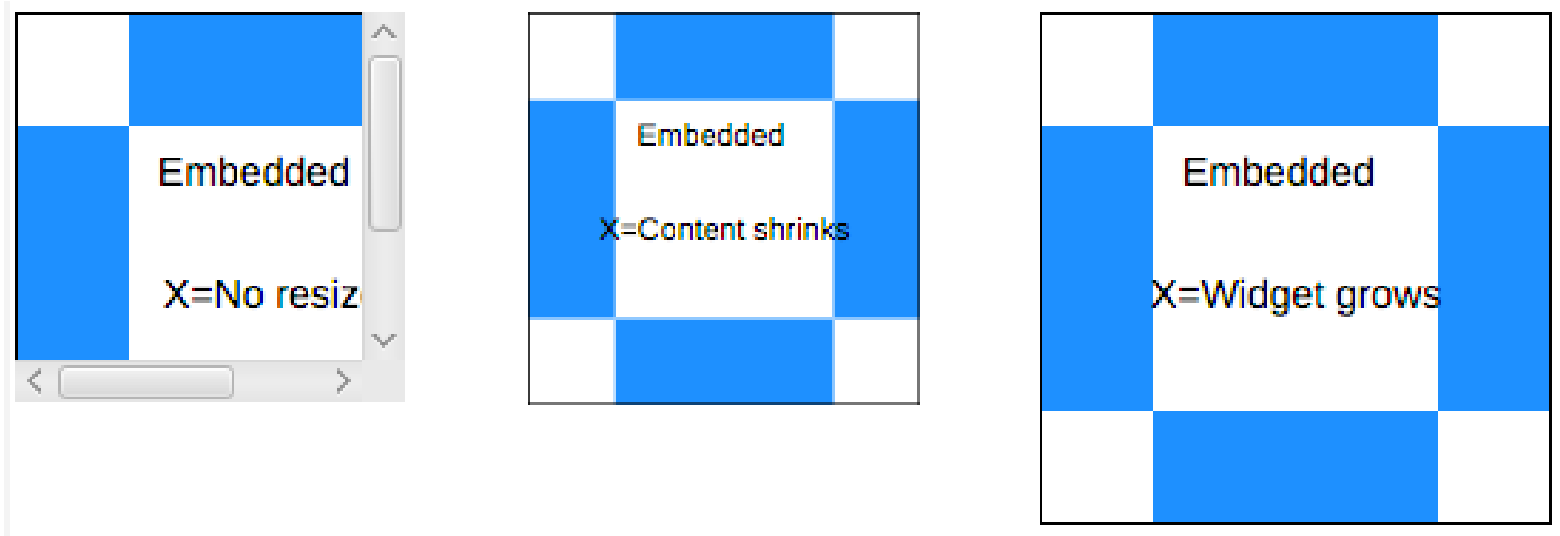
What if those sizes differ?

Embedded Display Resize Options

No Resize

Size content to fit widget

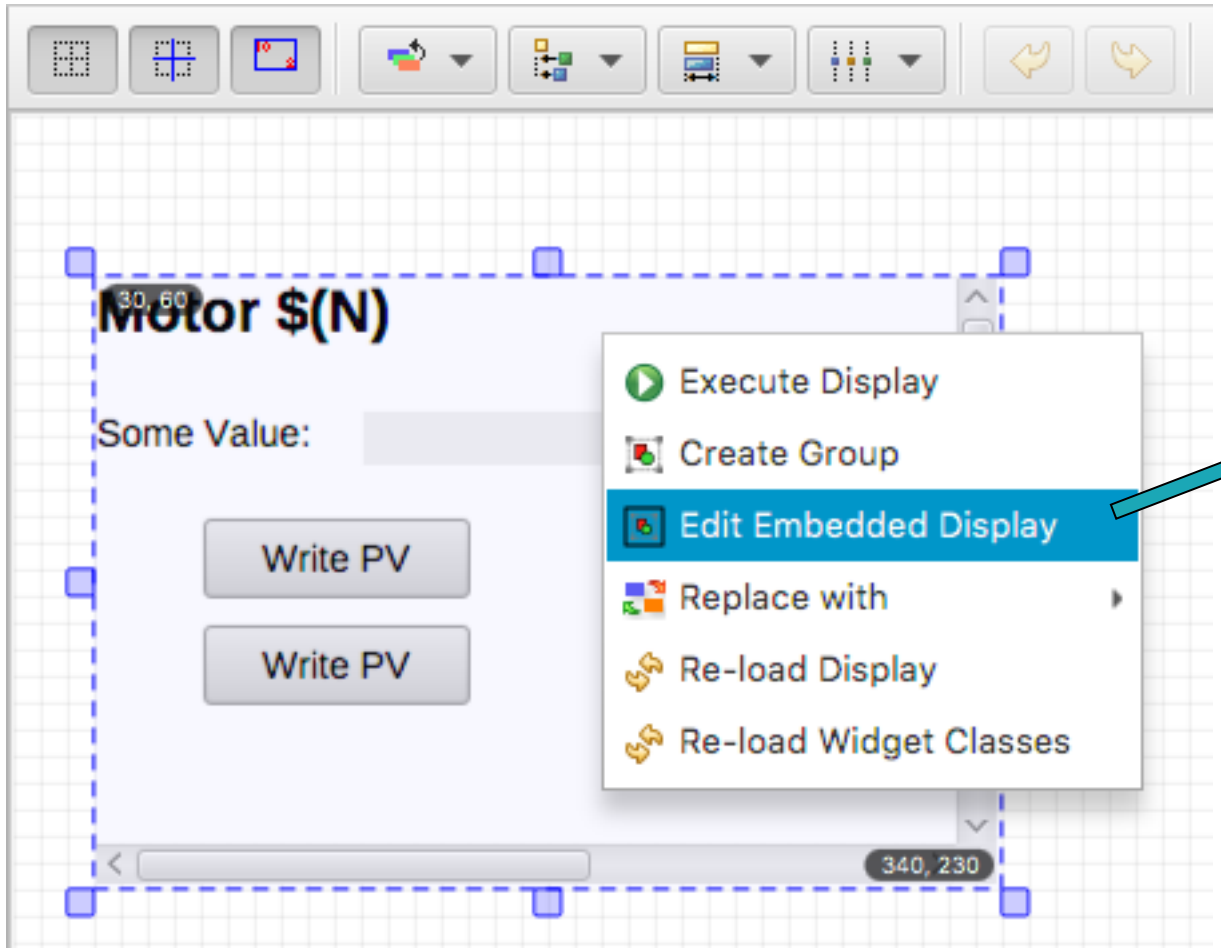
Size widget to fit content

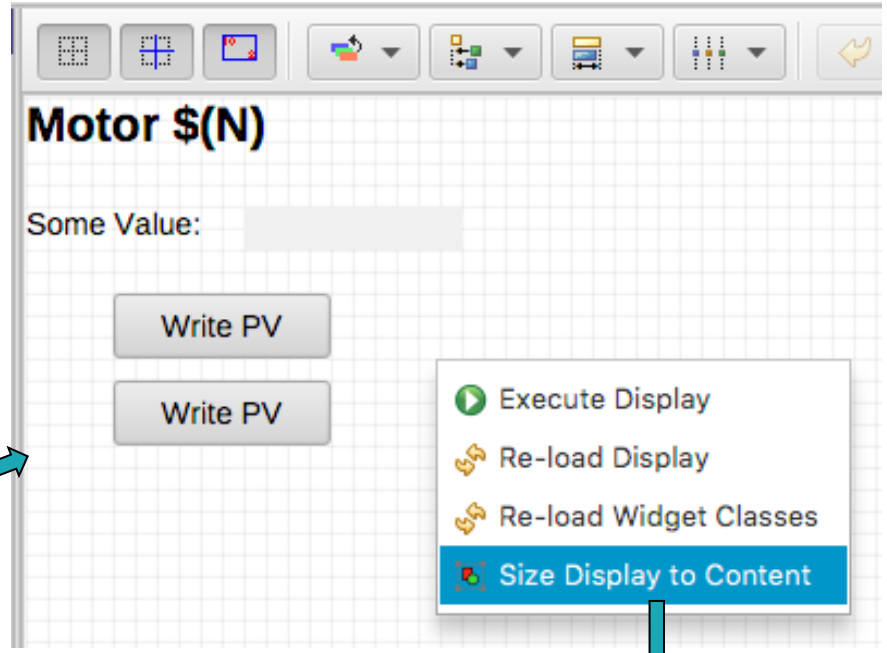


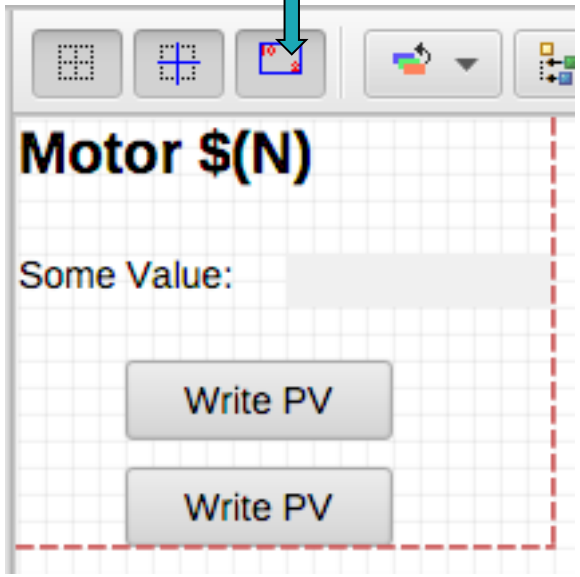
✓ No Resize usually best. Scrollbars appear as needed.

- Resizing results in odd font sizes or widgets that outgrow their initial space.

Embedded Display Editing








TO BE
CONTINUED....

Top Resources

See Help, Preferences Listing

```
# ----------------------
# Package org.phoebus.ui
# ----------------------

# Top resources to show in "File" menu and toolbar
#
# Format:
# uri1 | uri2,Display name 2 | uri3,Display name 3
top_resources=examples:/01_main.bob?app=display_runtime,Example Display | pv://?sim://sine&app=probe,Probe Example | pv://?sim://sine&loc://x(10)&app=pv_table,PV Table Example | http://www.google.com?app=web, Google
```

Start phoebus with "-settings /path/to/my_settings.ini":

```
org.phoebus.ui/top_resources=/home/controls/displays/main.bob, Start Page |
                             http://controls.my.site/displays/main.bob, Start Page
```

- File system: Use NFS or 'git pull' to distribute files
- http: All users always see the same set of files

Many Widgets and Properties

Compared to earlier EPICS display tools,

- Group Widget instead of Lines
- LED instead of Circle-with-changing-color
- Tab/Navigation Tabs instead of buttons, local PVs, conditional visibility,..

Display describes Meaning:

- Group of related widgets
- LED for binary PV, not circle that happens to change color

Files with meaning are easier to translate into the next tool

Widget Classes

- Instead of creating a Label with large font, define a "TITLE" class for the Label
- Instead of creating an LED with Orange color, define a "WARNING" LED class

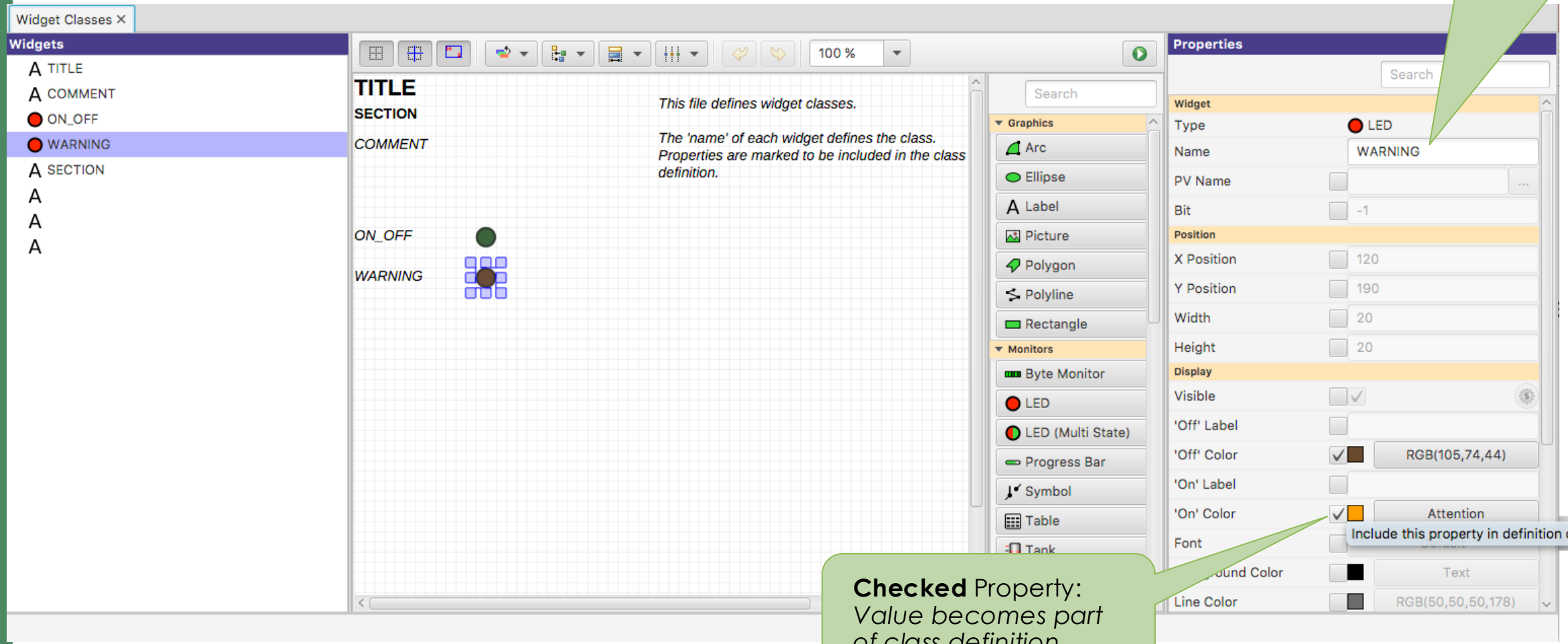
```
# ----------------------------------------
# Package org.csstudio.display.builder.model
# ----------------------------------------


# Widget classes
# One or more *.bcf files, separated by ';'
# Defaults to built-in copy of examples/classes.bcf
class_files=examples:classes.bcf
```

Editing *.bcf Widget Class Files

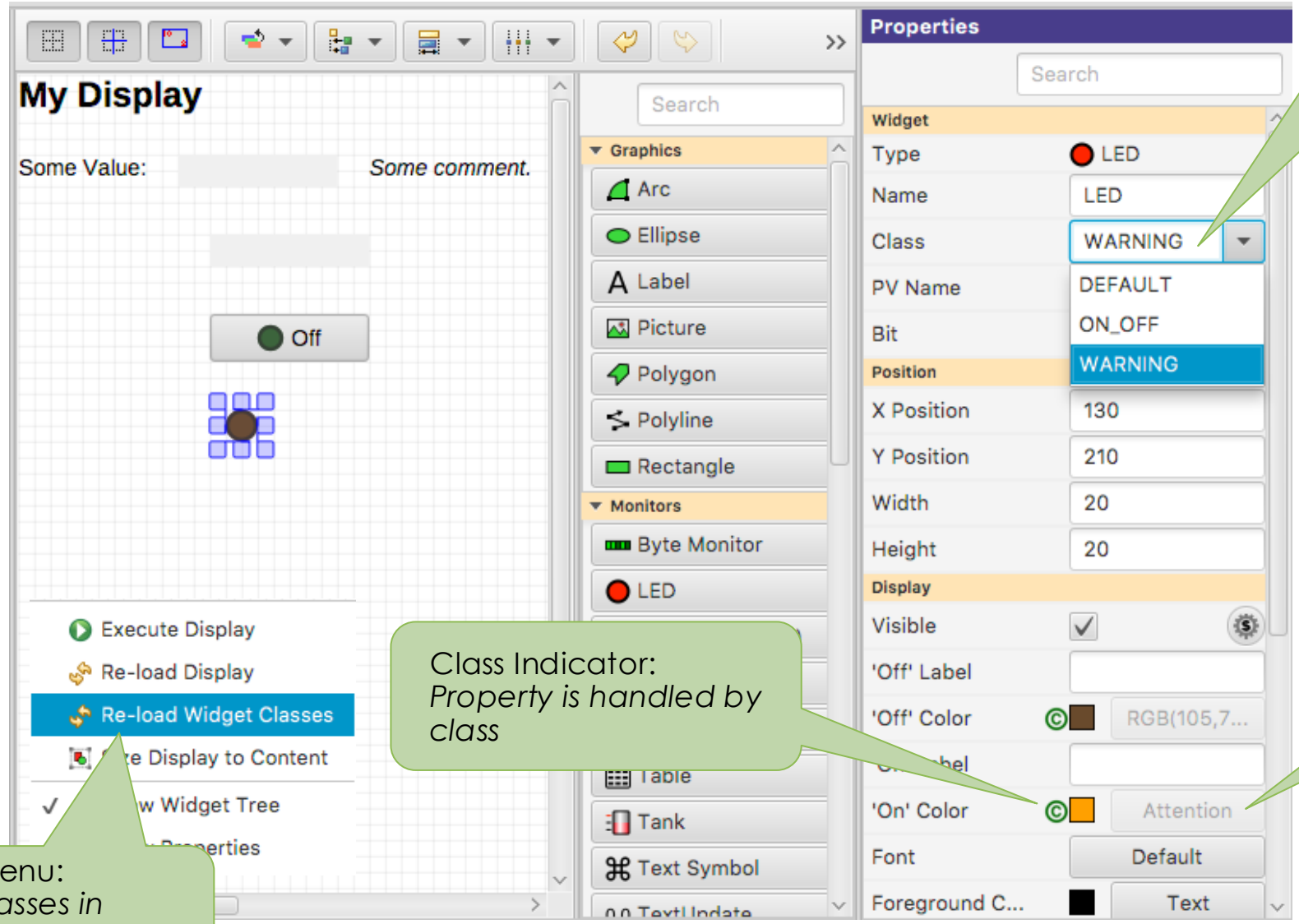
Slightly different editor behavior

Name Defines a widget Class:
'WARNING' LED,
'TITLE' Label,
…



Checked Property:
Value becomes part of class definition

Using Widget Classes



Select Widget Class

Class Indicator:
Property is handled by class

Disabled:
Cannot change the class-based property

Context Menu:
*Re-load classes in case *.bcf is changed while editing display*

Class Details

- *.bcf files define widget classes
  - *Label* of class *TITLE* uses font XYZ
- When editing a *.bob file, classes are applied.
  Add Label, select Class TITLE:
  - Font is set to XYZ
  - Can no longer change the font
  - File is saved with font=XYZ, marked as "use_class"
- *.bob files use widget classes, if they are defined.
  Open a file with Label of class TITLE, and
  a) TITLE is a known class:
     Whatever that class defines is used. If it sets font=EFG, that'll be used.
  b) TITLE is not a known class:
     Using font=XYZ as saved in file.

Compare *.bcf and *.bob to *.css and *.html

*.bcf classes are similar to *.css style settings,
*.bob files are similar to *.html content

a) Have same *.bcf/*.css

→ Display looks the same

b) Use different *.bcf/*.css

→ Display looks as requested in my *.bcf/*.css

c) Have no *.bcf/*.css

→*.html turns into rubbish, lacking any description of what to look like.

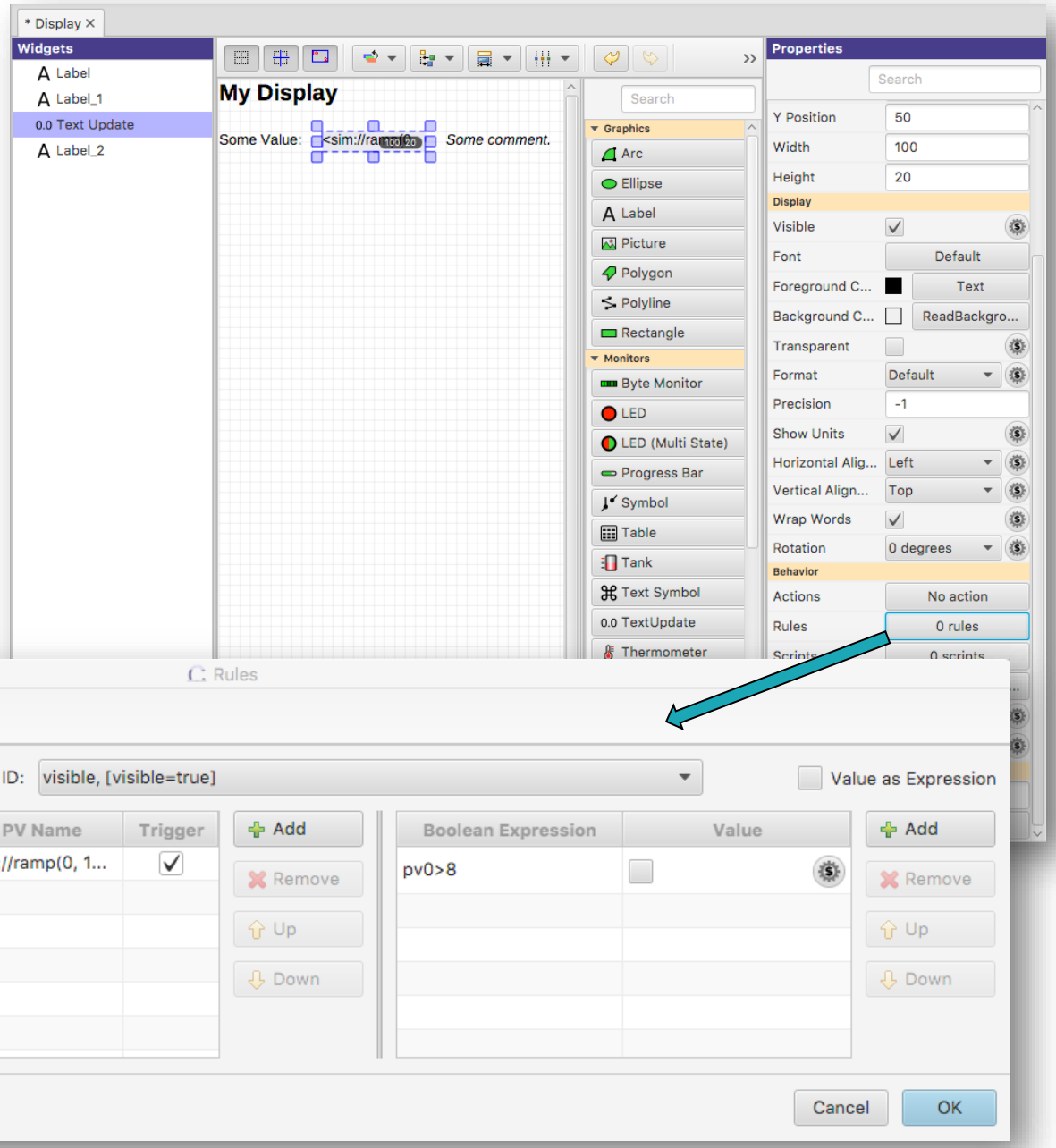
*.bob display looks as seen by last person who edited it,
since the class settings effective at that time are in the *.bob file.

Rules

- Ideally, use widgets' built-in functionality
  - Value of PV displayed in TextUpdate, LED, ..
  - Alarm indicated via Border
- Sometimes useful to for example hide a widget, i.e. change visibility based on a PV
  - Rules can accomplish this
  - .. But functionality may not be obvious to the next person who needs to maintain a display

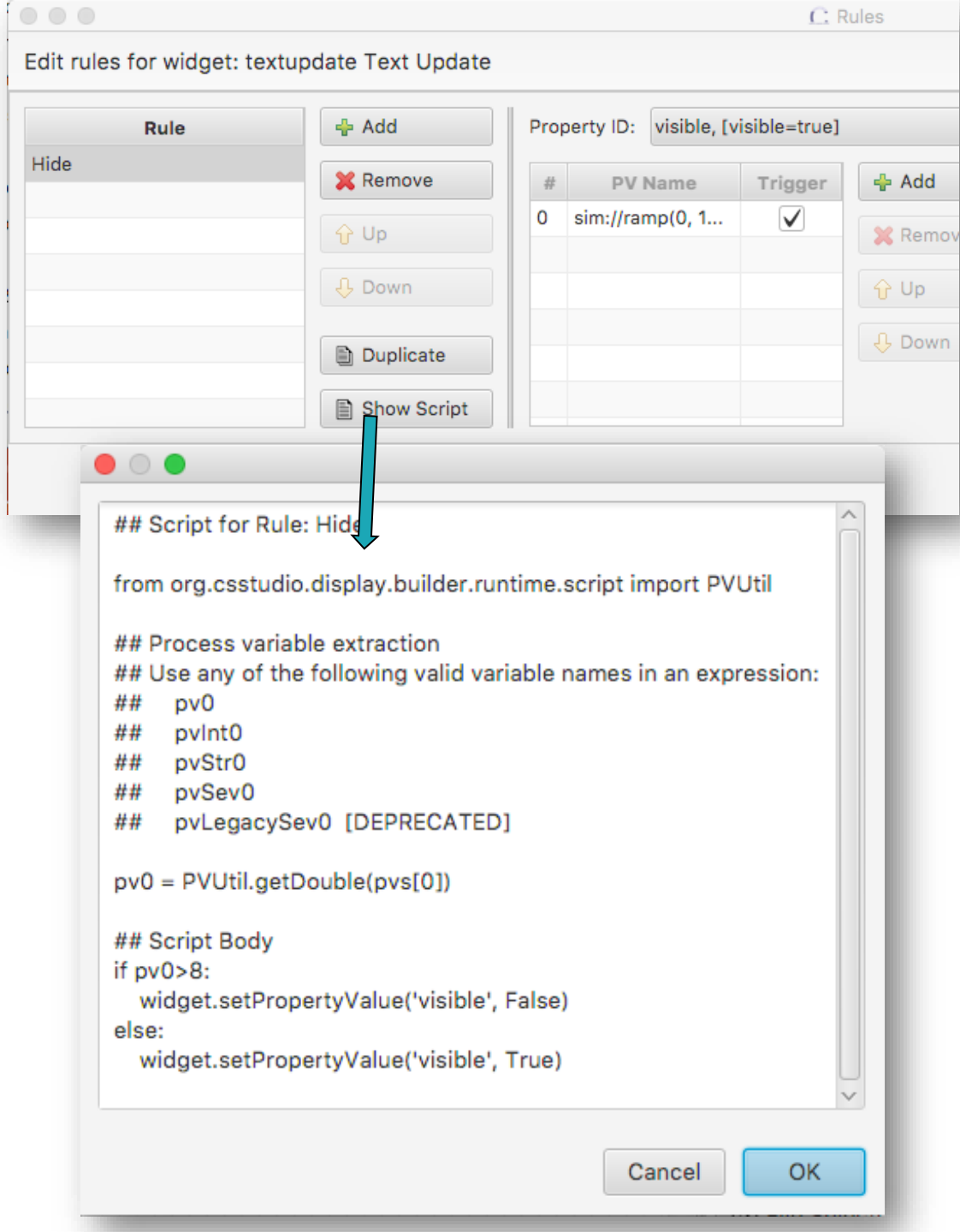
Adding a Rule

- Add TextUpdate widget
- Set PV to sim://ramp(0, 10, 1)
- Open Widget's Rules
- Add Rule, name it "Hide"
- Select "visible" property
- Add PV sim://ramp(0, 10, 1)
- Add Boolean Expression "pv0>8"
- Un-check value
- Run



Rules Detail

- Triggered by at least one PV
  - May use additional non-trigger PVs
- Expressions use pv0, pv1, .., pvStr0, pvStr1, .. to access PVs' values
- Rule internally converted to Jython
  - Use preview to debug
- "else: .." sets property to original value



Scripts

- Scripts are attached to a widget
- Triggered by at least one PV
  - May use additional non-trigger PVs
- Invoked with
  - pvs[] – Array of requested PVs
  - widget – The widget
- Script can
  - Read & write the received PVs
  - Set widget properties
  - Locate other widgets in the display
  - Invoke any Java code in the product
  - Be very powerful
  - Result in an unmaintainable mess
- One Script Executor per *.bob file, Runs in background thread
  - Slow scripts do not block the UI
  - One script per display at a time
    a) Many short-duration scripts
    b) One that never quits

Rules vs. Scripts

- Both are in the end Jython code
- Both should be the exception.
  Plain displays don't need them.
  But can be powerful,
  replacing separate custom Java/Python/C/C++ applications.
- Prefer Rules because they describe meaning, easier to maintain

When to use a script

- It's simple, well documented, and tremendously improves the UI
- Would be a one-of, specialized, hard to maintain, separate application anyway.
  With a script, at least its integrated into the operator UI

Examples:

- Turn scalar PVs into loc://waveform for guideline in XYPlot
- Fill display with 50 widgets based on config file, examples/template_and_script
- Add information from web service to display

When not to use a script

- It adds logic to the display that should be on the IOC
  - Display should only display PVs and allow user to write PVs.
  - Display must never do anything
- You have to ask for help implementing the script
  - If you can't implement it, you can't maintain it, either

Examples

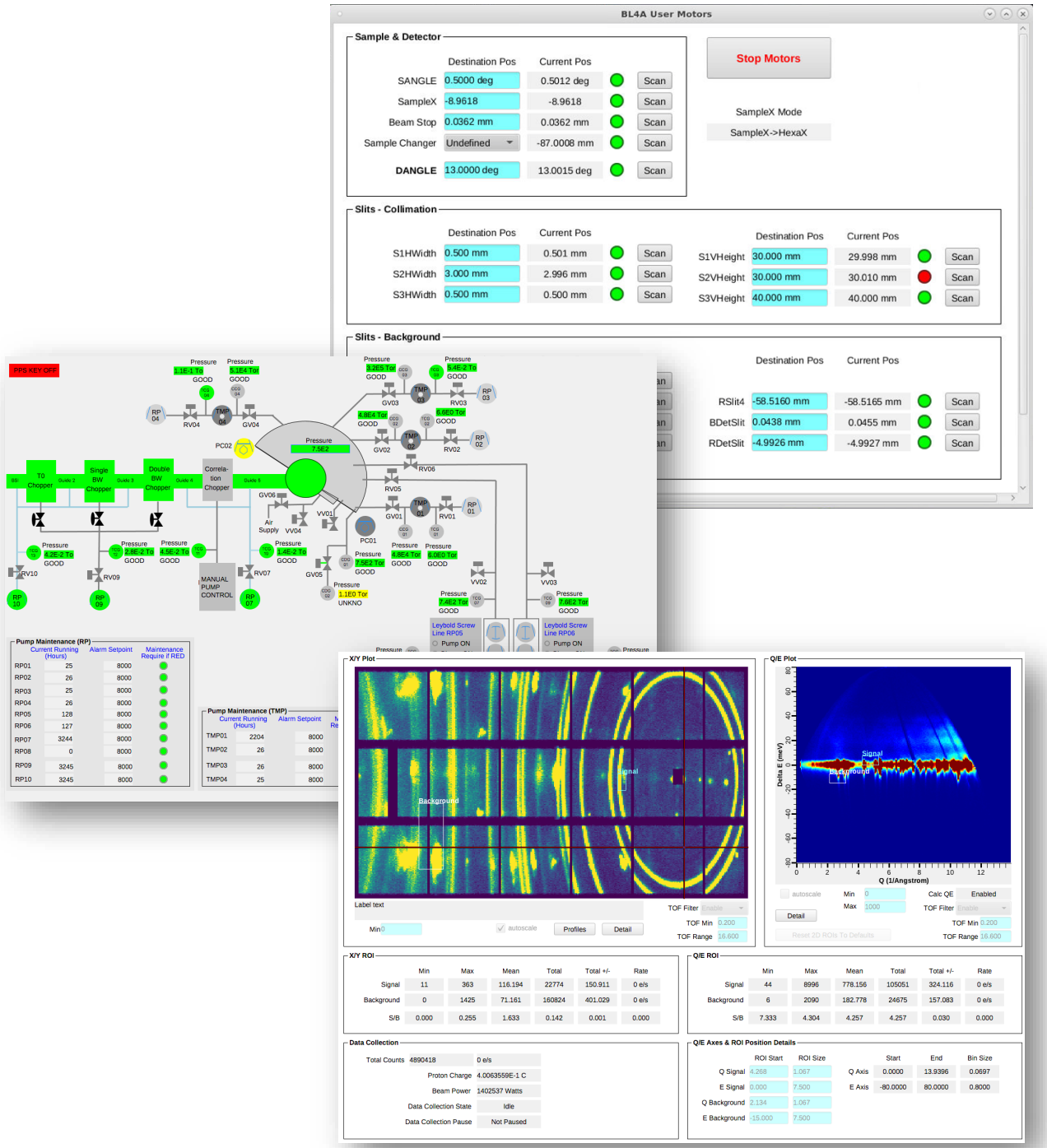
- Open relieve valve when pressure too high. Ramp Power Supply.
  - What if somebody closes the display? Opens two displays?
- Wiggle something on the display
  - It's not a video game

Summary

Display Builder is powerful Editor and Runtime with many Widgets, Macros etc.

Keep it Simple

1. Add a Widget
2. Enter Label's Text or Widget's PV Name
3. Done



OAK RIDGE National Laboratory | SPALLATION NEUTRON SOURCE