DEVCON
Enabling the Smart Society
OCTOBER 22-25, 2012
HYATT REGENCY ORANGE COUNTY

Extreme Makeover with the RX600:
Adding Touch/Graphics to Your Product

Terry West, CEO
Serious Integrated Inc.

Class ID: **5L03I**

Renesas Electronics America Inc.

We are… *Serious*

PLATINUM PARTNER RENESAS

- Founded 2008 by former RIM, Microchip, and Intel Executives
- Experts in Hardware, Software, Comms, GUI design

Terry West: CEO and co-founder

- 1st RIM Employee
- Built XScale® business at Intel
- Launched PIC32 MCU at Microchip

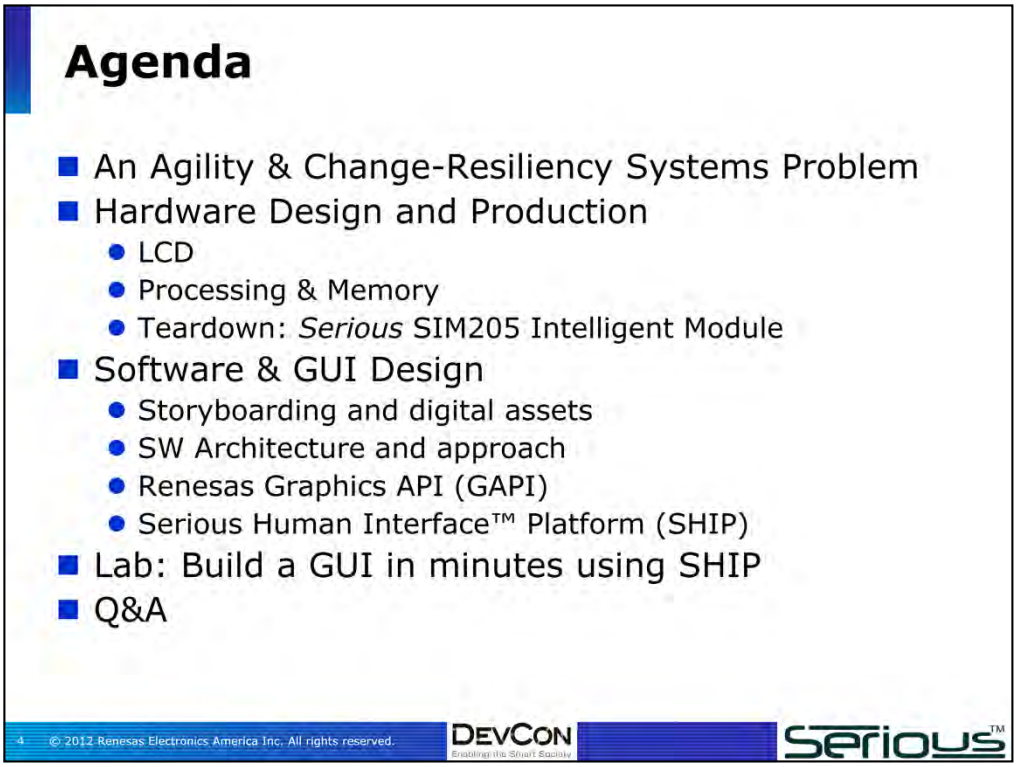
Serious enables remarkably **fast time-to-prototype and production** of an OEM's new **graphic/touch front panel** with **rapid GUI development tools**, **software**, and **off-the-shelf hardware**.

DEVCON Enabling the Smart Society

Purpose

- RX MCUs: best in class for cost effective engine graphic/touch front panel applications
- Graphic/touch: **hard engineering problems!**
  - what software?
  - what OS?
  - what GUI development strategy?
  - will the HW support the SW?
  - will the GUI end up looking great?

This workshop covers the major steps & decisions when developing a complete graphic/touch front panel on the Renesas RX MCU Family.


Agenda
An Agility & Change-Resiliency Systems Problem
Hardware Design and Production
LCD
Processing & Memory
Teardown: Serious SIM205 Intelligent Module
Software & GUI Design
Storyboarding and digital assets
SW Architecture and approach
Renesas Graphics API (GAPI)
Serious Human Interface™ Platform (SHIP)
Lab: Build a GUI in minutes using SHIP
Q&A
4
© 2012 Renesas Electronics America Inc. All rights reserved.
DEVCON

Target Markets & Examples

Home Automation

Fitness

Appliance

Security

Entertainment

Office Equipment

Industrial

Medical

Military

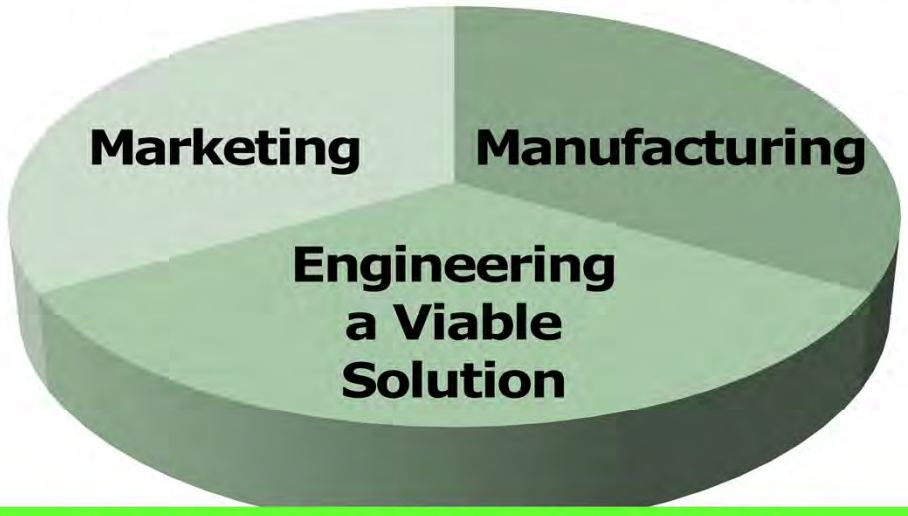
Avionics

Automotive

Building Control

Diverse Applications, Common Touch/GUI Challenges!

Your Multi-Faceted Challenge



Your system design must…

- Help marketing & sales w/fast time to prototype
- Empower marketing with GUI change agility
- Enable purchasing across supply line changes
- Deliver a great GUI at a viable cost

Hardware Design

- LCD
- Processing & Memory
- Teardown: *Serious* SIM205 Intelligent Module

LCD Selection: Many Issues

- **LCDs in the Embedded World**
- **Touch Technology**
- **Drive Technology**
- Quality
- Brightness/Luminance
- Interoperability
- Operating Range and MTBF

LCDs in the Embedded World

- Few long-lifetime providers
- Some long-lifetime models, sizes
  - 4.3/7.0 ok – avoid 5/6/8! – careful on 3.x,10.x
- Must examine volume drivers
- Many companies in supply chain:
  - Chips, Glass, Assembly, Materials
- Chip change → SW change
- Glass change → PCB change

Plan for LCD Refreshes in HW, SW, and Purchasing

Touch Technology

- Resistive still cheapest, easiest
  - A2D w/SW or HW controllers
- Capacitive problematic
  - Very hard systems integration
  - Reliable controllers very expensive
  - Controller lifetime highly questionable
  - Patents, Patents, Patents
- No-touch with button/LED wraps still common for "harsh" environments

Drive Technology

- RAM Buffer On Glass + Parallel Port
  - Common on <4" LCDs
  - Need to check timing carefully
  - Don't believe the data sheet
- Direct Drive to RGB
  - Great looking GUIs at very low cost
  - Common on WQVGA (4.3" typically)
  - Some limits on GUI capabilities
- Graphics Controller to RGB
  - High performance graphics
  - Many accelerator options (2D, Video, Camera overlay,...)
  - Harder to contain costs and SW design portability

MCU/MPU? 16/32?

- Go 32-bit or go home
  - Software is far more than you expect
  - You *will* want an RTOS
  - Graphic objects make 16-bit addressing/data problematic
- Think three times before jumping to an MPU
  - Much higher HW costs (power supplies, clocks, ….)
  - Higher power
  - Larger software inevitable
- For <= WQVGA, the RX600 is one of the most cost-effective and powerful choices

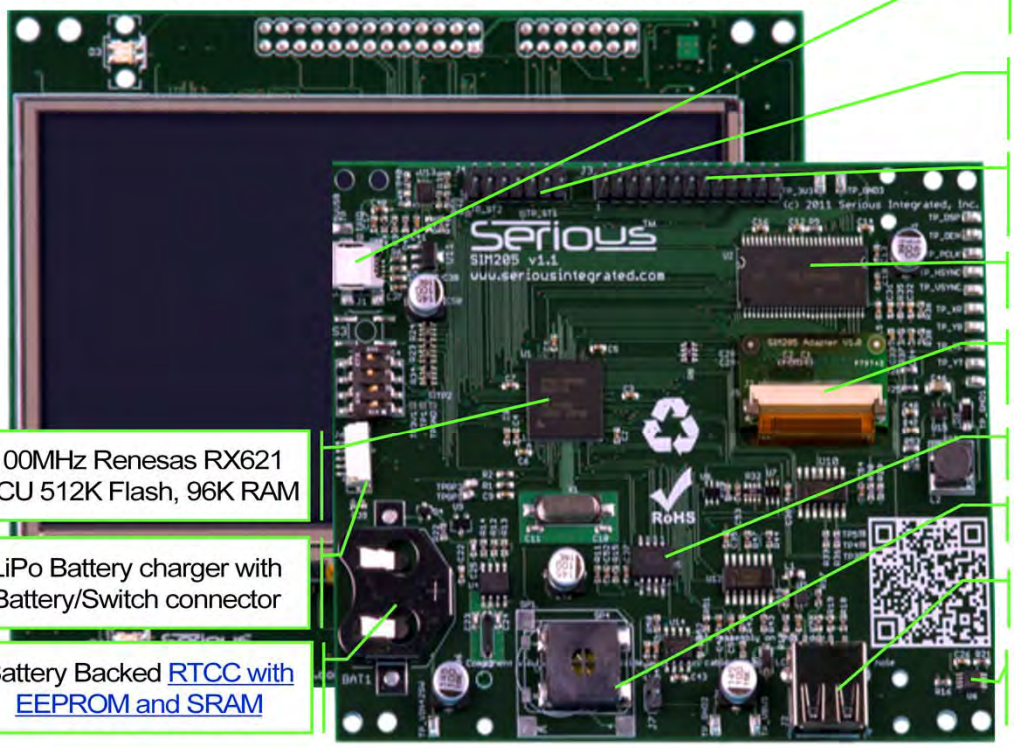
MCU/MPU Considerations

- Choose the best for the resolution, GUI expectations, and "other" CPU requirements
- Code & memory bandwidth is important!
- I/O bus can make/break GUI performance
  - Match I/O bus to optimized LCD Drive
  - E.g. RX600 SRAM bus with OTM4001A RAM-on-LCD controller
  - E.g. RX600 SDRAM bus with Direct Drive

Memory Considerations

- RAM Bandwidth drives GUI performance
- You **can** build a complete GUI with only 128KB RAM + RAM-on-LCD but your software will be challenging
- More RAM → more performance, easier software, higher cost
- Use fast-read bulk FLASH storage for images, etc.
  - Micron N25Q series is impressive!

SIM205-A00 Teardown



- USB device port (can power whole module!)
- Renesas E1-type standard debug port
- SPI,I2C,USB,CAN, UART,GPIO,DACs, …
- 8MB (64Mbit) SDRAM
- **4.3" WQVGA 480×272** TFT Resistive Touch Display
- 8MB (64Mbit) Serial FLASH
- Audio w/0.7W Speaker
- Embedded USB Host Port
- System Temp Sensor

- 100MHz Renesas RX621 MCU 512K Flash, 96K RAM
- LiPo Battery charger with Battery/Switch connector
- Battery Backed RTCC with EEPROM and SRAM

Handles Rich Graphic GUIs, including alpha blending.
Lower cost/feature options & volume pricing available.

DEVCON Enabling the Smart Society
Serious™

Serious Integrated Modules

- "Your new front panel. Done."
  - Off-the-Shelf fast prototyping
  - Off-the-Shelf cost-effective production
- Balanced LCD, MCU, memory, peripherals
- Best in class MCU technology
- Easily connect existing OEM system
- Resiliency across LCD changes

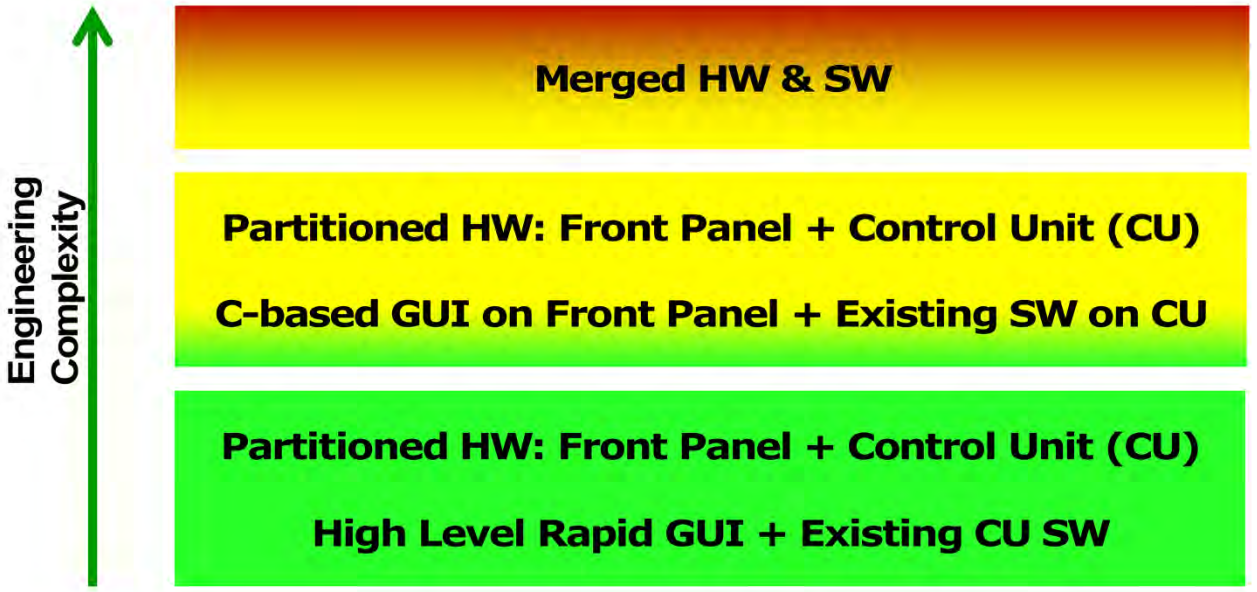
Software & GUI Design

- Storyboarding and digital assets
- SW Architecture and approach
- Renesas Graphics API (GAPI)
- Serious Human Interface™ Platform (SHIP)

Storyboarding & Managing Expectations

- Marketing teams need to be level-set up front
  - It's not an iPhone/iPad… not in BOM cost expectations
- Few marketing teams have designed GUIs
- PowerPoint/Adobe screen-by-screen *storyboard:* critical tool for agreement on GUI capabilities
  - Drives MCU, LCD,… needs and BOM cost
  - Drives software & GUI strategy
  - Drives cost and time to completion
- Rapid GUI tools can turn a storyboard into a functioning simulation very quickly

Merged or Partitioned Architecture?

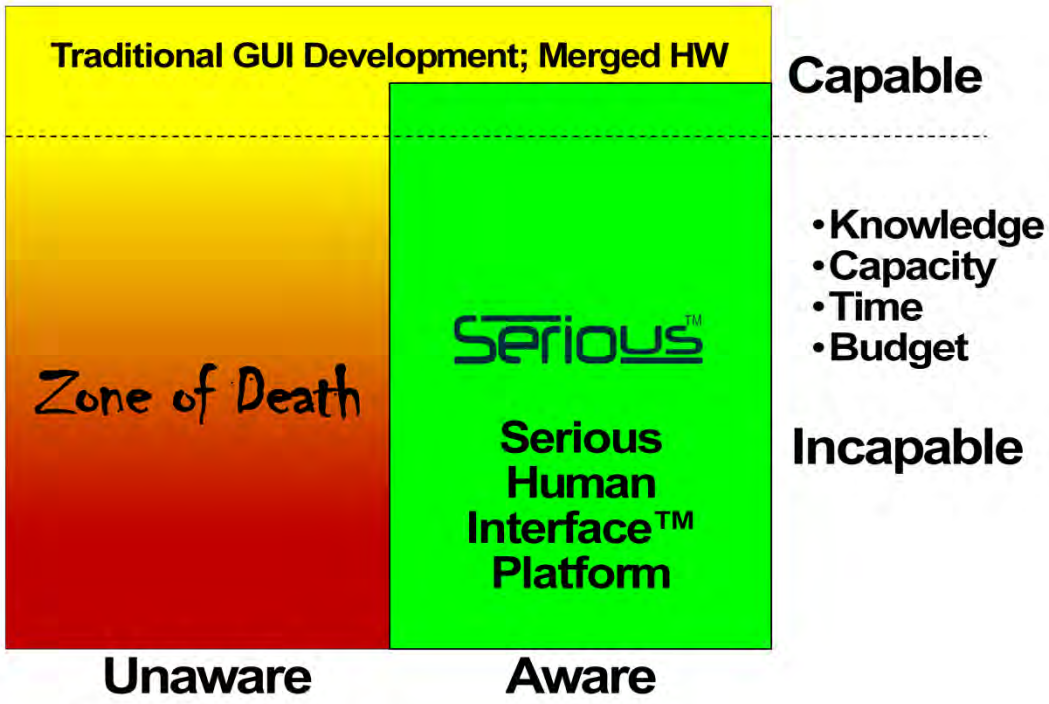


Why Partition The Problem?

- GUIs: big, complex, different than control SW
  - Most teams don't have experience/capacity
- Control SW: your product/company specific
  - Complex low level code with legacy code/architectures
  - Networking & connectivity: many options
  - OEMs generally have a long history in their control SW
- Combining GUI + Control: big custom engineering problem – initially and for on-going maintenance
- Keeping these separate: focus on differentiation and leverage the *Serious* off-the-shelf GUI system

"Nothing is particularly hard if you divide it into small jobs" – Henry Ford

Avoid the *Zone of Death!*



Merged C-Based Design: Resources

RENESAS

- AE Support
- Reference Code
- App Notes
- GAPI

Sirius™

- Pre-Ported OSs
  - FreeRTOS, Micrium, Segger,…
- No Cost Drivers
- Full BSPs
- GAPI

SW Partners

- Micrium uC/GUI
- Segger emWin

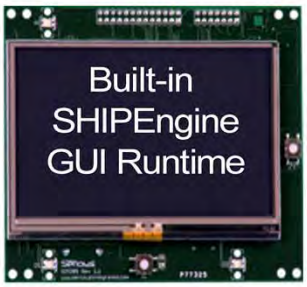
Best in Class Ecosystem for Getting Started

The Serious Human Interface™ Platform (SHIP)

Rapid GUI **Development** and **Deployment** System

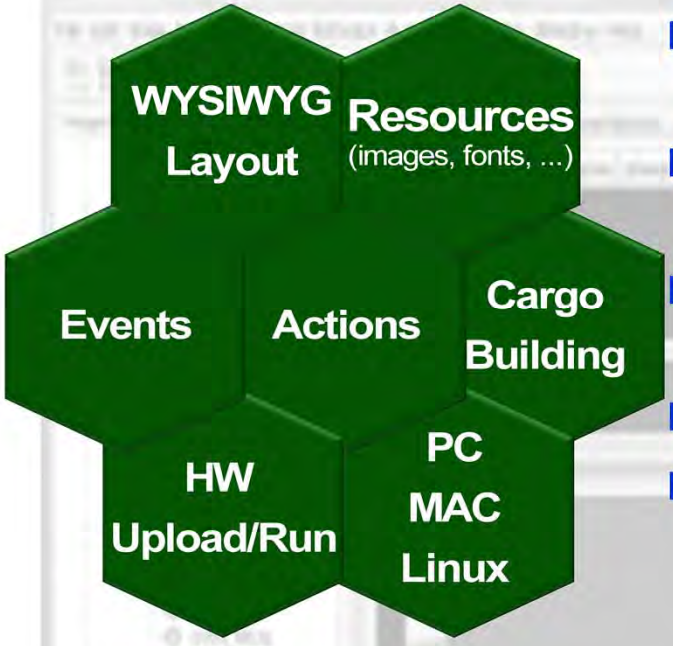
GUI Authoring Tool
The SHIP Total IDE: SHIPTIDE

Target Hardware
Serious Integrated Modules



GUI data file created by SHIPTIDE
The "cargo"

SHIPTIDE: the SHIP Total IDE



- Manages all aspects of GUI creation
- Fast modify/upload/try cycles
- Powerful yet simple events, actions, layout capabilities
- Minimal coding
- Fosters collaboration between graphics artists and GUI engineer

SHIP Deployment Example

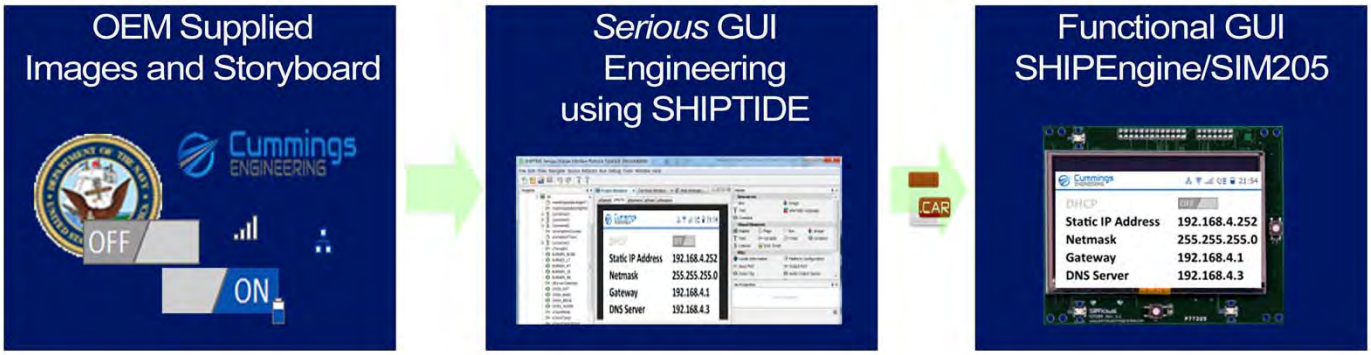
- OEM had next-day customer demo
  - A comms management front panel
- 4 interactive screens with simulated functions
  - They supplied digital media assets (gfx, storyboard,...)
- *Serious* custom services team: demo GUI in 4 hrs

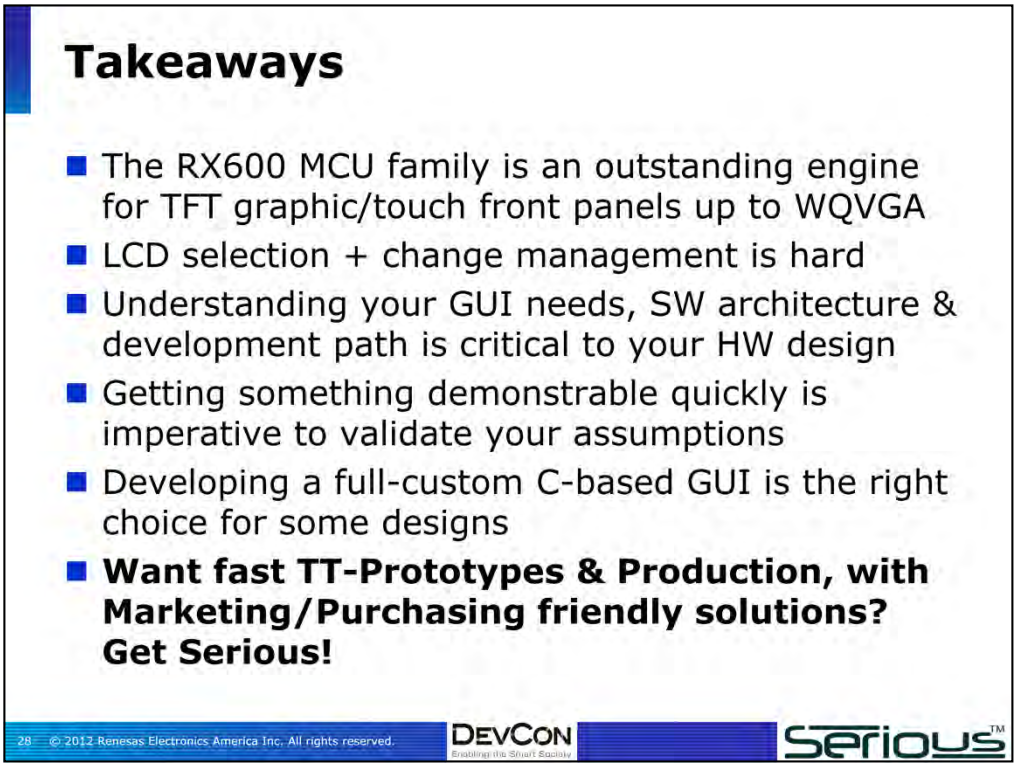
SHIP Runtime GUI Engine

- Portable & Scalable
  - Multiple platforms from128KB – 64MB RAM
  - RAM-on-Glass, Direct Drive, and full Graphics Controllers
- Optimized
  - Tuned to each CPU/Graphics architecture for the best GUI performance
- Robust
  - Significant run-time code checking
  - SAIL virtual machine puts customer "code" in safe sandbox

SHIP Summary

- Extremely rapid GUIs in days, not years
- Modern looking front panels – minimal coding
- Available with on all Serious Integrated Modules
- Prototype ready, Production Worthy





By the end of this session you will be able to:

•Know about market trends in HMI technologies

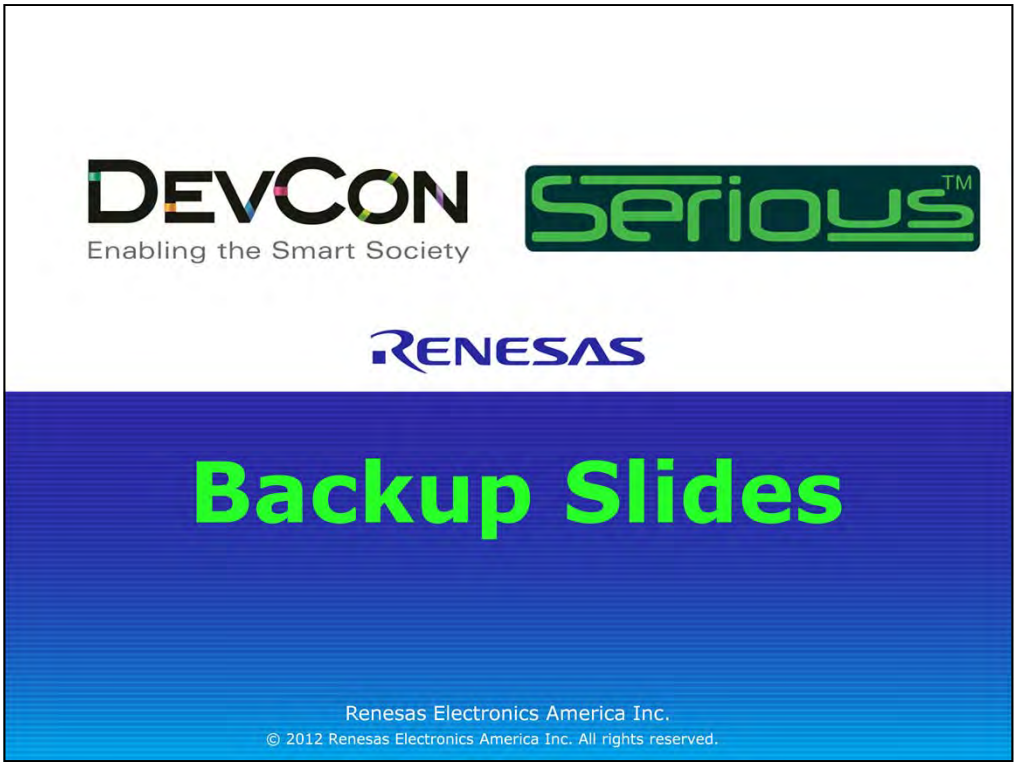
•Understand types of capacitive touch solutions available

•Identify pros and cons of hardware- and software-based solutions

•Recognize the benefits of Renesas' touch solution

Need More Info?

- Visit our corporate website: seriousintegrated.com
- Visit our community website: mySerious.com
- Contact your local Arrow Electronics FAE
- Contact your local Serious Manufacturer's Rep
  - see website for list
- Contact us directly: support@mySerious.com


DEVCON
Enabling the Smart Society
RENESAS
Backup Slides
Renesas Electronics America Inc.
© 2012 Renesas Electronics America Inc. All rights reserved.

LCD Quality

- All assemblies are not equal
- Infant mortality, MTBF, etc. highly dependent on quality assembly
- Wide variation in quality processes
- Some LCD vendors are primarily brokers

Know Your Manufacturers' Supply Chains!

Brightness/Luminance

- Measured in cd/m^2 aka **NITS**
- Lose up to 80 NITS with a touch layer
- Your eyes aren't linear: small screens need smaller NITS to look similarly bright

NITS	Usage Model	Cost
<100	Night, Dark Area	$
100-300	Indoor	$
300-500	Bright Lit Indoor Indirect Outdoor	$$
600-800	Normal Outdoor	$$$
800-1000	Strong sunlight readable	$$$$$

Interoperability

- Some LCD vendors have compatible models
- Some supply line resilience

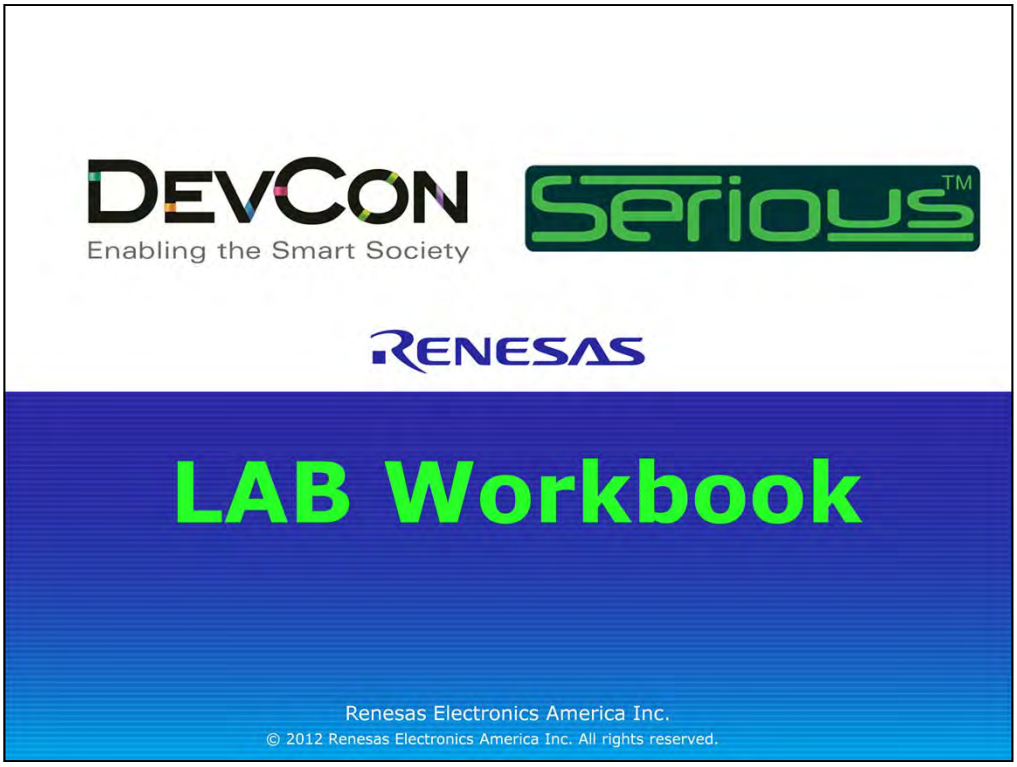
HOWEVER…

- Often same base glass supplier
- Same chip supplier
- Sometimes even same actual module supplier

Plan for LCD Refreshes in HW, SW, and Purchasing

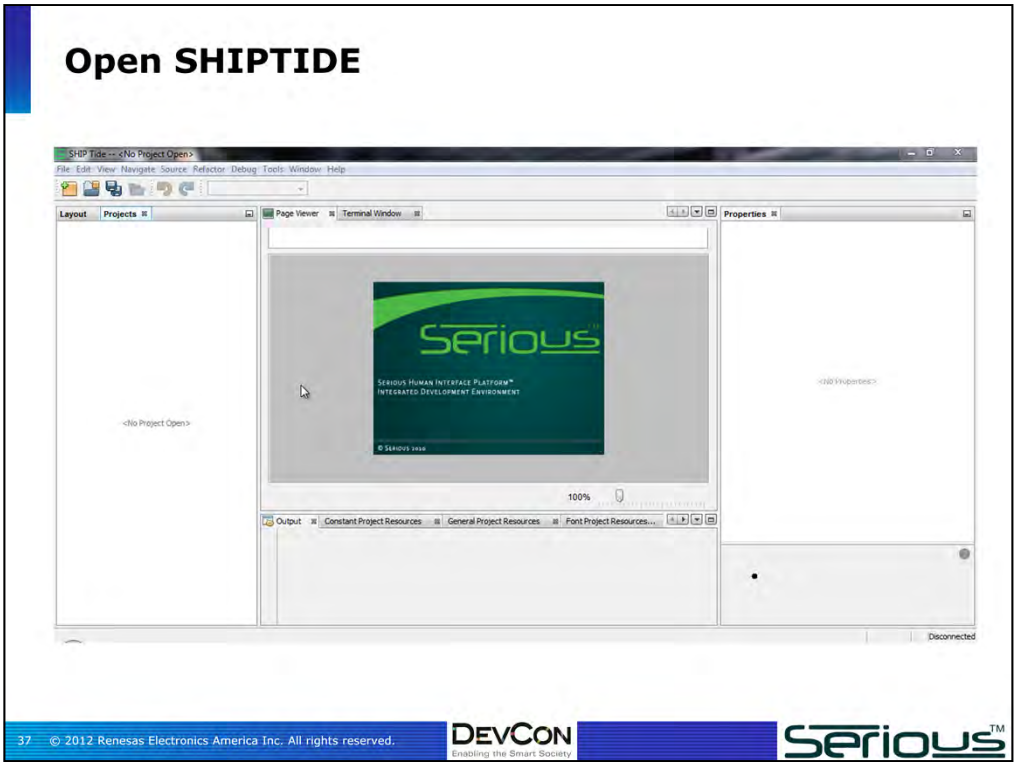
Operating Range and MTBF

- 0-60C/70C common
- -20 to +60/70C on some models
- Below -20C is problematic
  - LCD's crack, layers separate, ...
  - Need a warming device
  - Many (most) heaters are unreliable
  - Look for field-proven solutions
- LED Backlight solutions vary widely
  - Check the backlight typical hours of operation
  - Use backlight-off/PWM to extend lifetime


DEVCON
Enabling the Smart Society
RENESAS
LAB Workbook
Renesas Electronics America Inc.
© 2012 Renesas Electronics America Inc. All rights reserved.

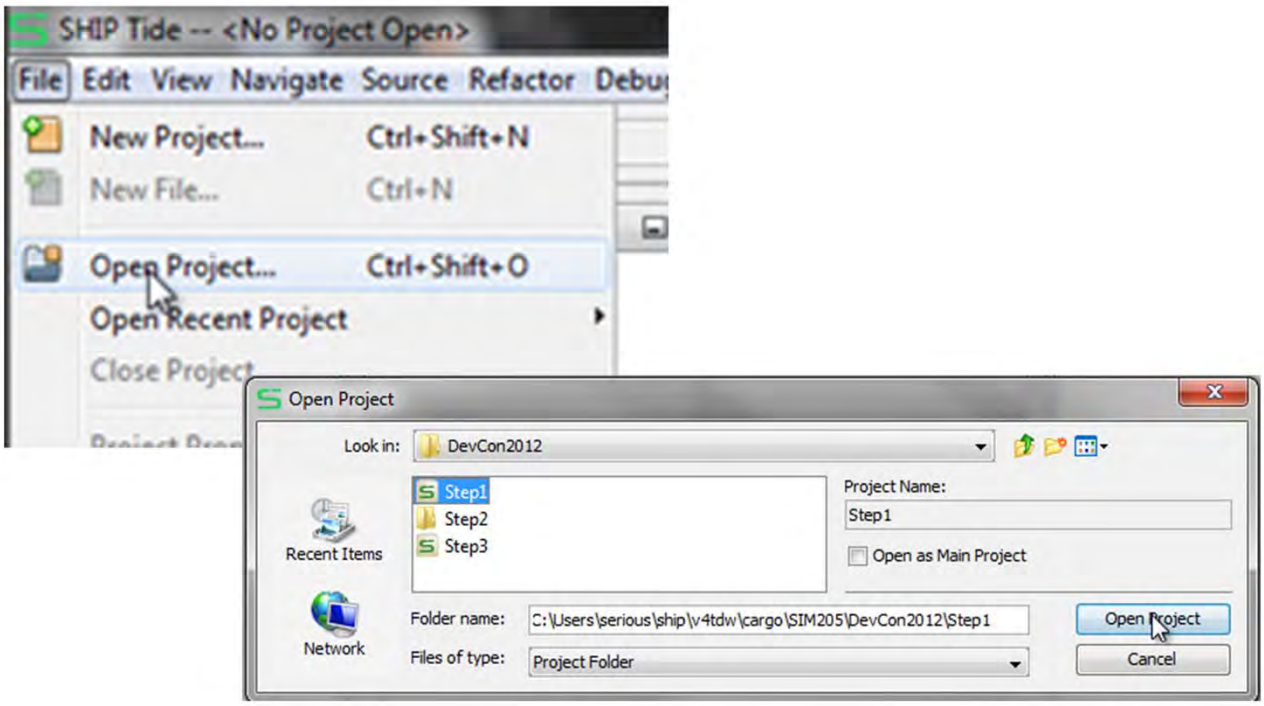
Lab Workbook

- Step 1: Basic Operations – 20 minutes
- Step 2: Adding Actions and Audio – 20 minutes
- Step 3: Advanced Features – 20 minutes



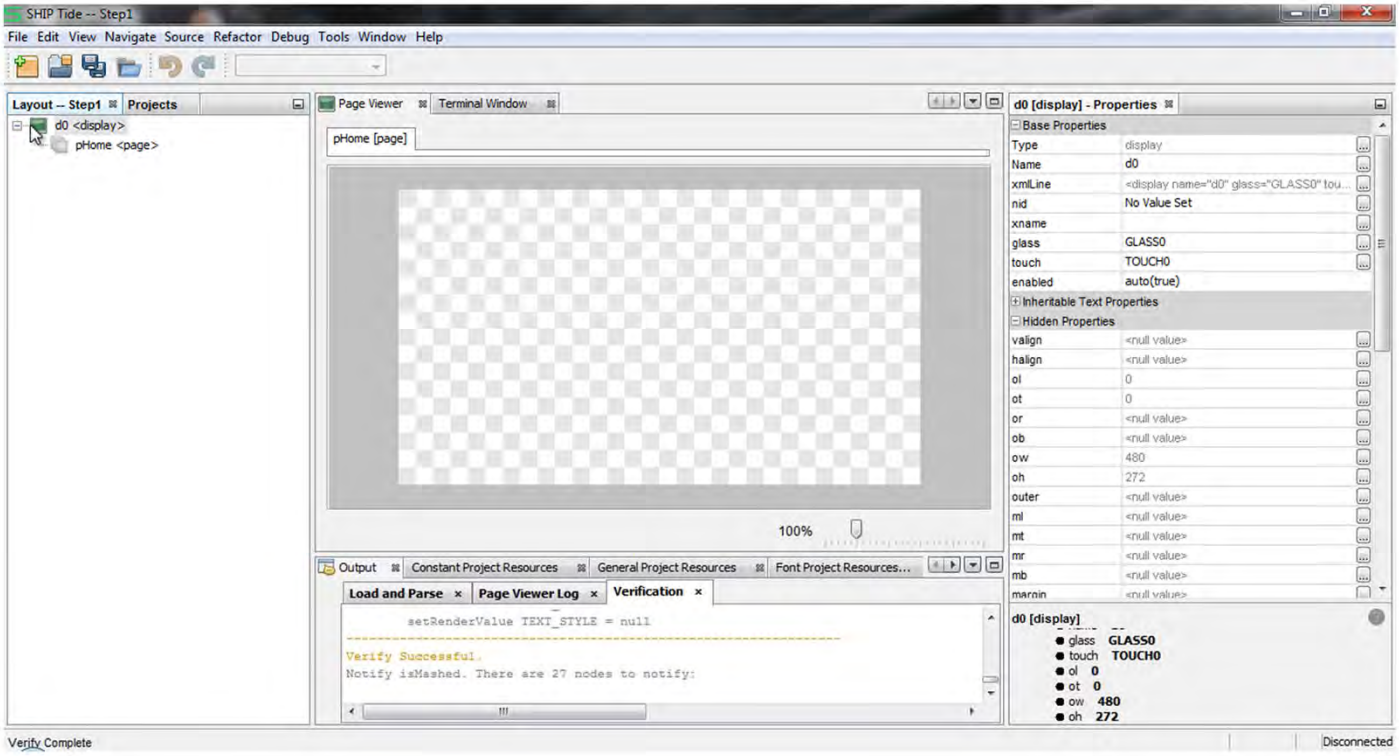
Start up the SHIPTIDE application. No project will be loaded, and it will look like this.

Open a Project



Using File->Open Project…. Open the project “Step1”. SHIPTIDE will open and read the project, including reading any assets (images, etc.) associated with the project.

SHIPTIDE Basic Elements



The Step1 project is a pretty bare-bones project: you're going to have to add some content and more to make this a real GUI.

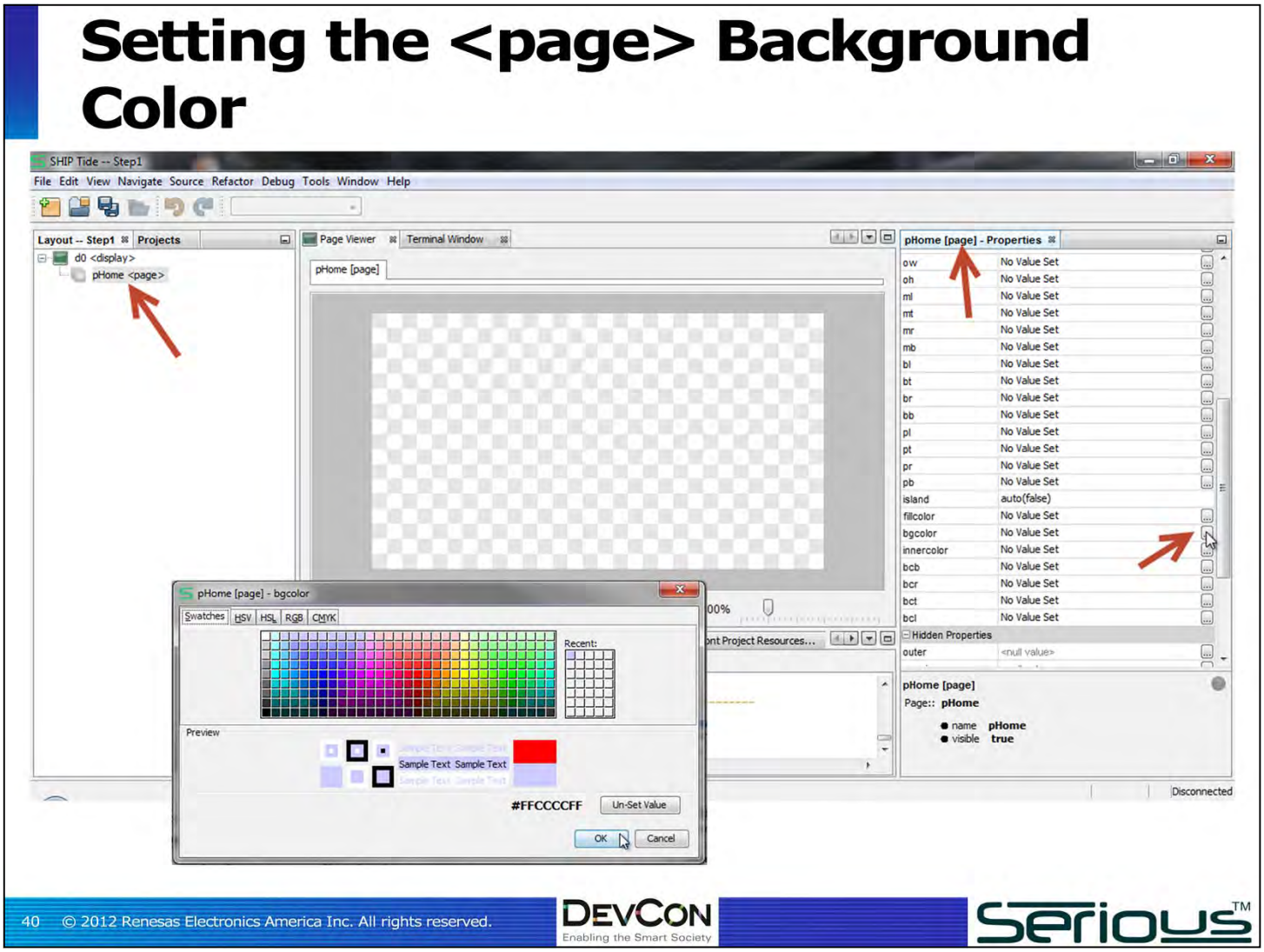
In the left "layout" area you see a tree-structured view of the layout of the GUI.

At the top of the layout is a single <display>, since the target module SIM205 only has 1 LCD display. Under the display is a single GUI <page> named "pHome". Every layout node can have a name that you supply – handy for remembering what's what in the layout.

In the middle of SHIPTIDE is the WYSIWYG area of the GUI authoring tool and shows only the one empty <page>. The checkerboard means the background is transparent.

On the right is the property sheet, listing the various properties of the selected layout node. With the <display> node selected as shown, the property sheet shows various properties of the <display> node, including the width and height of the display.

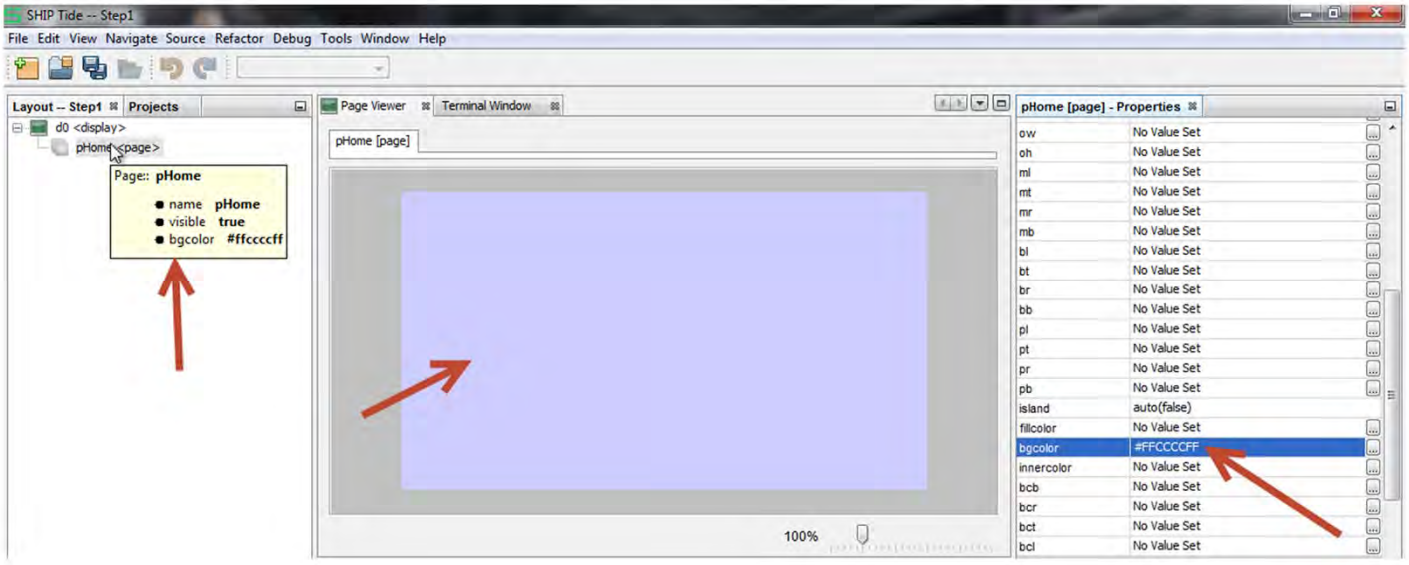
Setting the <page> Background Color



To set the background color of the <page> to something else, click on the <page> node in the layout area, then in the property sheet find the “bgcolor” property and click the “…” button to bring up a color picker.

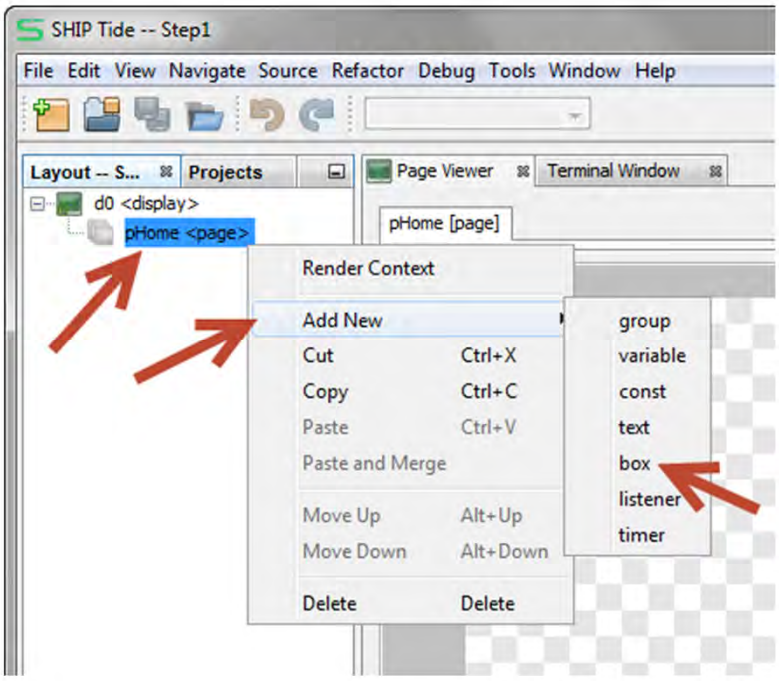
Let's choose an off-white color.

Background Color Now Set



The background color of the <page> named pHome is now set, and you see the change in the WYSIWYG viewing area. Note how hovering over the pHome node in the layout tree shows the important properties set on the node.

Adding A Logo (part 1 – adding a <box>)



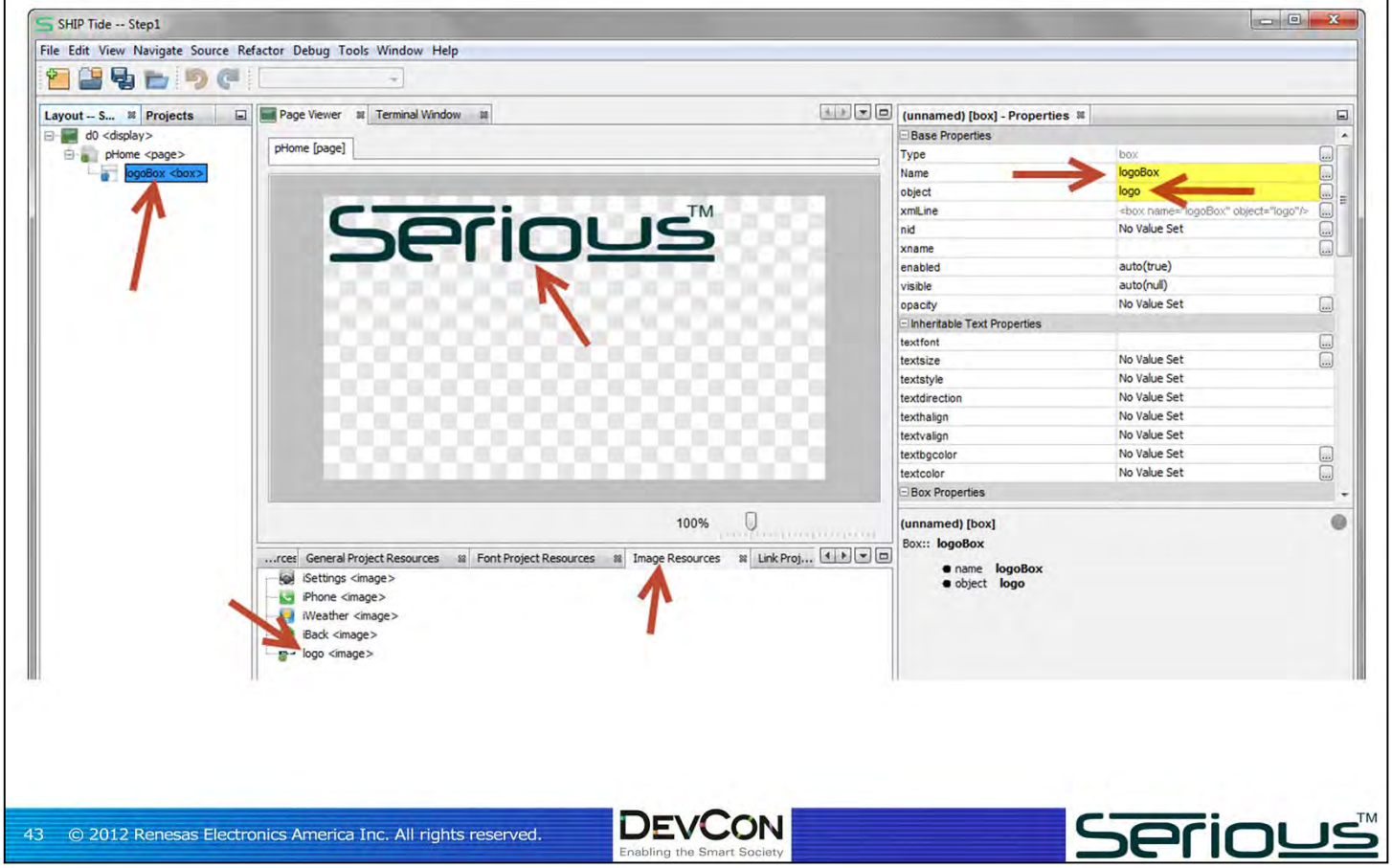
To make it a little more interesting, we need to add our logo right in the middle of the **<page>.** Images live in **<box>s**, so right-click on **pHome** and select **Add New→box**.

Using the property sheet (ensuring the new **<box>** is selected) give the box a name, say "**logoBox**".

Boxes are containers that can be stacked, nested, and so on. They are very versatile and have many properties.

When you complete this addition, you won't see much change in the WYSIWYG viewing area. By default, the <box> is completely transparent, with no borders, and inherits the size of the parent (in this case, the <page>).

Adding a Logo (Part 2 – attaching the image)

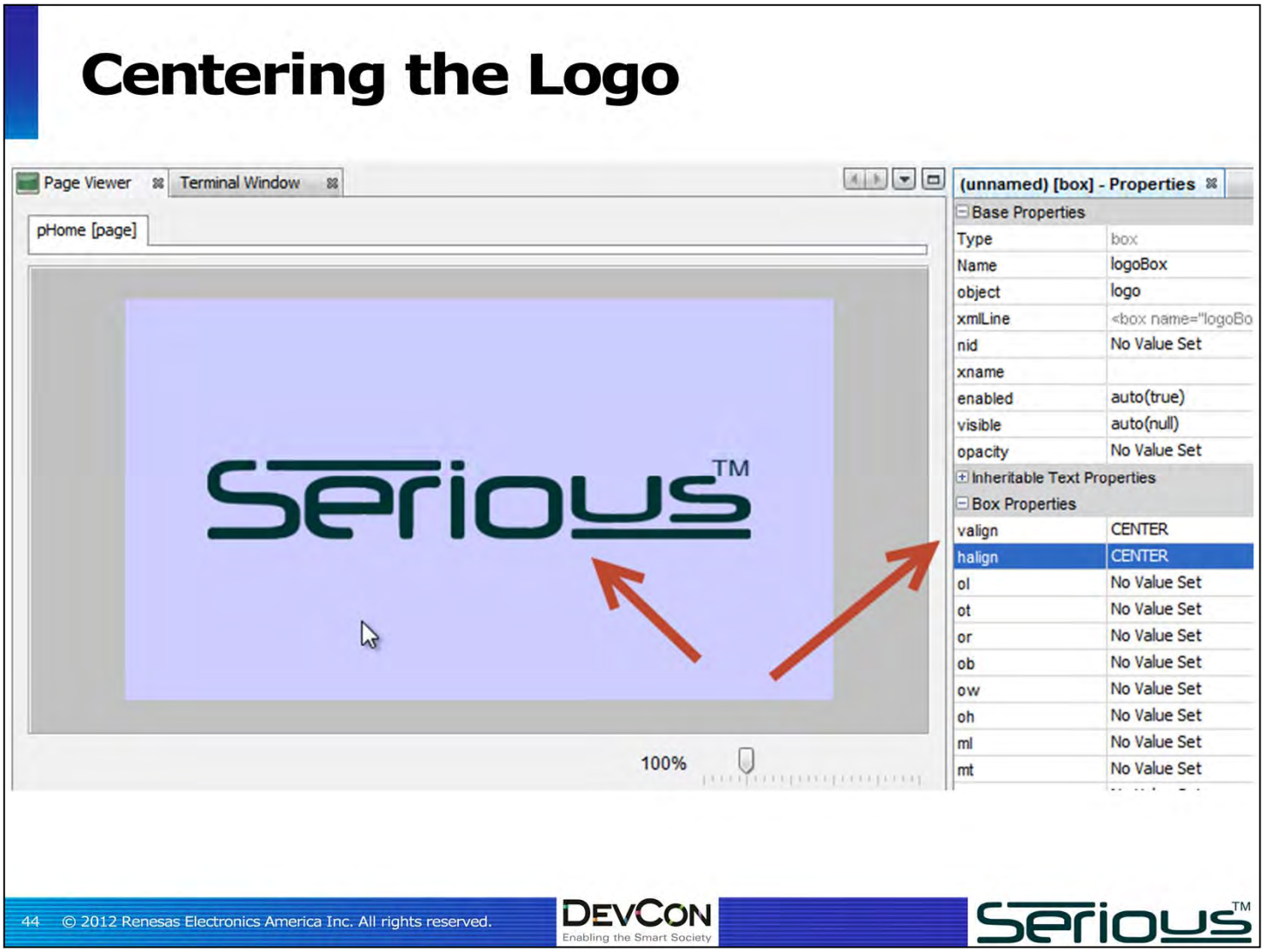


Now we need to attach our logo image to this **<box>**.

Find the image named "**logo**" in the Image Resources panel. It's a 380x72 pixel PNG file with a transparent background. We've preloaded a number of images into this project to get you started.

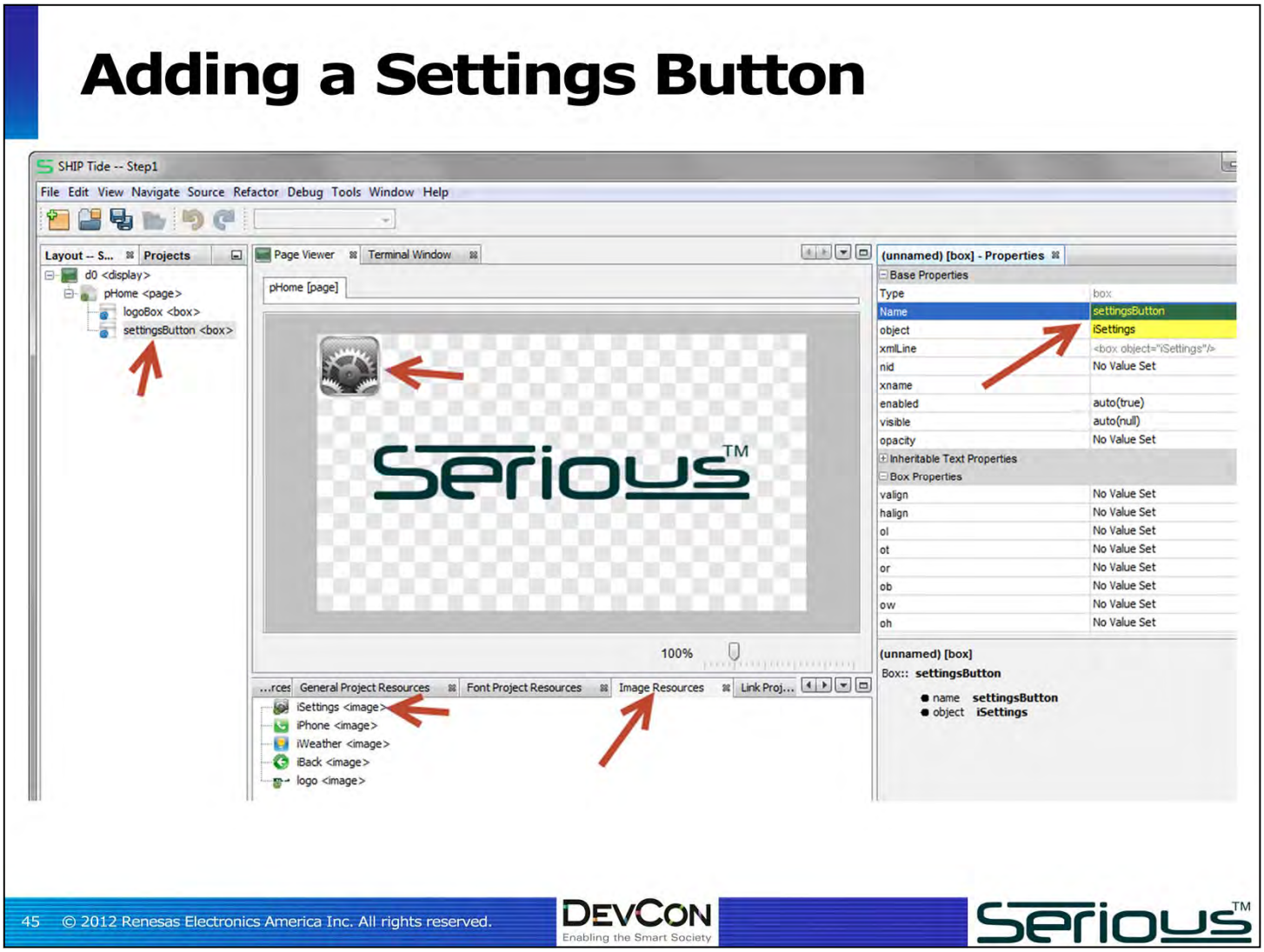
To add this image to the box, either (a) change the "object" property in the **<box>**'s sheet to "**logo**" (the name of the logo image resource), or (b) drag the logo onto the **<box>.**

The enclosing box (named **logoBox**) now has the image attached. Attaching the image provides width and height information to the **<box>,** so **logoBox** is now 380x72, and by default positioned at the upper right of the parent container (the **<page>** named **pHome**).



To center the logo, we need to position its enclosing **<box>** (**logoBox**). Ensure the **logoBox** node is selected in the layout area, and the property sheet will become visible. Change the **halign** and **valign** properties to **CENTER**.

Adding a Settings Button

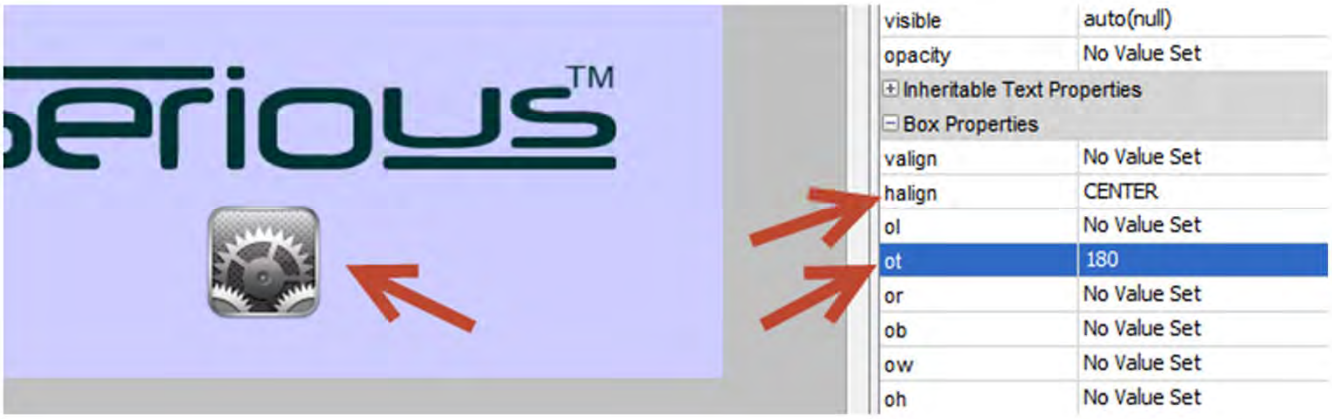


Let's add a button below the logo. A button is any **<box>**, with or without an image attached, with extra "stuff" attached to the box. We'll get to the "stuff' in a minute. But for now, we want a "settings" button positioned below the logo.

As before, select the parent container for this new **<box>** (in this case, **pHome**) and **right-click/Add New→box**. Give the box a name (say, **settingsButton**), and drag the **iSettings** image over into the new **<box>** (or enter the **iSettings** name into the "**object**" property of the **<box>**).

The button appears, and as normal it is positioned at the upper left of the parent container (the **<page>**).

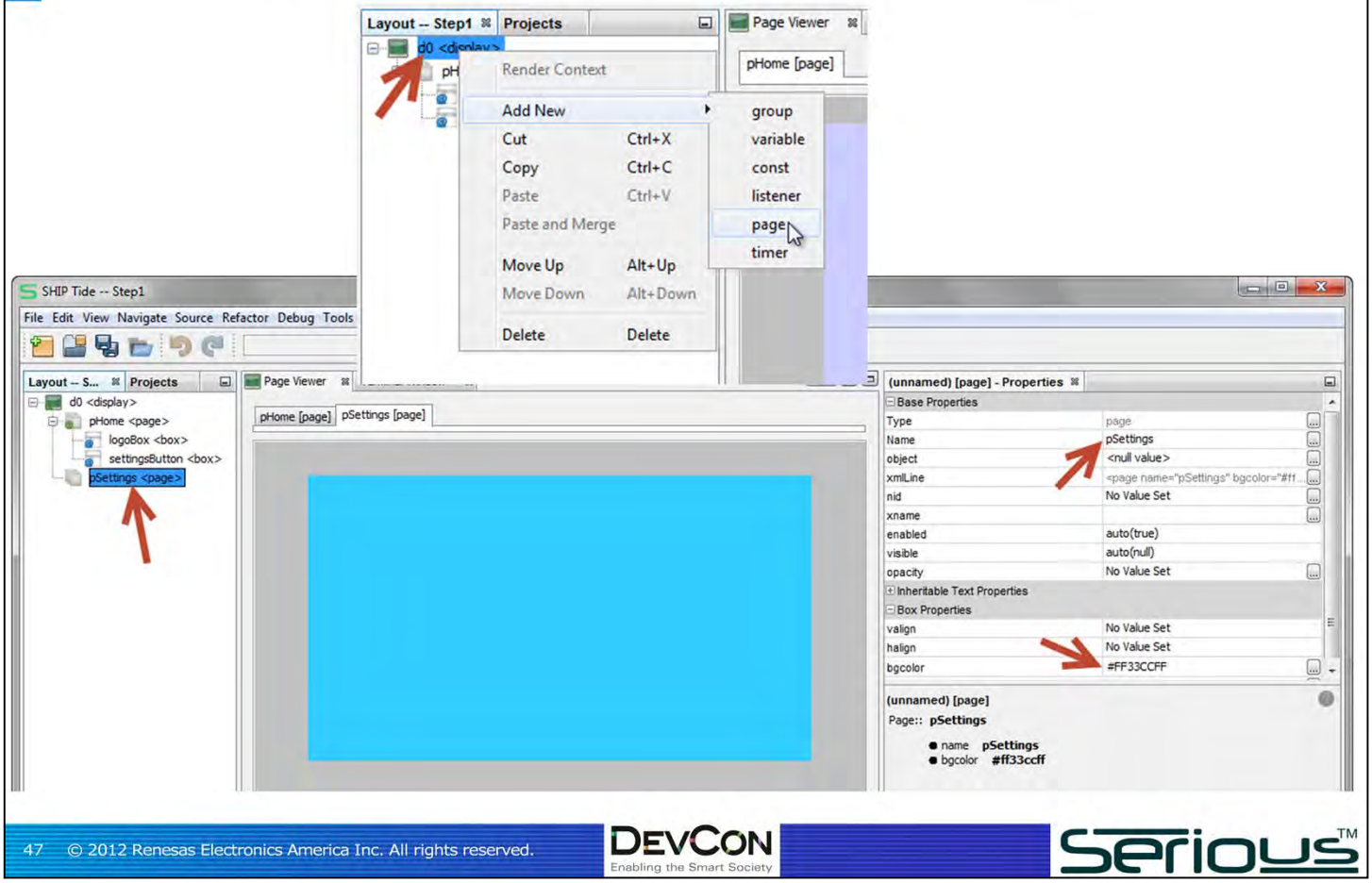
Positioning the Button



Changing the **settingsBox** properties can position it below the logo. **halign CENTER** will center the **settingsBox** inside the **<page>** horizontally.

To position it vertically there are many different methods but we'll choose the obvious one: setting the outer top (**ot**) property of the box to **180**. This is easy, but not the best way…. we'll show you a better way later.

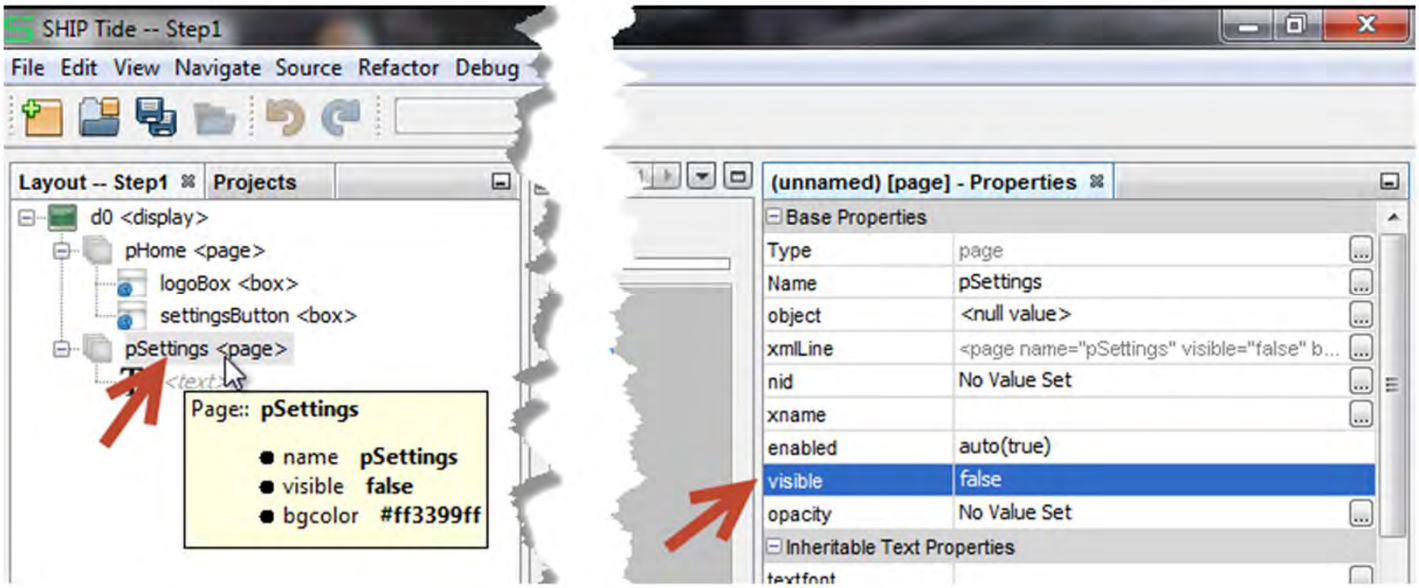
Adding the New Settings Page



We need to build an empty “settings” page that the button will go to.

First select the **d0 <display>** node in the Layout area --- the parent of the new **<page>** – and **right-click/Add New→page**. The new page will appear. Change it’s name to **pSettings** and it’s background (**bgcolor**) to something interesting – say a robin egg blue.

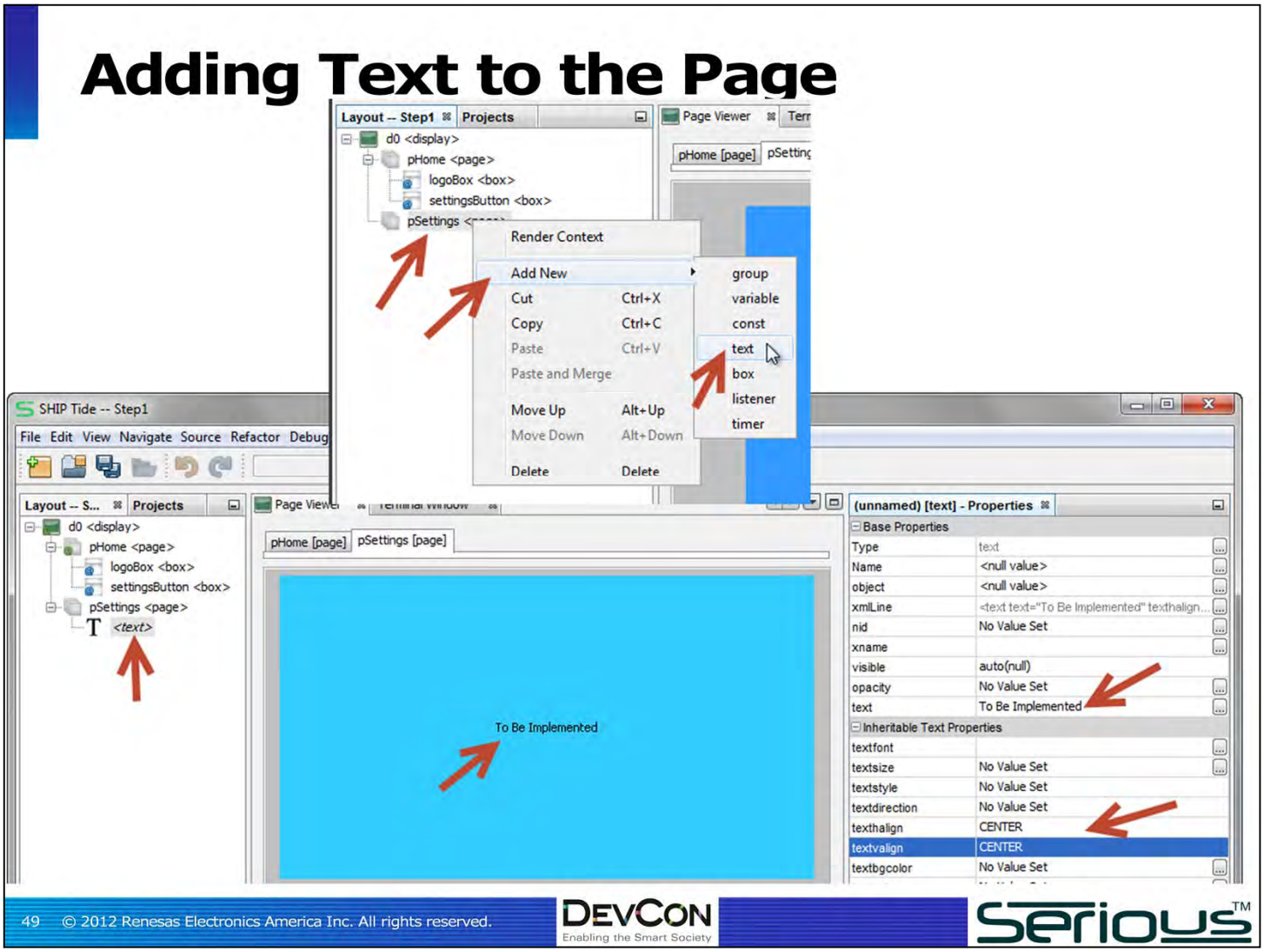
Ensuring the Right Pages are Visible at Boot



We only want 1 **<page>** visible at boot time – the **pHome <page>**.

Ensure that the **visible** property of **pSettings** is **false**. If you leave it **true**, both **pHome** and **pSettings** will display, and since **pSettings** comes after **pHome**, all you will see is **pSettings**.

We'll show you how to make **pSettings** visible and **pHome** invisible when the button is pressed in a moment.



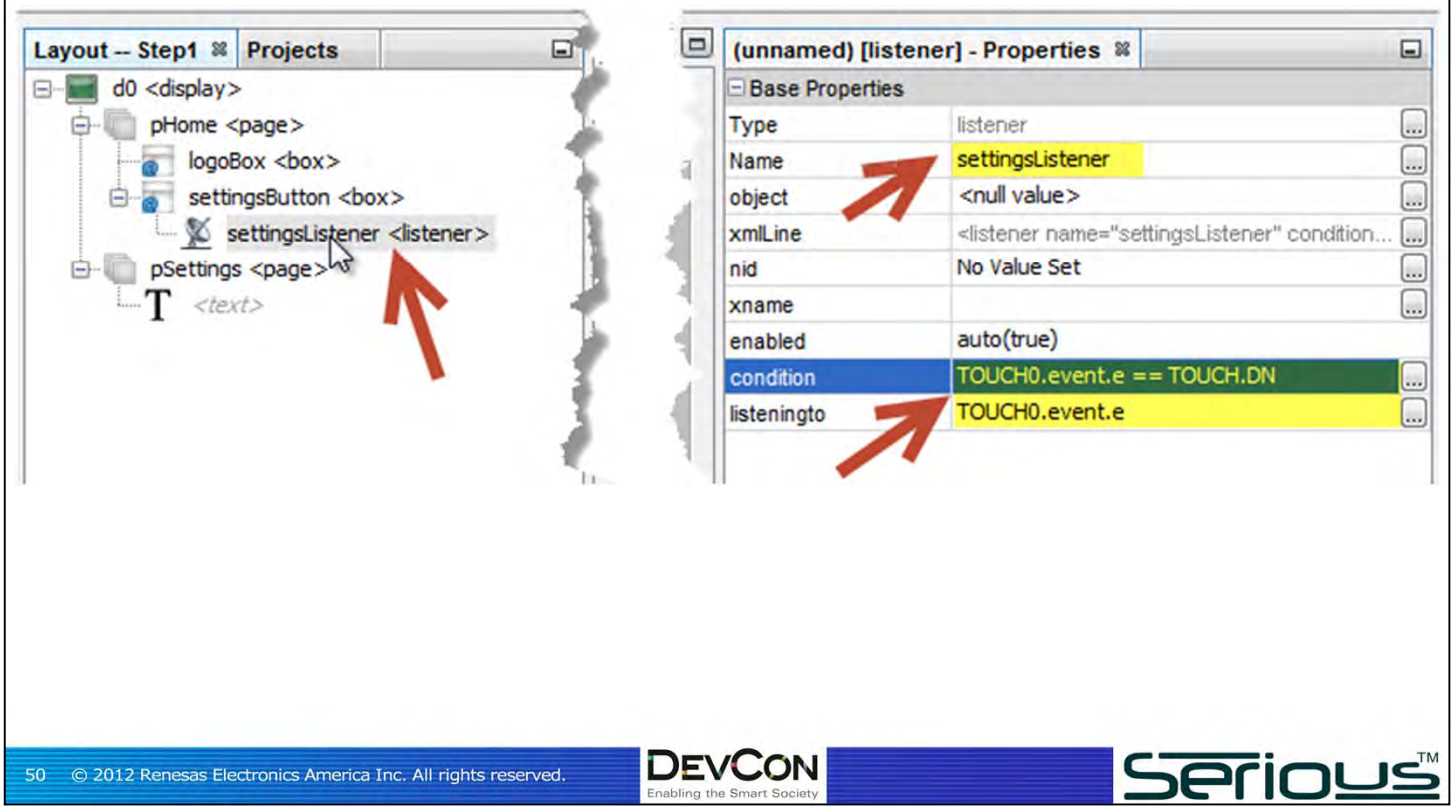
Let's put the words "**To Be Implemented**" in the middle of the **pSettings <page>.**

Select the **pSettings <page>**, and **right-click/Add New→text**.

In the property sheet for the **<text>** add the words "**To Be Implemented**" to the "**text**" property, and use the **texthalign** and **textvalign** properties to center the text inside the parent container (the **<page>**).

<text> nodes are not as heavy as **<box>** nodes: they don't have all the positioning elements and sizing elements that a **<box>** does. If you need more precise positioning of **<text>**, put it in a **<box>** and position the **<box>.**

Making the Button Listen to the Touch Screen



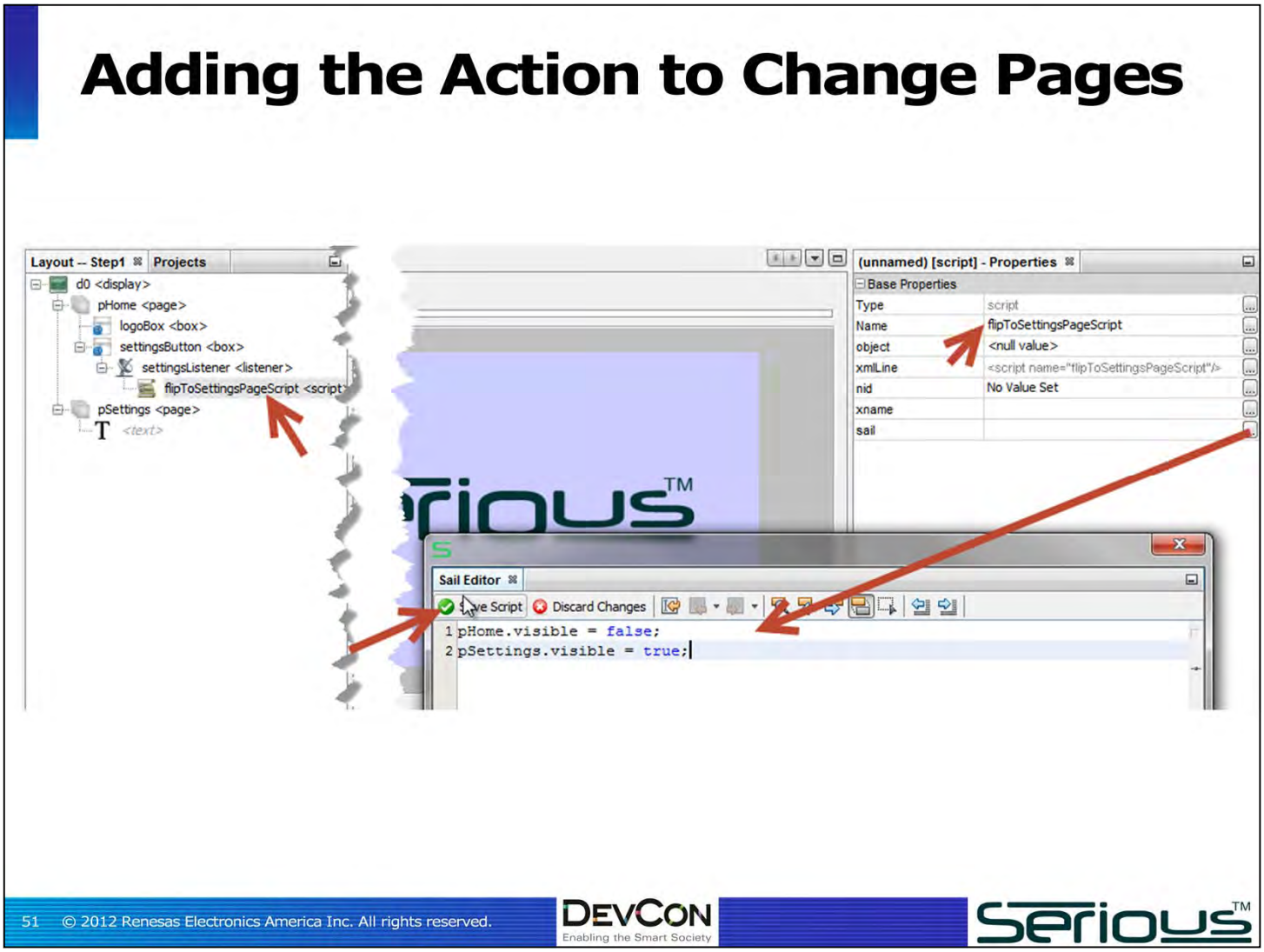
The **settingsButton** needs to somehow, when pressed, transition the GUI to the **pSettings <page>**. There are two parts to this mechanism. The first is the **<listener>** that listens to the touch screen for the correct event. The second is the action that happens when that **<listener>** detects the event.

To add the **<listener>** to the **settingsButton**, select the **settingsButton** in the Layout area and **right-click/Add New→listener**. Give the **<listener>** a name (e.g. **settingsListener**).

settingsListener needs to monitor (listen to) the touch screen. On this platform this event is **TOUCH0.event.e**. This is documented for the platform, and normally you'd just look this value up in the documentation.

TOUCH0.event.e changes on any touch up/down on the touch screen *inside the containing <box>*.

So add **TOUCH0.event.e** to the "**listeningto**" property of the **<listener>**. Further, add a **condition** filter to only listen for touch down events – not touch up events.



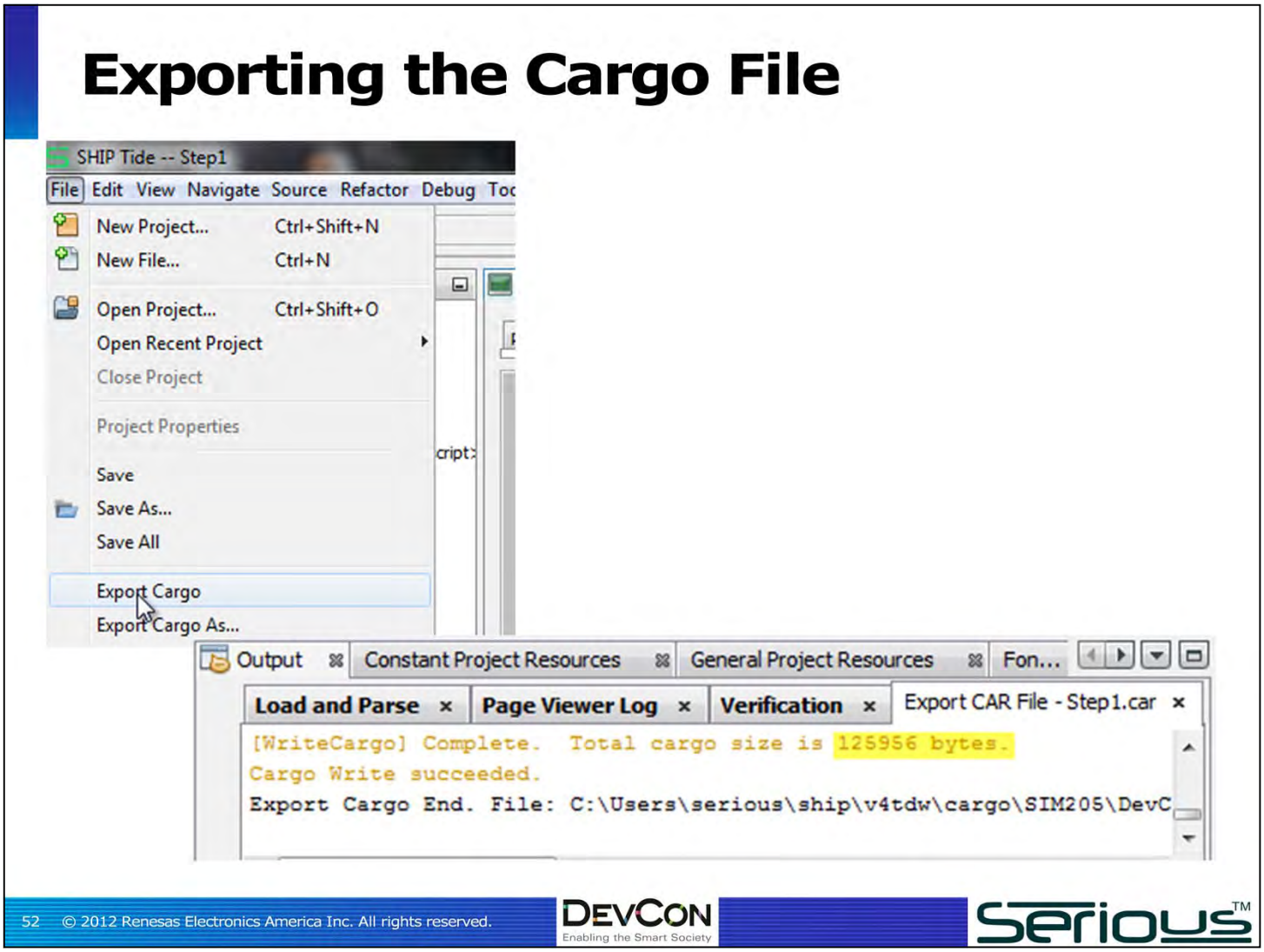
The second part of the button wiring is the action. When the **<listener>** "wakes up" and the **condition** passes, it can invoke a **<script>** that can perform an action.

Add a new **<script>** under the listener. Give a name (e.g. **flipToSettingsPageScript**).

Now edit the **sail** property. SAIL is the Serious Advanced Interpreted Language, a very simple but powerful scripting language that looks a lot like C or Java. If you've programmed at all in a mainstream language you'll find SAIL pretty straightforward.

When the button is pressed, we want the **pHome <page>** to become invisible and the **pSettings <page>** to become visible. So enter in the SAIL editor the two lines as shown: the first making **pHome** invisible, and the second making the **pSettings <page>** visible. That's it.

There are more sophisticated ways to manage <page> visibility outside the scope of this workshop: in more advanced GUIs a global page state variable is used to manage which page is currently displayed.

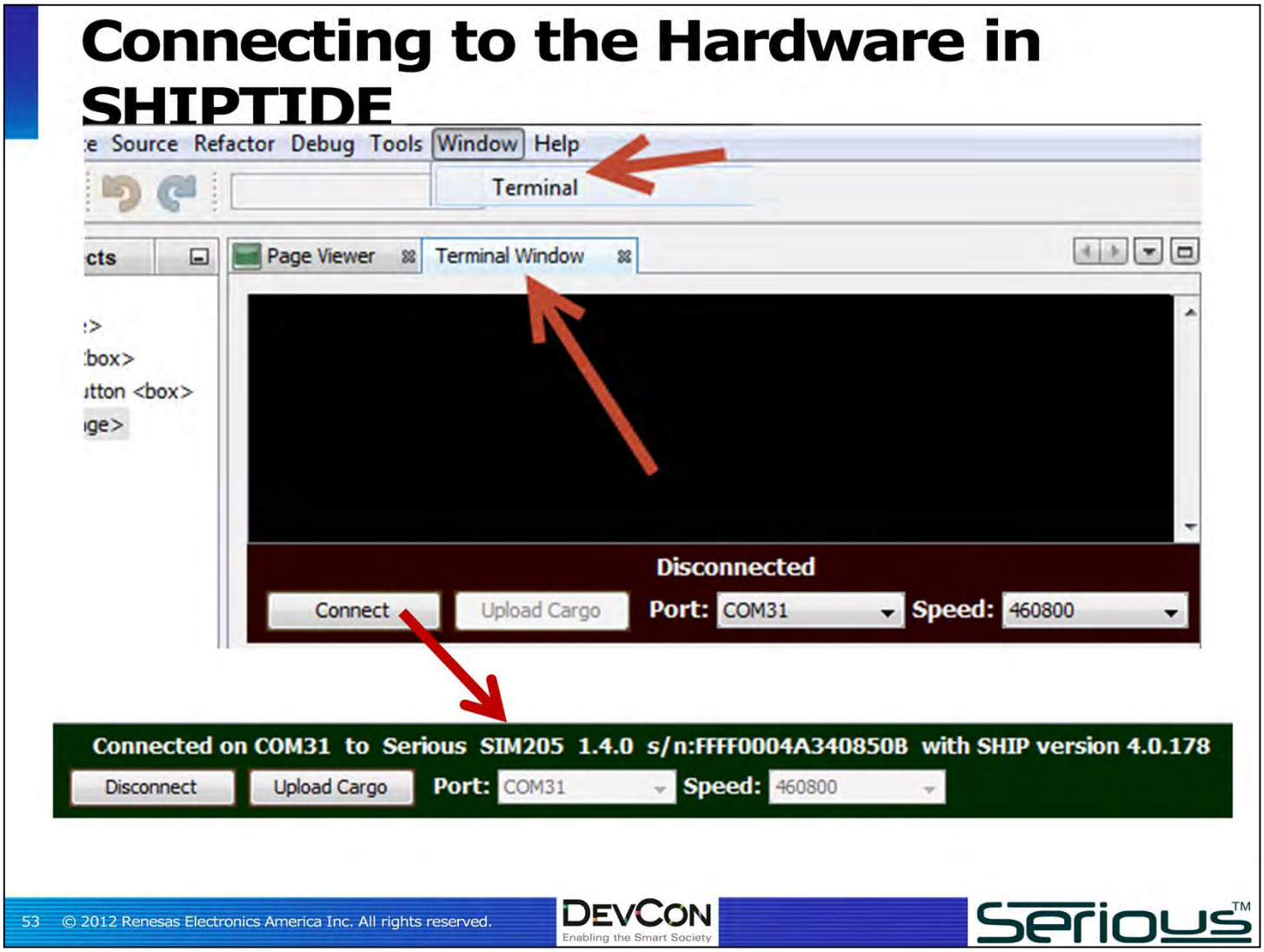


It's time to try this GUI out on the actual hardware! There are 3 parts to this process:

a) Exporting (assembling) the cargo file that contains all the aspects of the GUI you've created and
b) Ensuring you're connected to the hardware
c) Uploading this cargo file to the hardware.

To export (assemble) the cargo, under the File main menu in SHIPTIDE select "Export Cargo".

If you have the Output window visible, you'll see the cargo export completion, including how big the cargo file is.

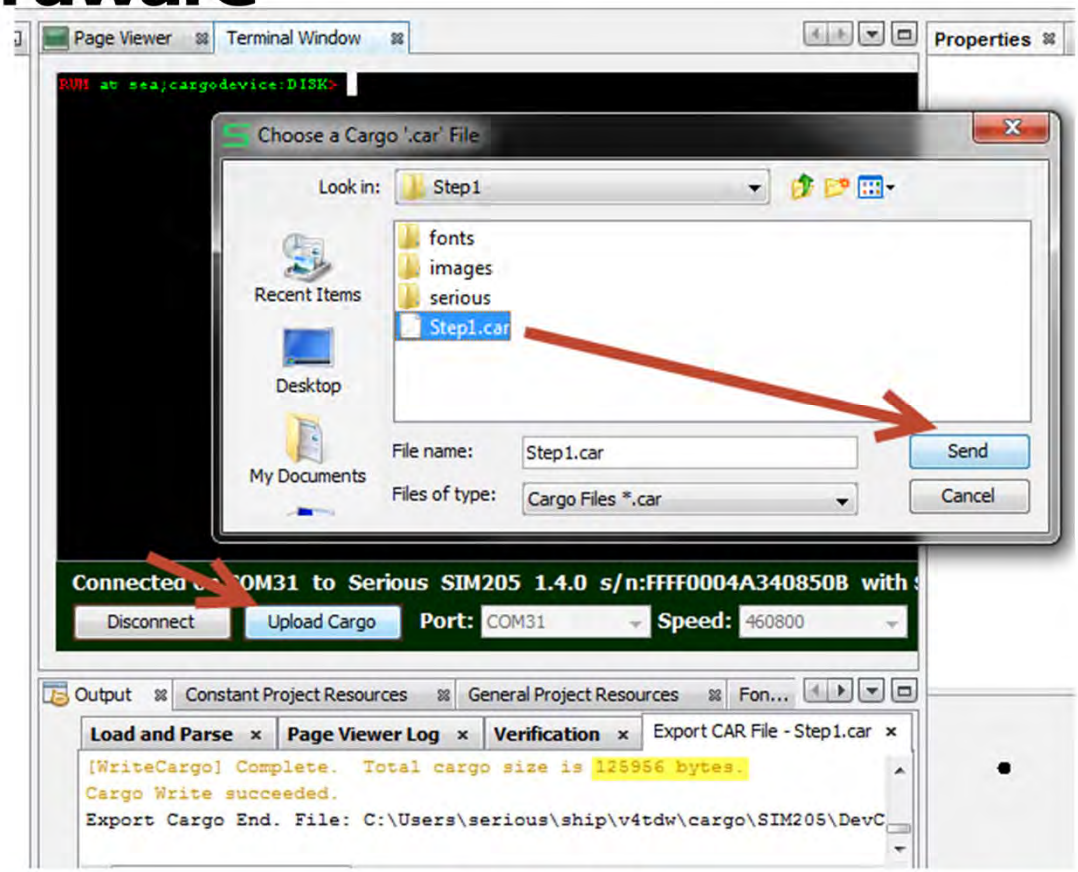


Ensure the SIM205 hardware is connected to your PC with a USB cable. If you have not already installed the driver, consult the documentation on mySerious.com and install it now. The SIM205 will appear on your PC as a COM port if the driver is correctly installed and the SIM205 is powered on and connected.

Within SHIPTIDE there is a built-in connectivity window. Make sure it is displayed (main menu item Window->Terminal).

Select the Port that the SIM205 is connected to and press the Connect button. You should see SHIPTIDE connect to the unit and display the type of unit and its serial number.

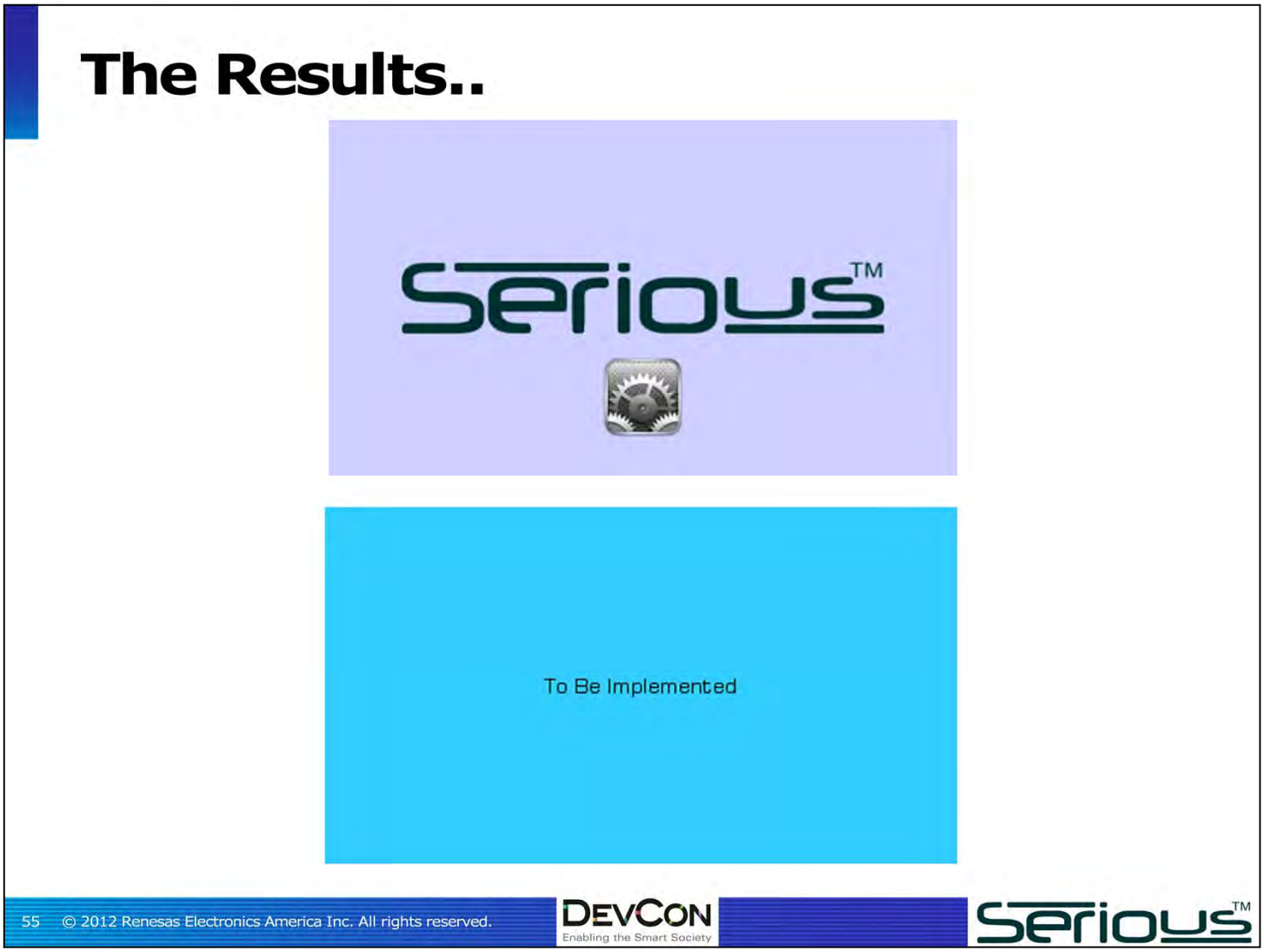
Uploading the Cargo to the SIM205 hardware



Select “Upload Cargo” and choose “Step1.car” – the cargo file you just exported.

SHIPTIDE will send the file to the Serious Integrated Module over the USB port. The cargo is stored in bulk FLASH storage on the SIM -- Serial, NAND, or eMMC FLASH depending on the module. The SIM205 has 8 megabytes of serial FLASH for this purpose.

When complete, the SIM will automatically reboot and run your new GUI.

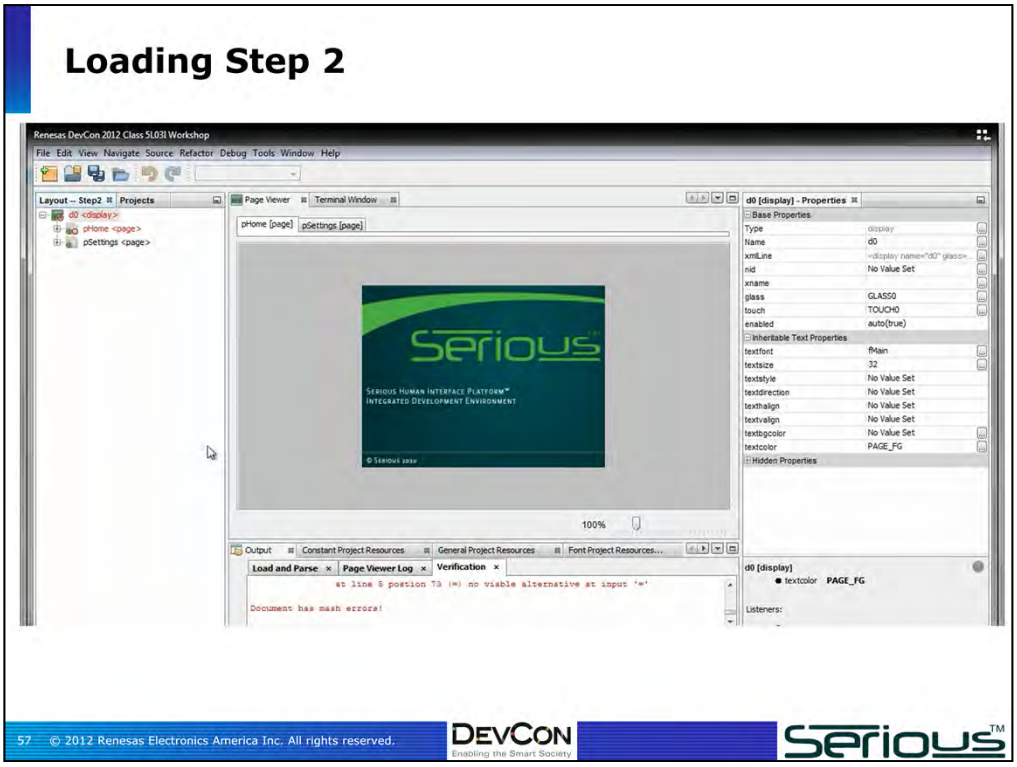


And, when the unit reboots, you'll see **pHome** displayed. Pressing the button down (and only inside the button area) will transition to the **pSettings <page>**.

This step of the workshop is now complete!

Lab Workbook

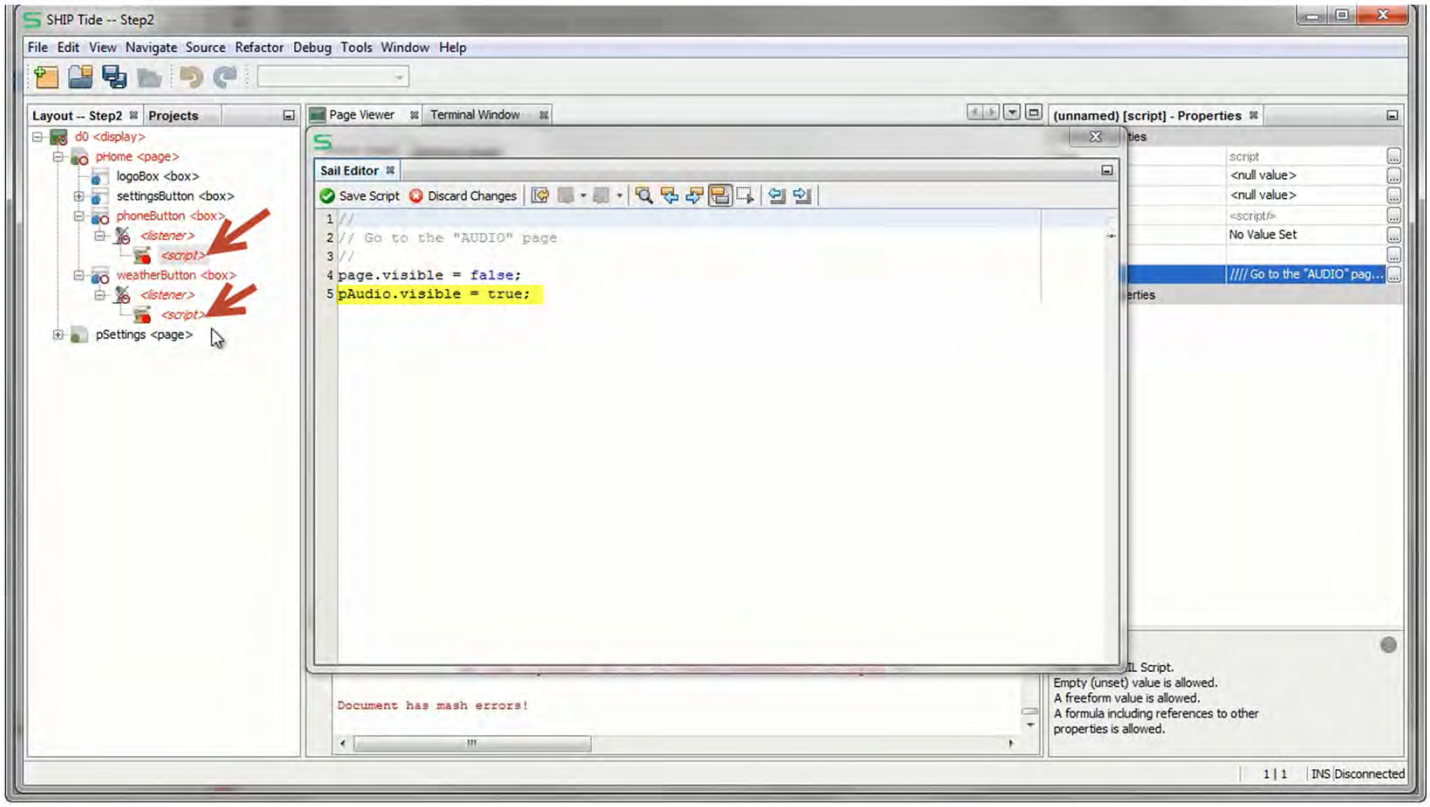
- Step 1: Basic Operations – 20 minutes
- Step 2: Adding Actions and Audio – 20 minutes
- Step 3: Advanced Features – 20 minutes



In this step will pick up where we left off at the end of step one and one walk you through a little more sophisticated layout techniques when flush out our infrastructure for GUI just bit more.

Open up project “step2”. It’s a bit more than where we left off, but it has some errors in it. Notice the red nodes. The hollow circle says there is an error inside the node somewhere.

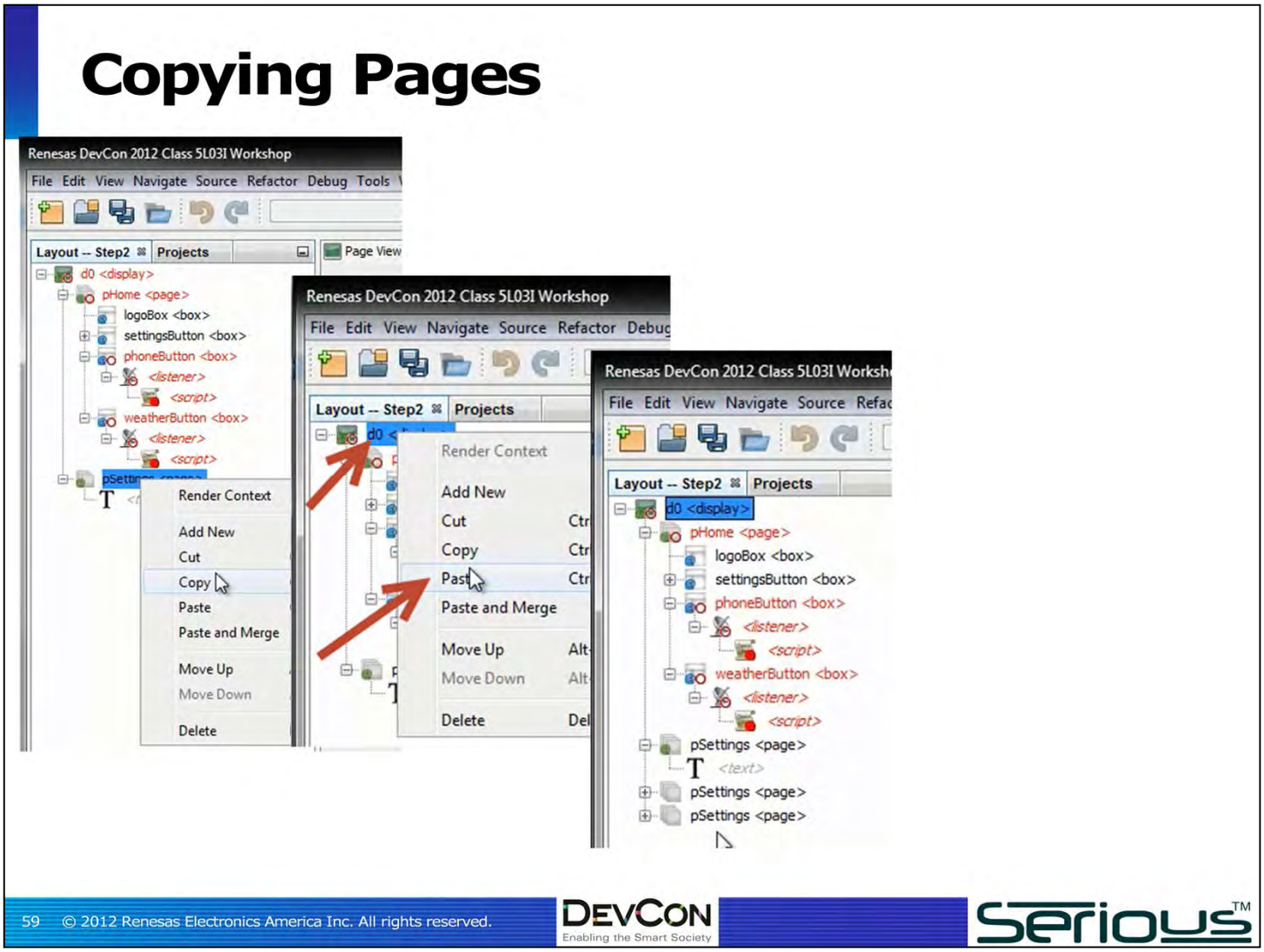
Finding and Fixing the Error



Notice we had **logoBox** and **settingsButton** before but now we've also got a **phoneButton** and a **weatherButton** – two more buttons – and both of these new buttons have errors in them. Drill down on them and keep opening and until you see the solid red dot – that tells us where the errors actually are.

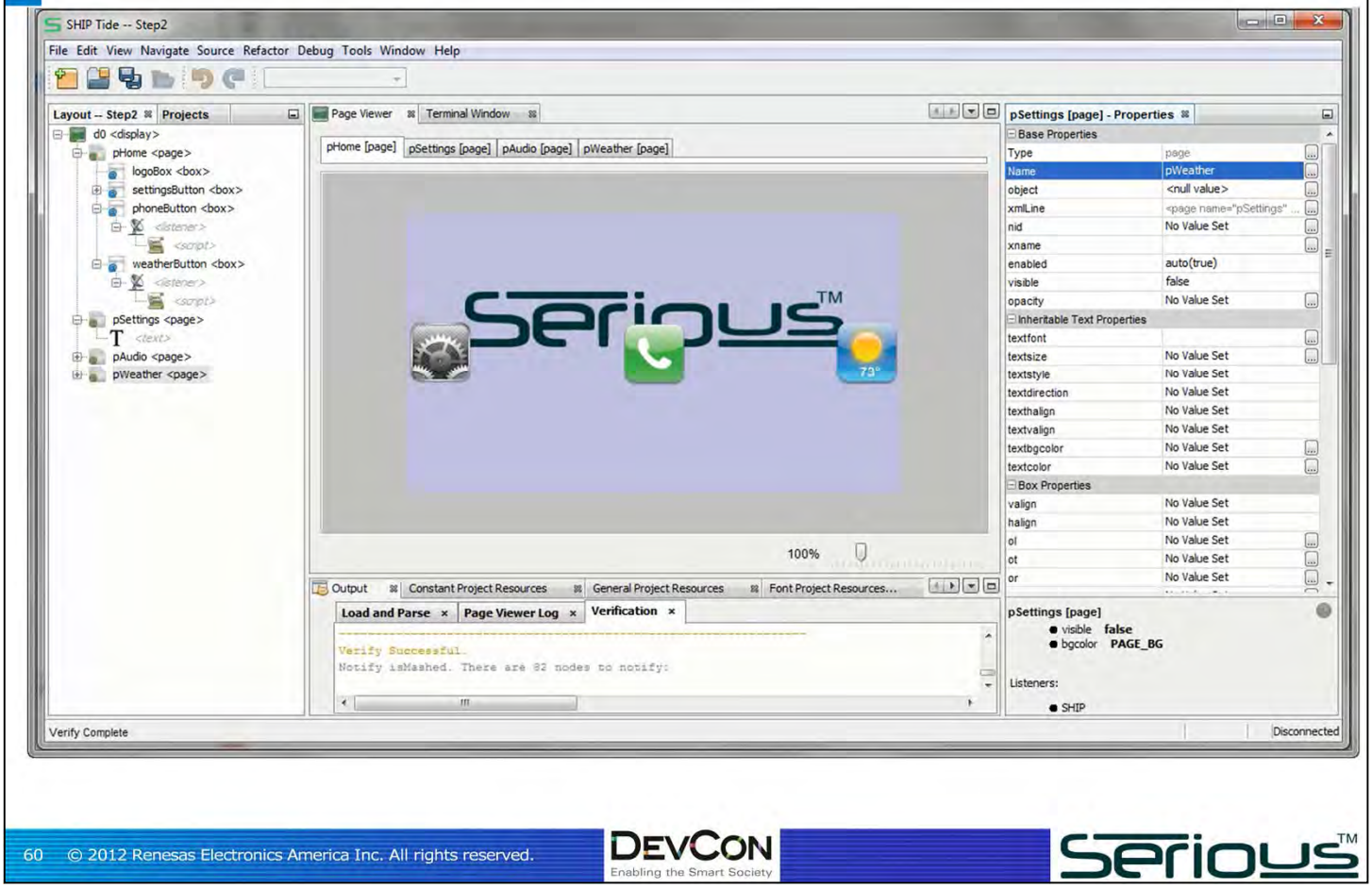
Let's look at the script and you'll see that just like we did on the **settingsButton** we made the current **<page>** invisible and we made a new **<page> pAudio** visible. There's the error: there is no **<page> pAudio**. Similarly under the **weatherButton**, the script is trying to make **<page>pWeather** visible, which does not exist yet.

The idea is these two buttons should transition to other pages, so we'll have to add those two pages if we want these errors to clear up.



Rather than **right-click/Add New→page**, this time let's just copy the **pSettings<page>** and paste it under the **d0<display>** twice. There are now 3 identical **<page>**s called **pSettings**.

Error Resolved



Rename the new instances of **pSettings** to **pAudio** and **pWeather** respectively. All of our errors are gone – the scripts now are able to resolve their references to the two **<page>**s – and we're looking at the main **<page> pHome** with the three buttons.

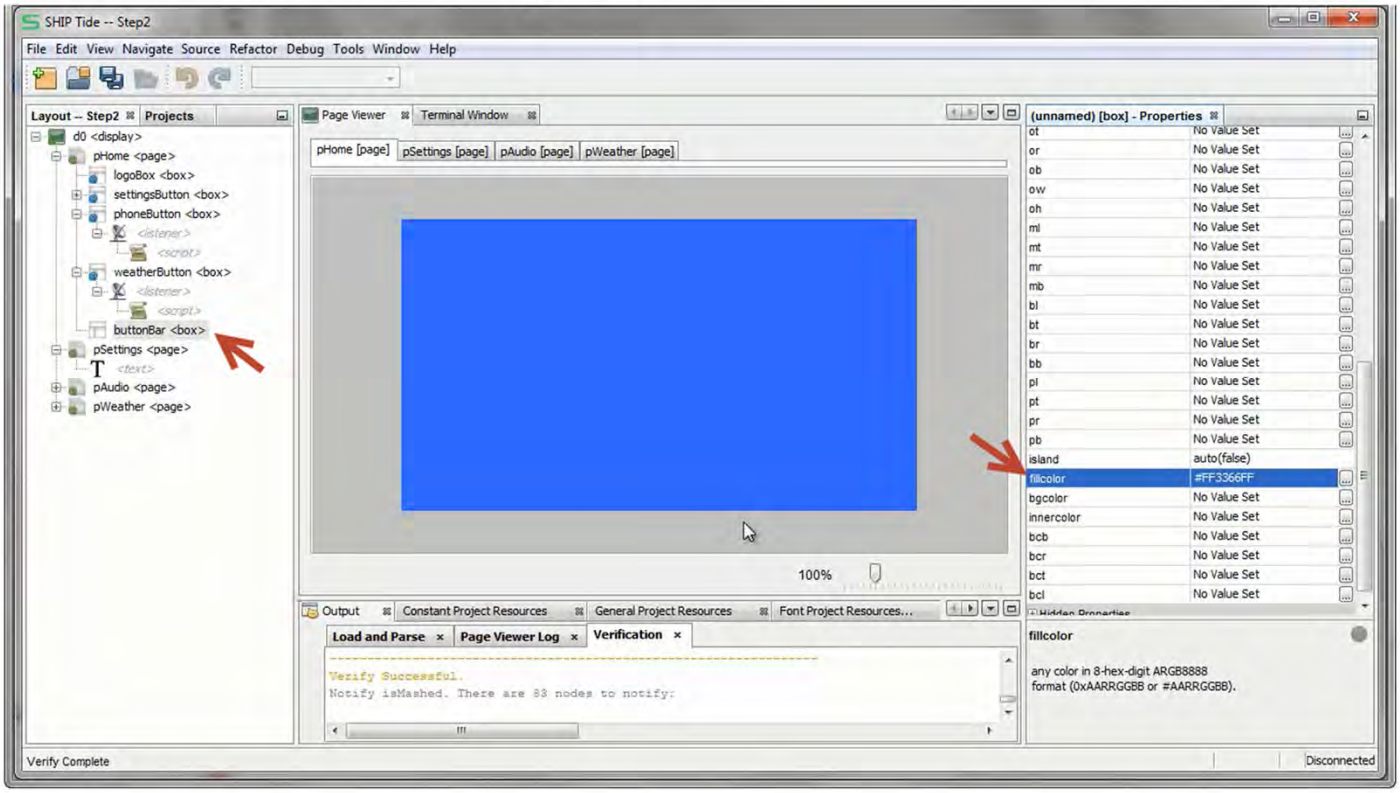
The positioning isn't quite right – all three are vertically centered and horizontally aligned left, center, and right respectively.

The edge buttons are too close to the edge, and we want them all vertically centered between the bottom of the logo and the bottom of the screen.

We could position each of these three independently (by calculating pixel offsets and changing each **<box>**'s outer top (**ot**) and outer left (**ol**) properties.

Instead, we're going to embed them inside an invisible **<box>**, like a button bar, that bounds the three images and allows for easier layout.

Creating the Button Bar for Enhanced Layout

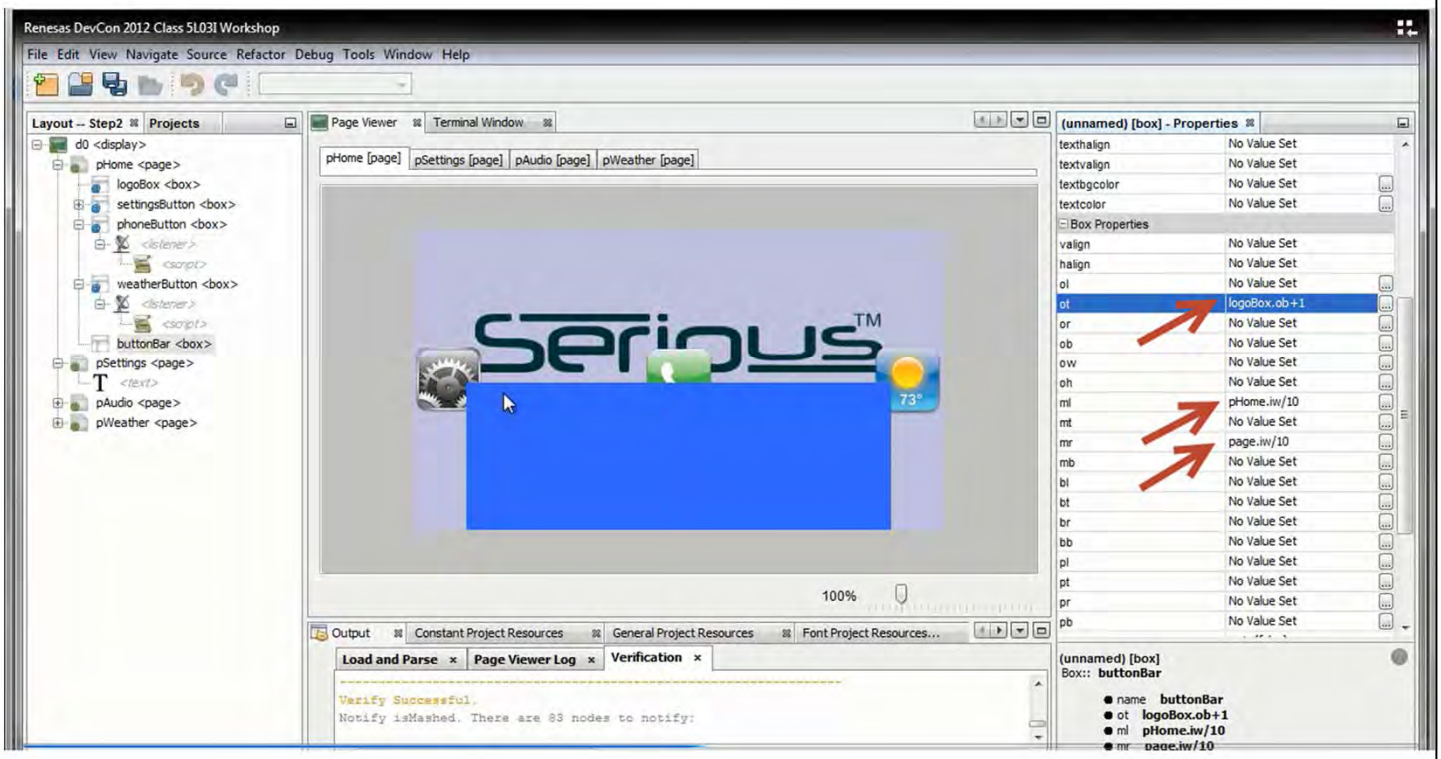


First, let's create a new **<box>** (call it "**buttonBar**") inside the **<page> pHome**.

Of course, the new <box> is invisible… so for now set **buttonBar**'s **fillcolor** property to some distinctive color.

fillcolor extends the **<box>** color fill only inside the margins of the **<box>** , rather than **bgcolor** that fills the whole **<box>** including margins. Since the margins are all unset, the color fills the whole **<box>**, which, because its extents are not explicitly set, fills the whole parent container.

Making the Button Bar with Layout Rules



We want:

- the left edge of this **<box>** in from the edge about 1/10th of the width of the **<page>**
- the right edge of the **<box>** about 1/10th in from the right of the **<page>**
- the top edge of the **<box>** just underneath the **logoBox**.

We can do it many ways, including a hardcoded 480/10 = 48 pixel value in both the margin left (**ml**) and margin right (**mr**), and set the outer top (**ot**) to some calculated number.

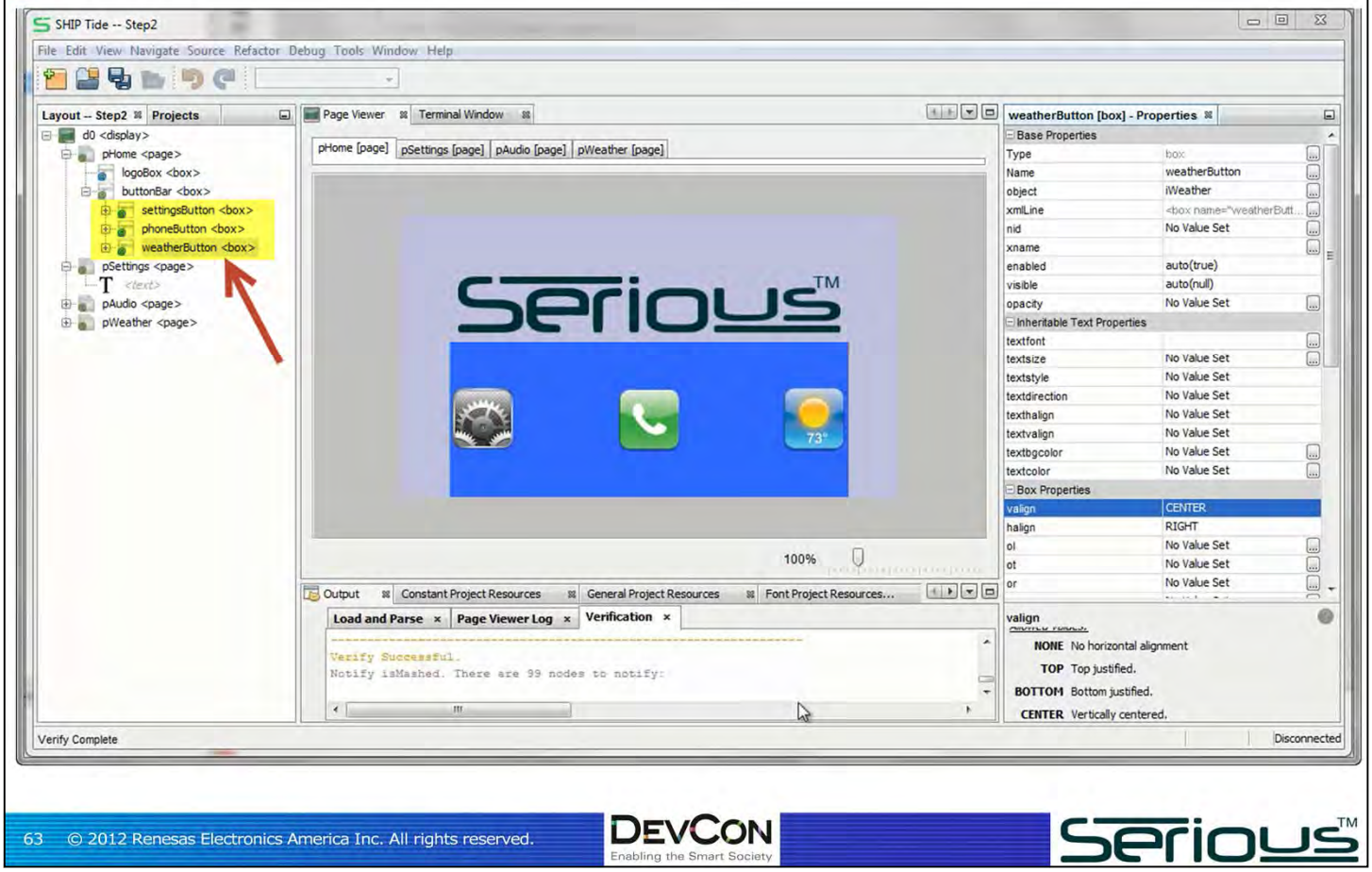
But we can do better: we can set these three dimensions using *rules*. That way if we change modules and screen resolutions, or want to re-proportion the **buttonBar**, it will be much easier.

On the **buttonBar <box>**, set:

- **ot** (outer top) to the rule "**logoBox.ob+1**", which sets the top of the **buttonBar** just under the bottom of the **logoBox**
- **ml** (margin left) to the rule "**pHome.iw/10**", which sets it to 1/10th the width of **pHome**
- **mr** (margin right) similarly to "**page.iw/10**", which sets it to 1/10th the width of the enclosing **<page>**, which of course is **pHome**.

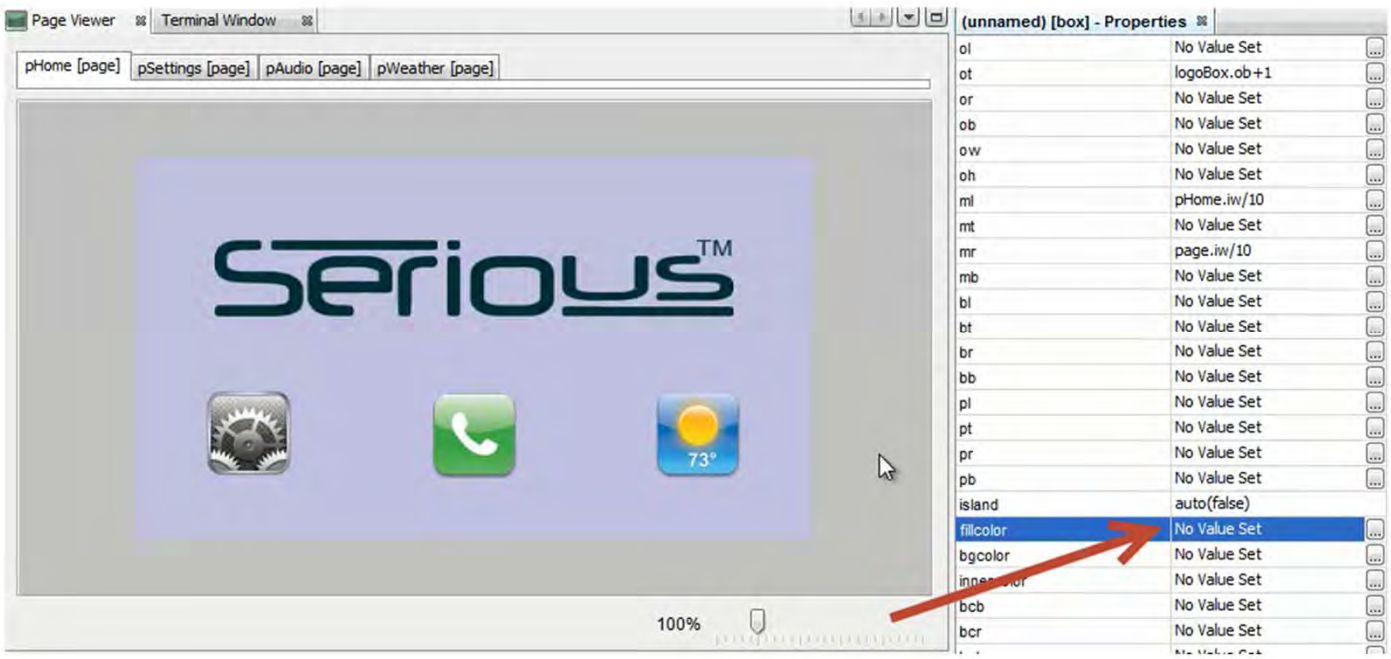
Now the **buttonBar** is perfectly positioned. If we move the **logoBox** up, the **buttonBar** will track it automatically.

Moving the Buttons into the Button Bar



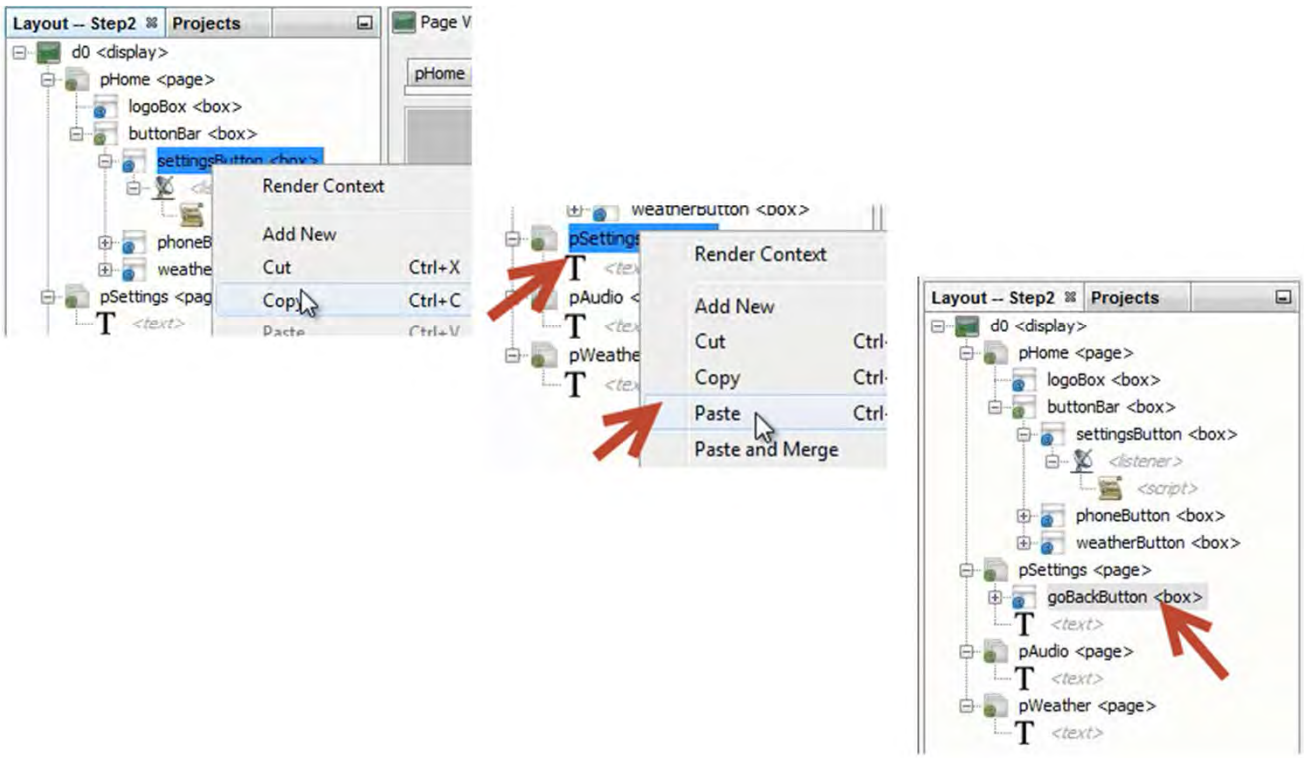
Just drag and drop the 3 buttons into the **buttonBar**. They'll now apply their **valign CENTER** property as well as their **halign LEFT**, **CENTER**, RIGHT properties in the context of their new parent: the **buttonBar**. And so they position perfectly inside the new **<box>**.

Finishing the Button Bar



Changing the **fillcolor** back to **unset** leaves us with an invisible **buttonBar** containing the three buttons.

Adding Back Buttons to the Pages



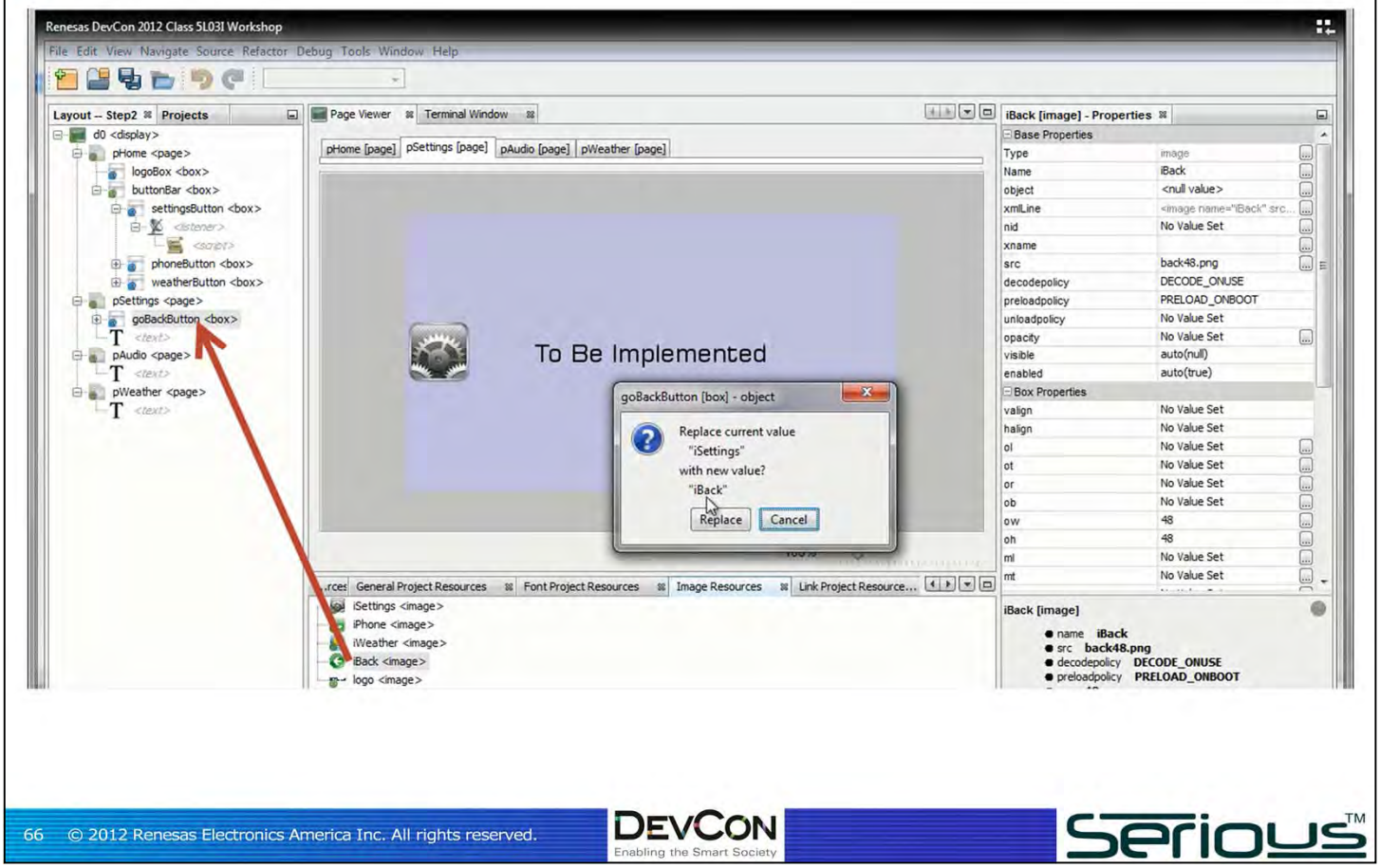
The three pages, **pSettings**, **pAudio**, and **pWeather**, can all be reached by pressing the three buttons **settingsButton**, **phoneButton**, and **weatherButton** respectively.

But once you get to any of those **<page>**s, it's a dead end… there is no way to get back to **pHome**.

We need to add "back" buttons to the upper right corner of each of these three **<page>**s.

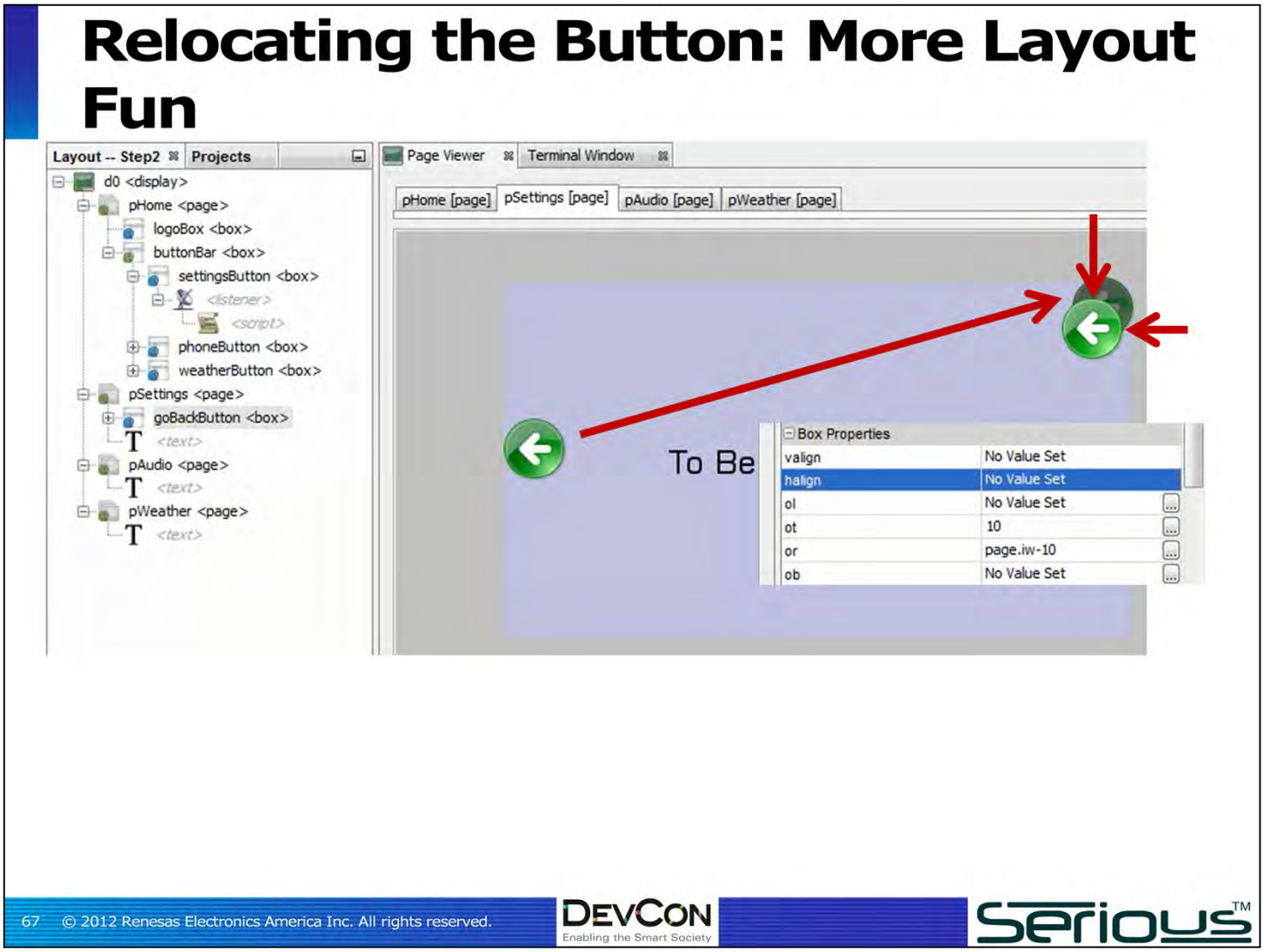
Let's copy our **settingsButton** which, when pressed, takes us to the **pSettings <page>**. Paste it into the **pSettings <page>** and change its name to **goBackButton**.

Replacing the Image



The **settingsButton** also has the **iSettings** image, and we want a “back arrow”. So, in the Image Resources panel, find he **back** image and just drag and drop it on the **goBackButton** node.

A warning will pop up: tell it to replace the old image in this specific **<box>** with the new one.

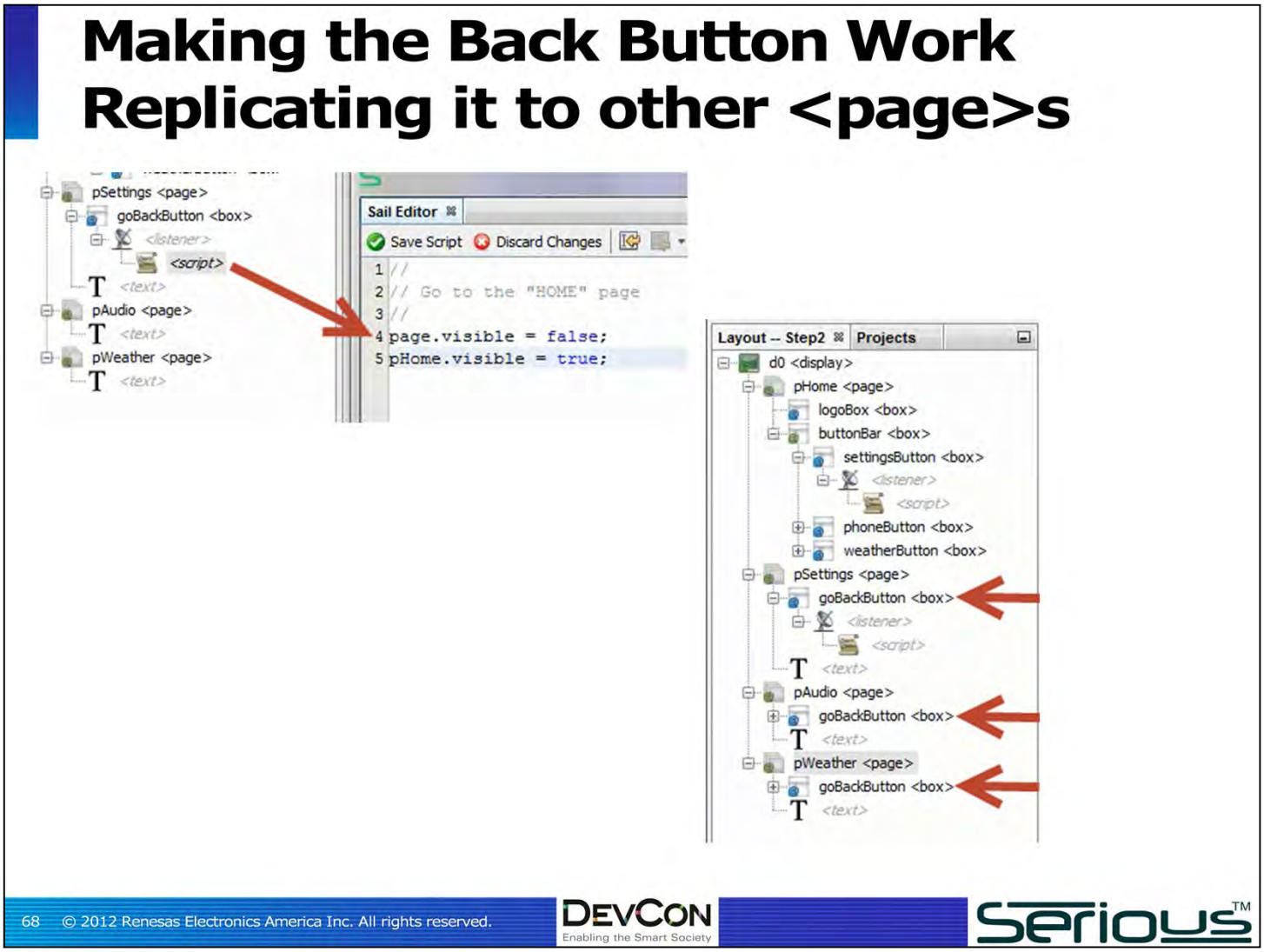


Of course, we need to reposition the **goBackButton** to the upper right corner of the **<page>**. We could do this by **valign TOP/halign RIGHT**, but this leaves the button crammed into the corner of the physical LCD screen – difficult to hit with your finger especially if there is a bezel around the screen.

Instead, set the following properties on the **goBackButton**:

- **ot** (outer top) to 10, which is 10 pixels down from the top of the **<page>**
- **or** (outer right) to the rule "**page.iw-11**", which puts the right edge of **goBackButton** 10 pixels in from the right edge of the **<page>.**

And now our button is in the right position.



Since our **settingsButton** went to the **pSettings <page>** and we want the **goBackButton** to go to the home page, open up the **<script>** and change the page to make visible to **pHome**.

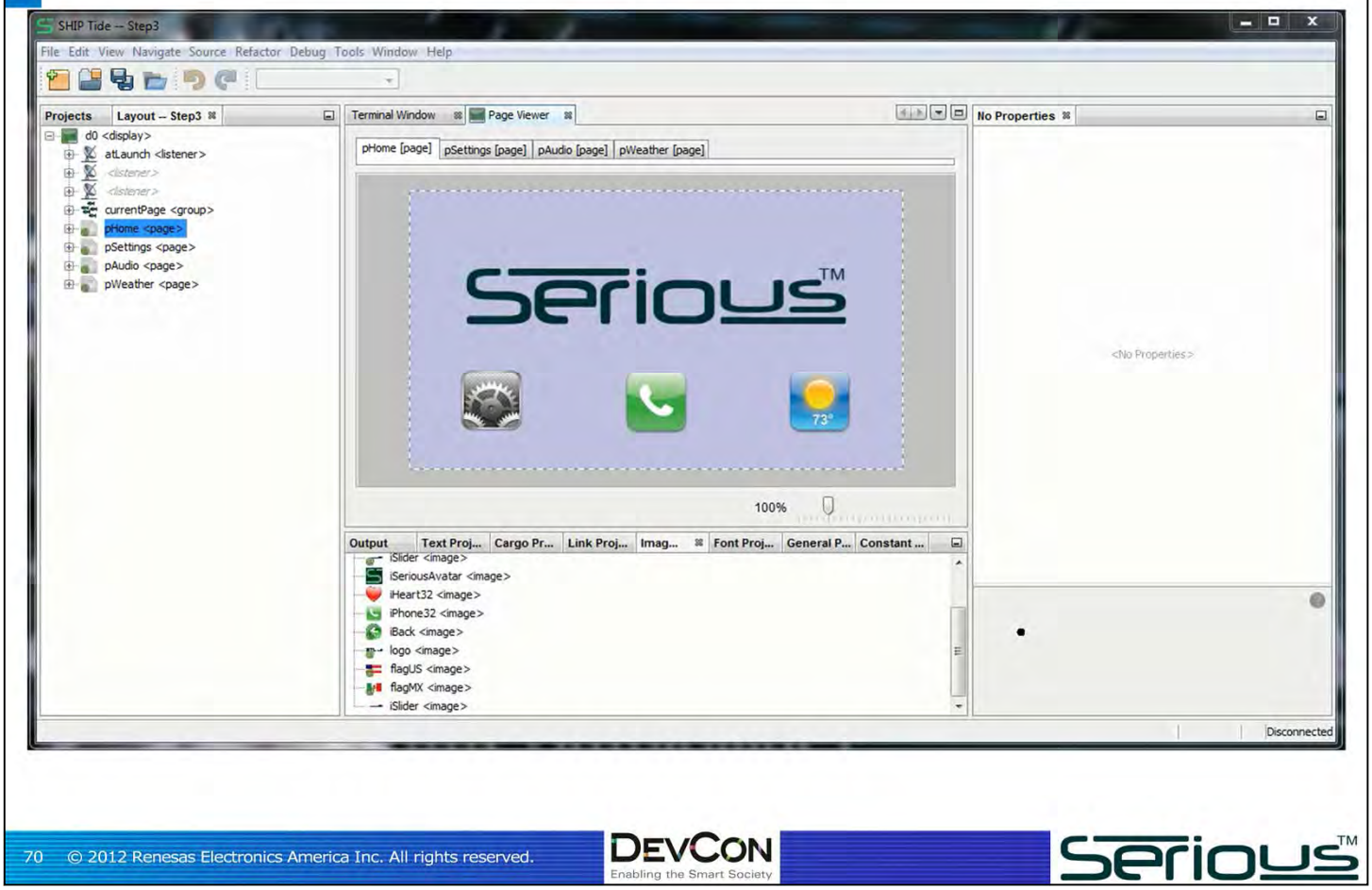
Then copy-paste this button into the other 2 pages (**pAudio**, **pWeather**). All three pages now have a functioning “back” button.

That’s the end of step 2 of the workshop. Export the new cargo, upload it to the SIM, and try it out!

Lab Workbook

- Step 1: Basic Operations – 20 minutes
- Step 2: Adding Actions and Audio – 20 minutes
- Step 3: Advanced Features – 20 minutes

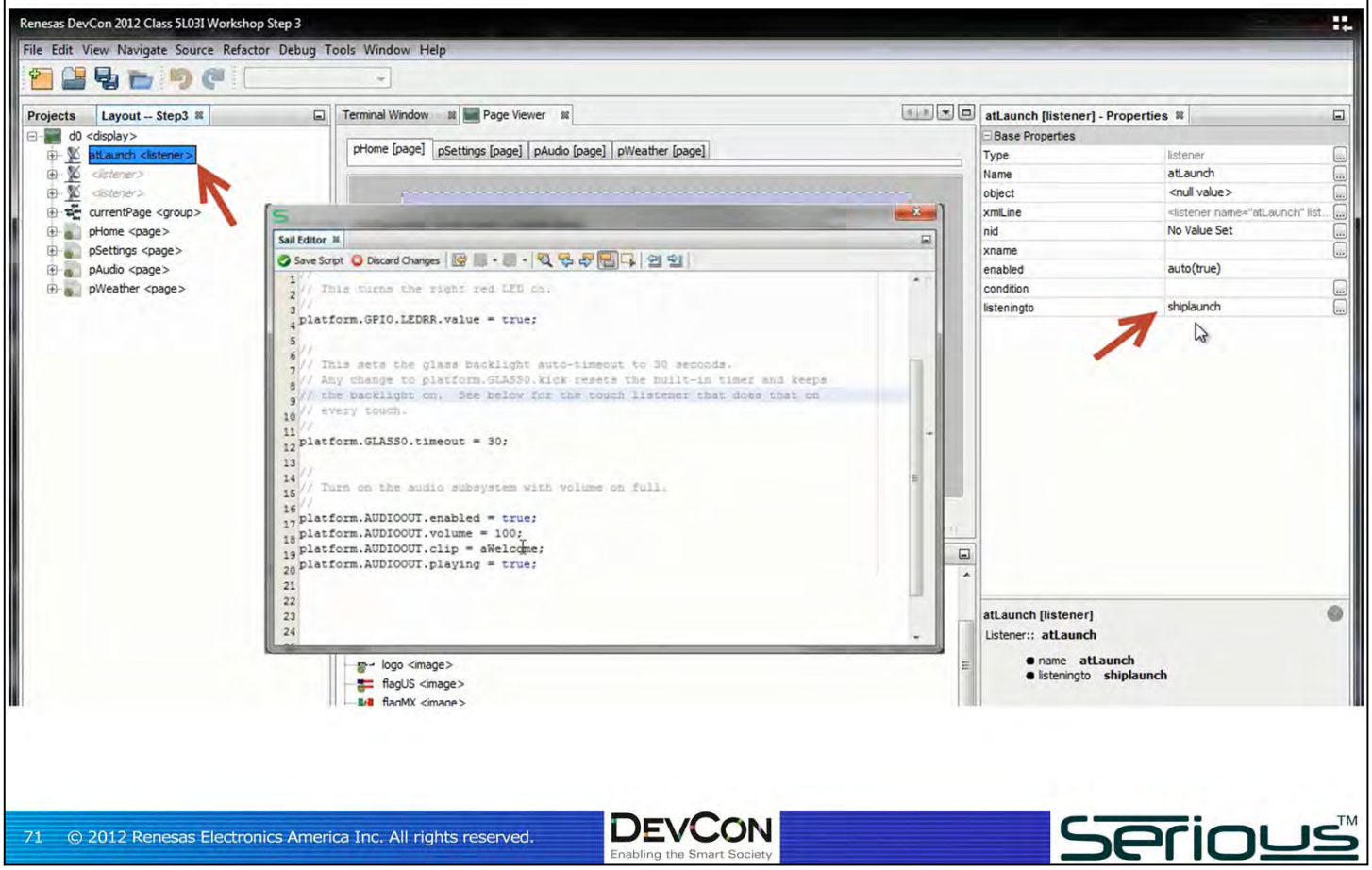
Step 3: Even More



Welcome to step 3 of the lab!

While the **pHome <page>** looks similar, we've made a number of improvements and additions for you to explore.

The shiplaunch Variable and Audio Clips



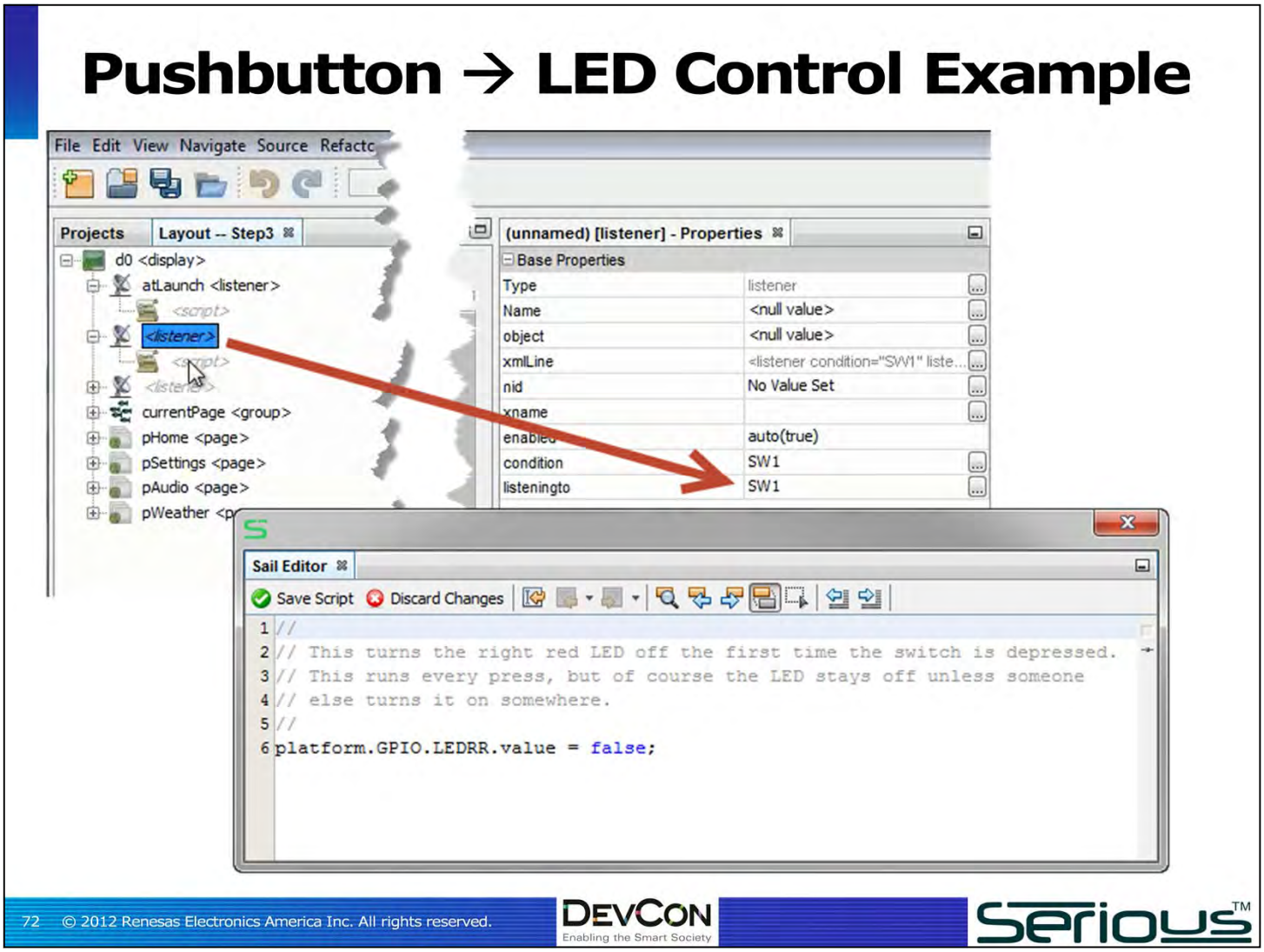
shiplaunch is a built-in boolean variable that goes active (true) when the GUI is booted.

You can listen to **shiplaunch** and run startup code; you can have as many **shiplaunch <listener>**s as you want, so rather than putting all your GUI startup code in one place, you can put it in every **<page>** or **<box>** to localize the initialization.

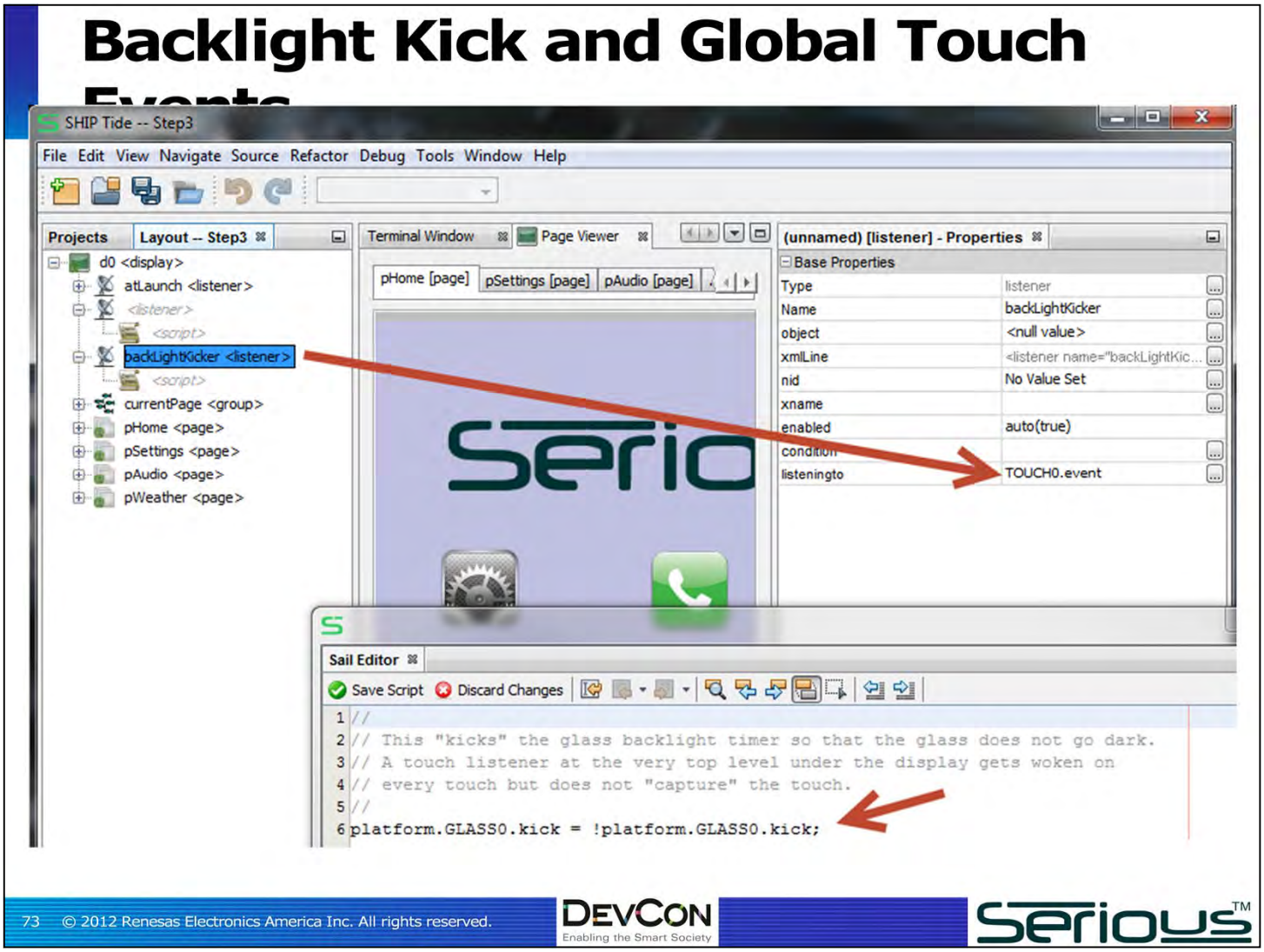
In step 3, you can see three things in the startup code:

1) Turning on the right red LED on the SIM205
2) Setting the LCD Backlight auto-timeout to 30 seconds
3) Turning on the audio subsystem and playing a “welcome” message.

Note how easy it is to play an audio prompt: just assign the audio resource (in this case **aWelcome**) to the audio output **clip**, and set the **playing** property to **true**.

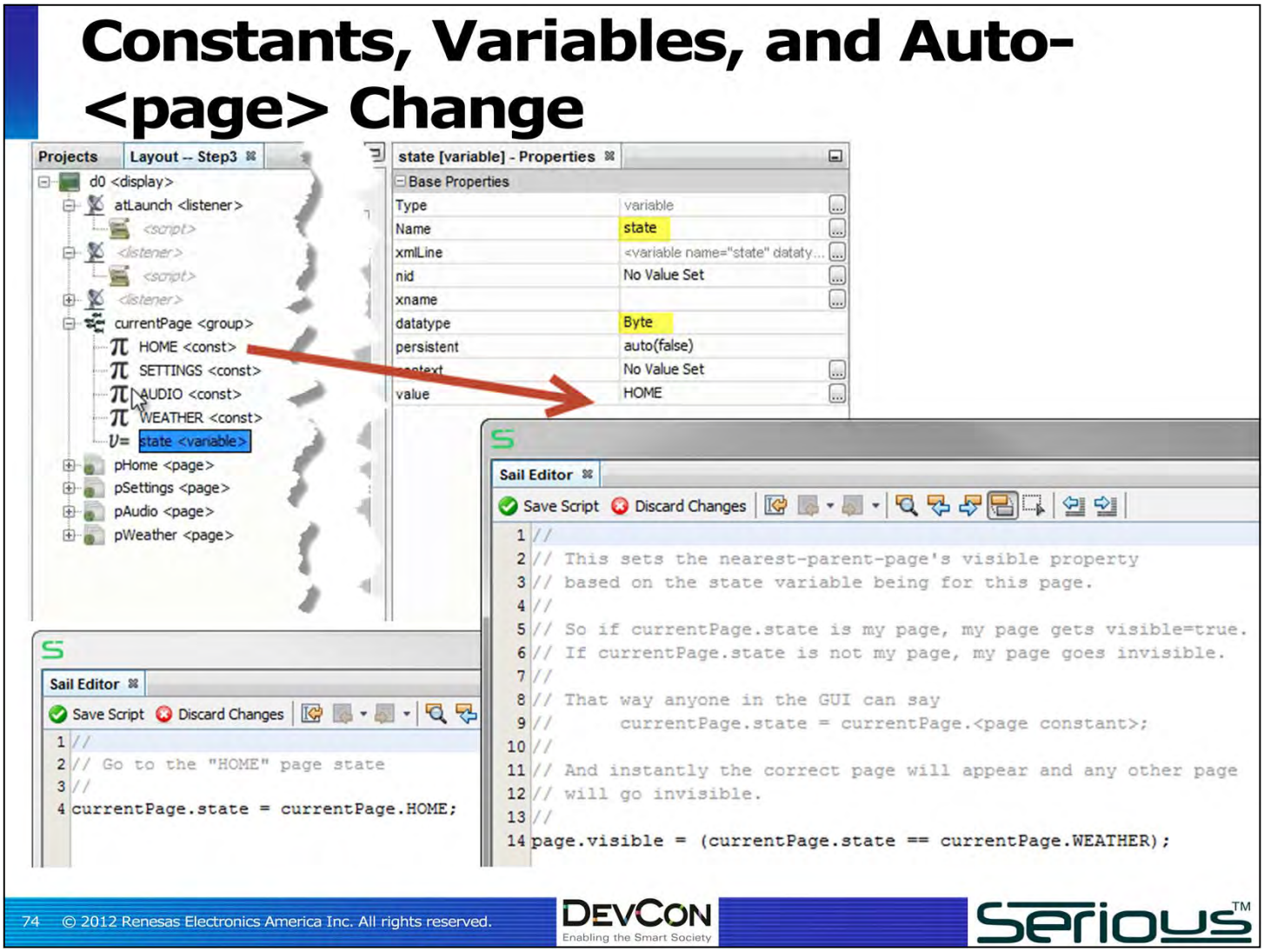


We just showed you how the red right LED was turned on at boot. This **<listener>** is listening to changes on the switch on the front of the SIM205, and when it is pressed (the **condition** tests against **true**) the associated <script> turns the red right LED back off.



A touch **<listener>** directly underneath the **<display>** is special, and can wake up on any touch event every time without impeding any other **<box>** that is waiting for a touch event. This special functionality is particularly useful when it “kicks” the LCD backlight timer to keep the backlight awake. The **<script>** just toggles the **kick** property of the glass, which resets the timeout on the backlight timer. So every time you touch/untouch the display, it ensures the backlight is (stays) on.

You could also, for example, kick the backlight when an alarm is detected in your system, and even disable the backlight timer (**platform.GLASS0.timeout = 0**) until the alarm is acknowledged by the user.



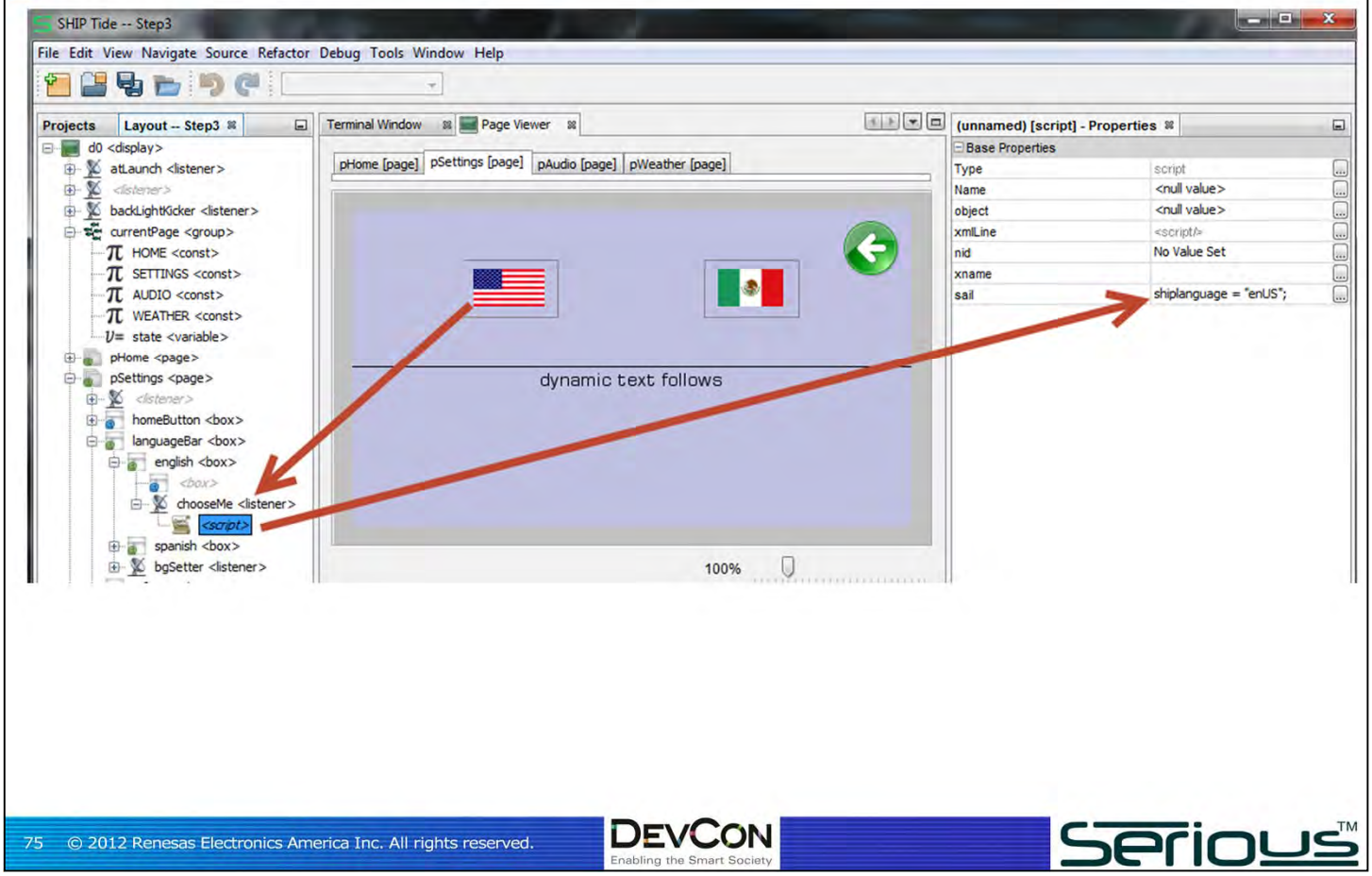
In the prior steps of the lab, we changed **<page>**s via a SAIL **<script>** like this:

pWeather.visible = false;

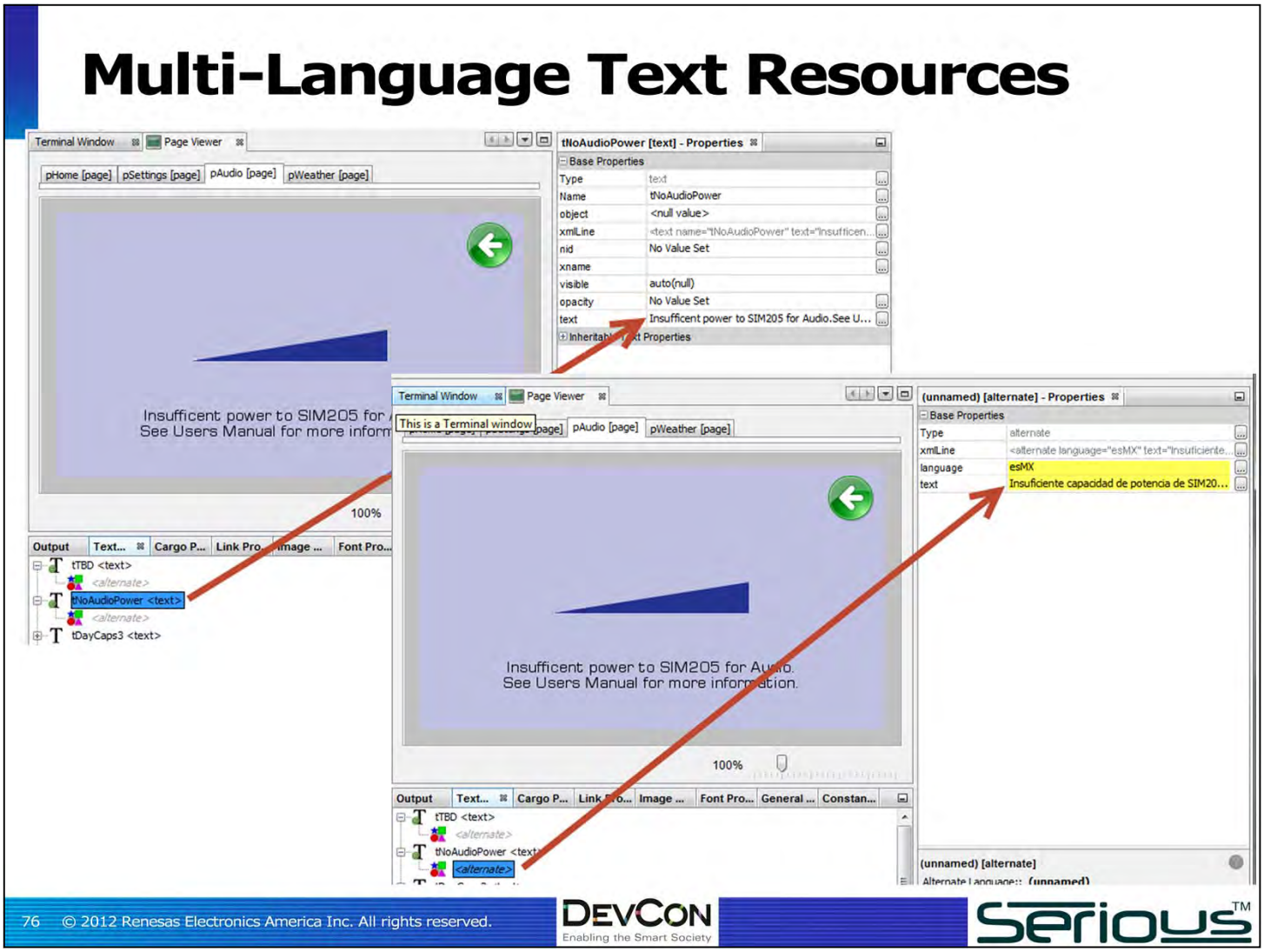
pHome.visible = true;

In step 3, you can drill into the **currentPage <group>** and see the equivalent of an enumerated type (values **HOME**, **SETTINGS**, **AUDIO**, **WEATHER**), with the value of the variable in **currentPage.state**. Adding a simple listener to each page, listening to **currentPage.state** can then change the page change system to a cleaner, more contained methodology. Examine the **<listener>** at the top of **pWeather** and see how it makes **pWeather** visible/invisible automatically based on the **currentPage.state** value.

The shiplanguage Built-in Variable

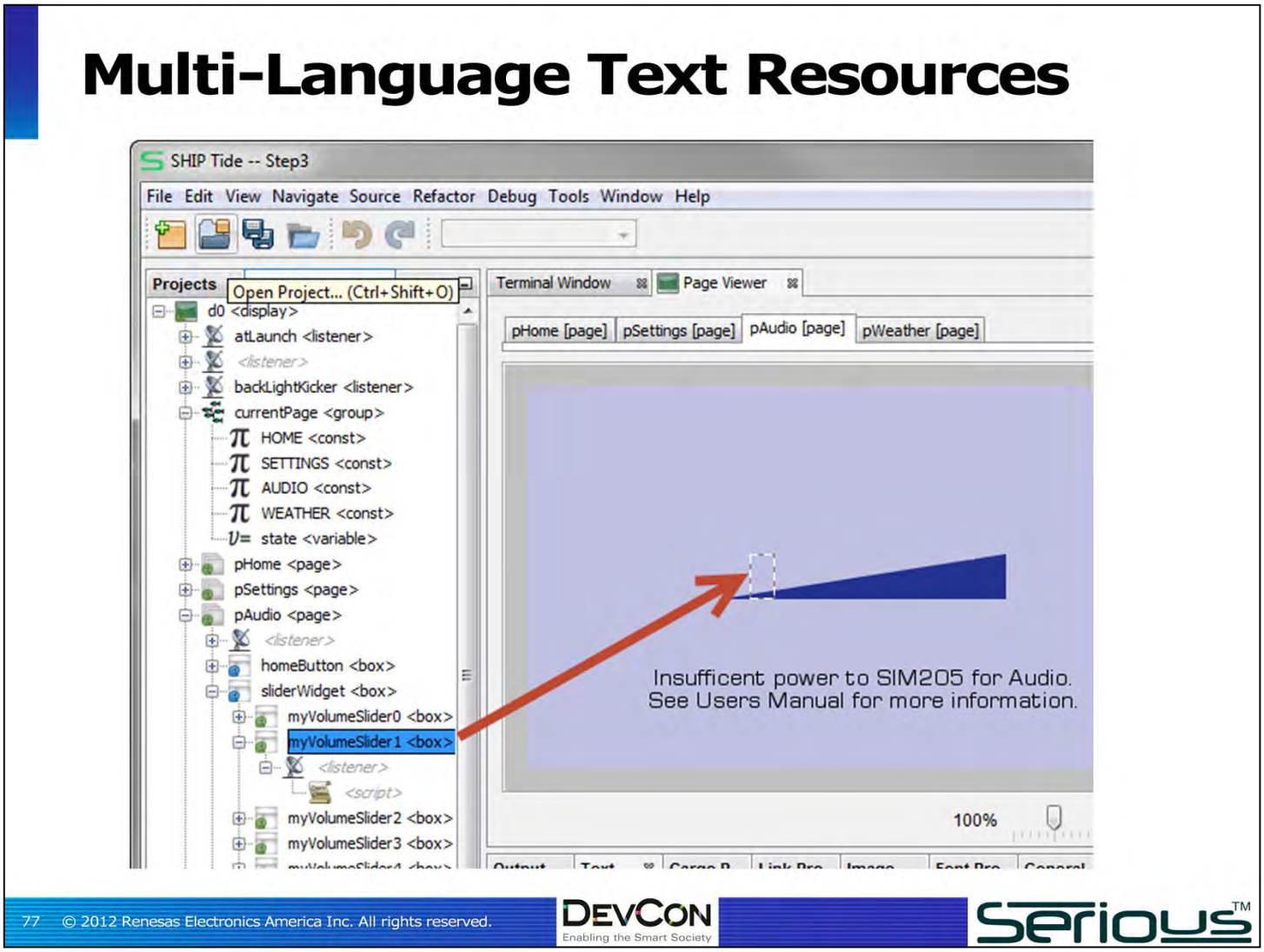


SHIP supports easy implementation of multi-language GUIs. In the Step 3 project, the **pSettings <page>** has 2 new buttons (the US and Mexican flag). The **<script>**s attached to these buttons set the built-in variable **shiplanguage** to an industry standard 4 letter code, for example "**enUS**" (meaning English as spoken in the US). The **shiplanguage** variable determines what text resources are displayed on pages: changing **shiplanguage** in a **<script>** will automatically cause all associated **<text>** that uses resources to refresh.



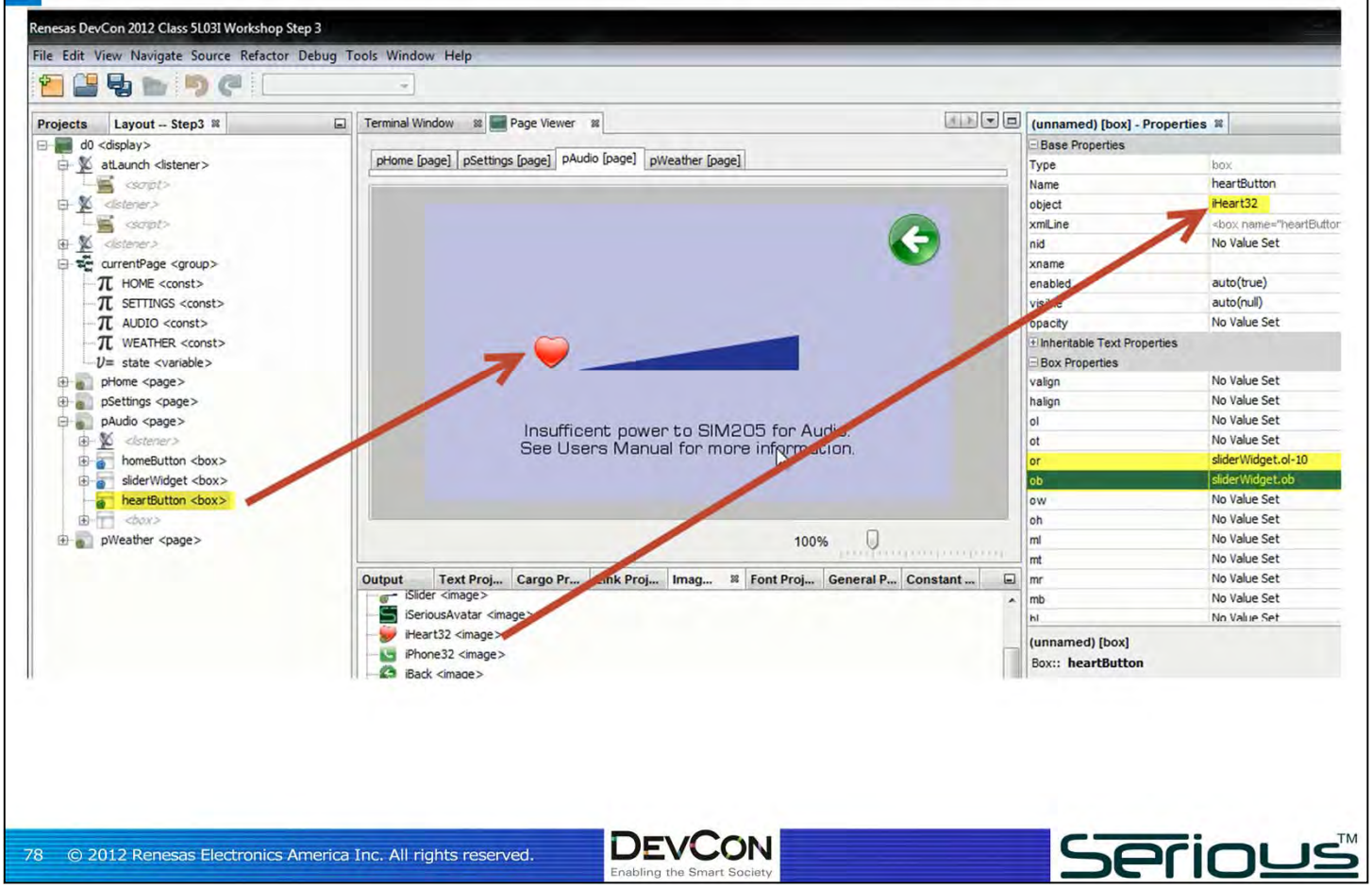
The text on the pAudio page is multi-language enabled. It uses the **tNoAudioPower <text>** resource which has a default value of “Insufficient power to SIM205 for audio…” value. But it also has an alternate version for the esMX language (Spanish as spoken in Mexico). When **shiplanguage** changes, the alternates are searched for a match to the **shiplanguage** 4-letter code. If a match is found, that text is displayed. If not, the default is used.

Multi-Language Text Resources



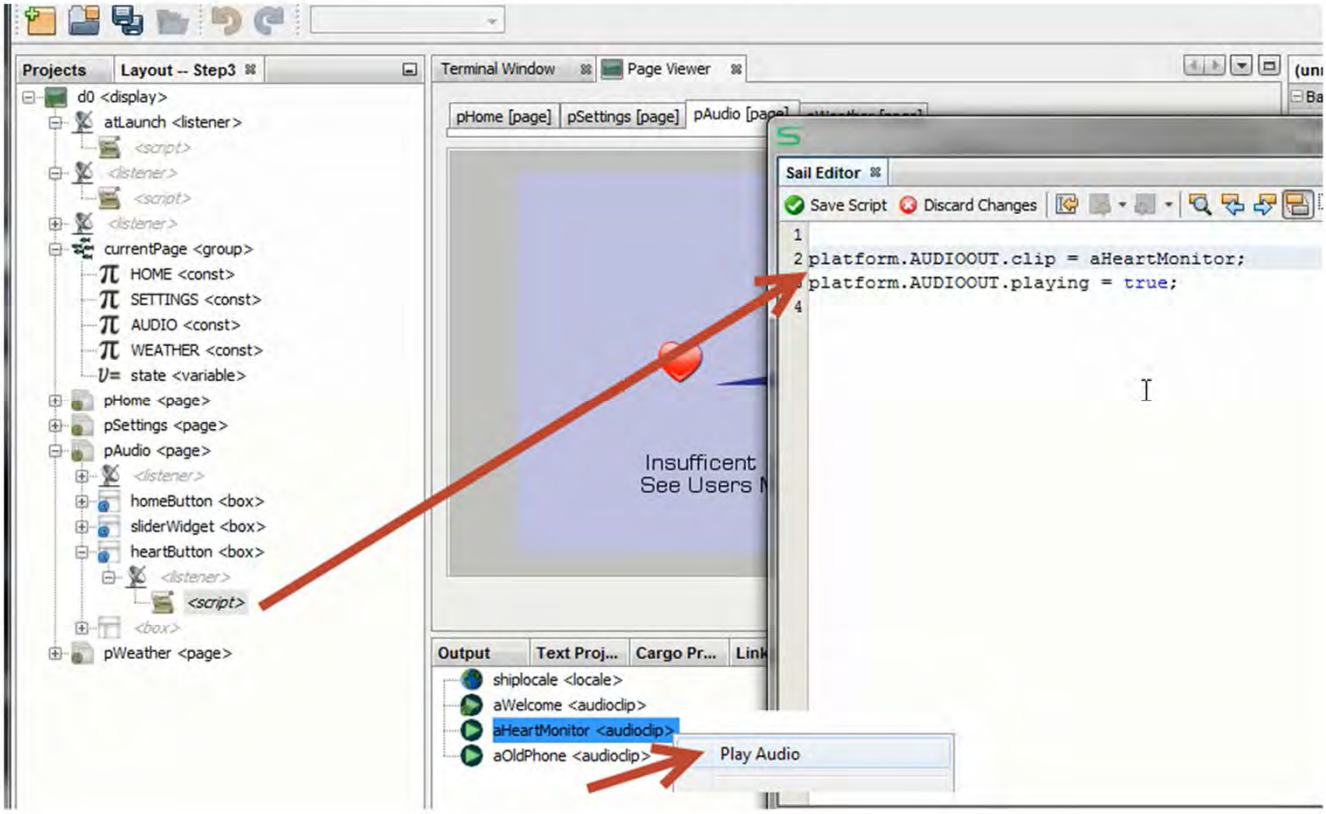
We've implemented an audio volume slider widget. Examine the transparent boxes that overlay the slider image, and see how each one has a touch **<listener>** that sets the audio to the correct volume based on the slider position. There's also a pretty interesting **switch** statement that manages a semi-transparent overlay at the current volume level.

Adding (yet another) Button



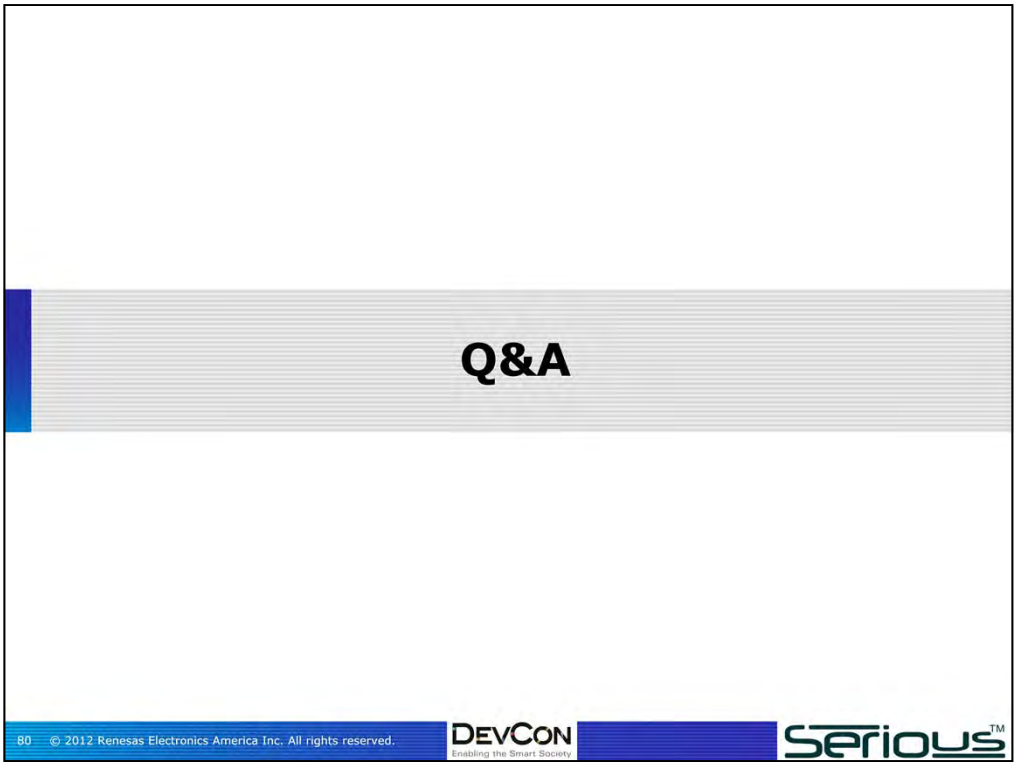
We're going to add a button to start an audio clip. Create a new **<box>** and assign it the **iHeart32 <image>** resource, and align it to the direct left of the slider.

Starting an Audio Clip



Add a standard touch **<listener>** to the button, and in the script assign to the audio output device clip the **aHeartMonitor <audioclip>** resource, then just start it playing by setting the **playing** property of the audio output device.

Note within SHIPTIDE you can actually listen to the **<audioclip>** by **right-click/Play Audio** on the resource.


Q&A
80
© 2012 Renesas Electronics America Inc. All rights reserved.
DEVCON
Enabling the Smart Society


DEVCON
Enabling the Smart Society
RENESAS
Renesas Electronics America Inc.
© 2012 Renesas Electronics America Inc. All rights reserved.