Running an ABL GUI as an HTML5 Web Application Without a Rewrite

Greg Shah, CEO
Golden Code Development

Friday November 17, 2017
www.beyondabl.com

Agenda

- Background
- UI Paradigms Over Time
- Modernization Alternatives
  - Rewrite
  - .NET Facelift
  - Enhanced 4GL
- Running 4GL GUI as Web
- Javascript Integration
- Enhanced 4GL Capabilities
- Status

What is FWD?

- **Open Source** implementation of the 4GL (not OpenEdge)
- Entire ABL applications
  - code, schemata, data
  - projects of any size
  - any level of complexity
- Fully automated, non-interactive
- Drop-in replacement (OE compatible)
- Leverages existing 4GL code investment (including GUI)
- Provides evolutionary path forward

Who Is Golden Code?

The team of engineers and computer scientists that created the FWD technology.

We use FWD to help our clients solve the toughest ABL refactoring, transformation, and modernization problems.

www.goldencode.com

UI Paradigm Evolution

Strategic Assessment

- Web/Mobile is the strategic UI paradigm
- The shift away from desktop applications occurred between 2007 and 2011
- In 2017, “green field” business software is NOT written using Windows GUI APIs, including Windows Forms (i.e. OpenEdge GUI for .NET)
- Windows Forms is not strategic, *even for Microsoft;* placed in maintenance mode in 2014
- OpenEdge GUI for .NET is an evolutionary dead end
- Moving to web and mobile should be a high priority for all business software
- Disparity between 1990’s style WIN32 GUI and modern UI paradigms driving UI Modernization

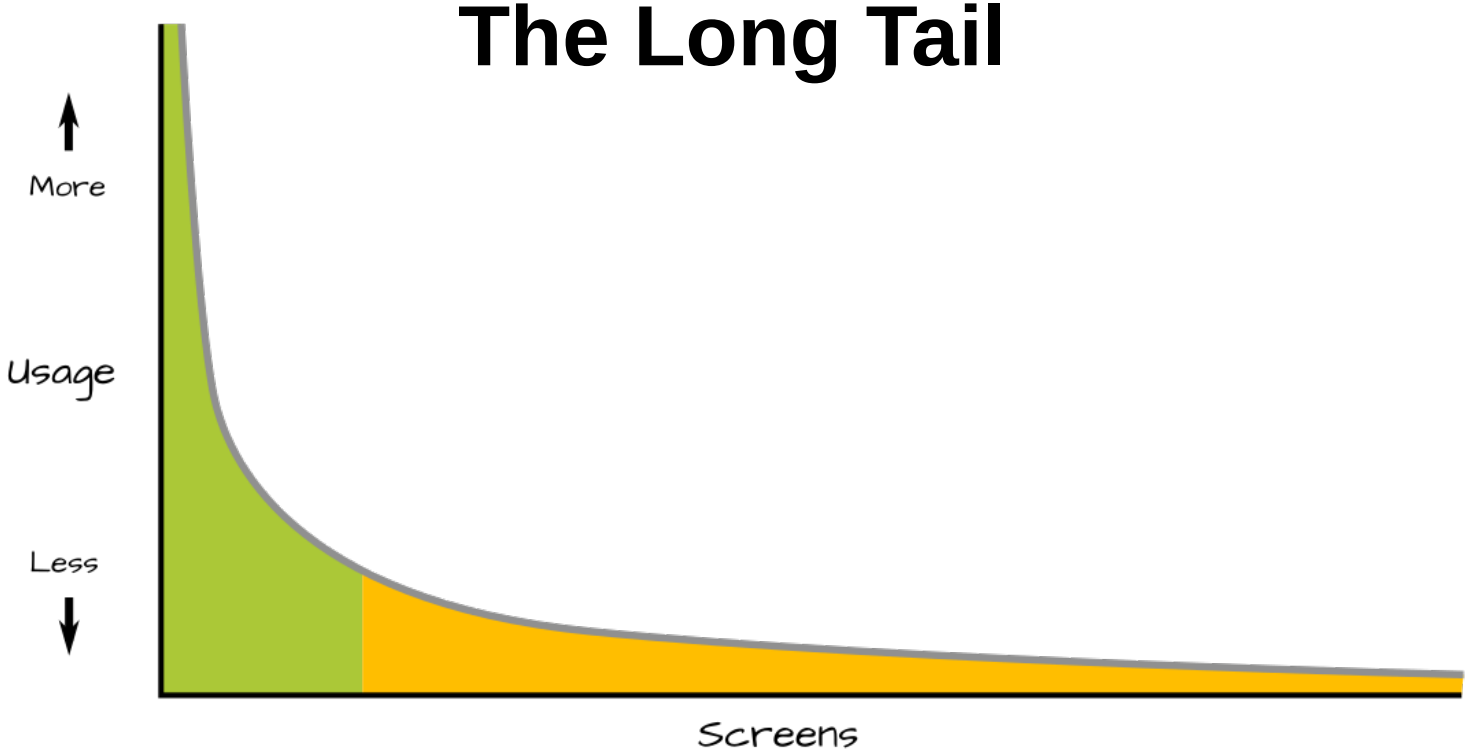
UI Modernization Approaches

- Rewrite
- .NET Facelift
- Enhanced 4GL

Rewrite

- Phase 1: Refactor
  - Traditional 4GL code: tight coupling of business logic, data, and user interface
  - UI code must be separated from business logic and data access
  - Often done as an API that exposes business services
  - Another common approach: expose business entities as data objects
  - Leave business logic and data access in 4GL, eliminate UI processing in that code
- Phase 2: Rewrite
  - Write new, modern UI as non-4GL front end
  - Use native web (and sometimes mobile) technologies
  - From non-4GL language, consume API / business entities from OpenEdge back end
- Progress' preferred approach is JSDO and Kendo UI Builder/Nativescript
- Solutions exist which are designed to write front end UIs, using OpenEdge as back end

The Long Tail



- Each dialog/window/ADM[2] tab is considered a “screen”
- Non-trivial applications have surprisingly large number of screens; empirical experience suggests at least 500 screens per 1MLOC of code*
- Most (~90%) of application’s high value features done with ~10% of screens
- On average, cost and effort to rewrite a low value screen is same as to rewrite a high value screen
- Return on investment for ~90% of screens never achieved

* FWD Code Analytics can be used to get an exact number, no matter how large the project.

Per Screen Work

- Front end tools help, but UI code is tricky and non-trivial; time-consuming to get replacement right
- Refactoring of old business logic into Business Entities a big part of the “hidden work”
  - Simplest case screen maybe 1 to 2 days
  - Complex, more fragile screen maybe 5 to 10 days
- UI side (creating a new screen or screens to replace the original)
  - Simple screen maybe 2 to 5 days
  - Complex screen perhaps 10 days
- Per screen best case: 3 to 7 person days
- Per screen worst case: 15 to 20 person days

Person Years Per 1MLOC

- Variations in developer productivity
  - other work, interruptions, meetings, unexpected problems
  - best case not often achieved
- With 48 working weeks and a 40 hour work week: maximum yearly productivity is 240 days
- Assume 20% of developer time spent with normal overhead
  - probably optimistic
  - would be remarkable to achieve 192 full person days of real work per year
- Assume average screen complexity is medium (10 days of work per screen)
- 500 screens * 10 days per screen = 5,000 person days (26 person years) per 1MLOC of code

Rewrite Odds of Success?

- For many non-trivial applications, effort to rewrite UI so large, it is too much to reasonably get to end of job
- Most of the work is for low value screens
- Refactoring tricky; carries a high likelihood of regressions and breakage
  - adding proper specs and testing into the development effort multiplies the effort further
- Effort spent on rewrite is not spent on other functional tasks
  - opportunity cost of dedicating this scale of resources; and/or
  - real cost of adding more resources
- How many organizations do you know that have finished this rewrite?
  - the classic “never ending project”

.NET Facelift

- Pressured to modernize UI, but finding that rewrite option impractical, some look to .NET as a short term solution
- Less work than a full rewrite, though not trivial (months, not days)
- Not an evolutionary path to web/mobile
  - fat client gets fatter
- Now harder to move to web/mobile
  - significant dependency encoded to non-strategic Windows .NET UI
- Some organizations desperate to show some UI improvement, so they go down this path, not realizing that there is a better option

ChUI to GUI Transition

- 4GL Character UI code was largely “future proofed”
- Applications could transition ***without*** completely:
  - refactoring business logic;
  - rewriting UI
- Transition was possible with only modest edits to existing ChUI code
- UI could be brought over to new UI paradigm and then over time, GUI enhancements and GUI-specific capabilities were leveraged
- Allowed a natural evolution while maintaining existing investment in UI and business logic
- If Progress had not taken this approach, it might not exist today

GUI to Web Transition

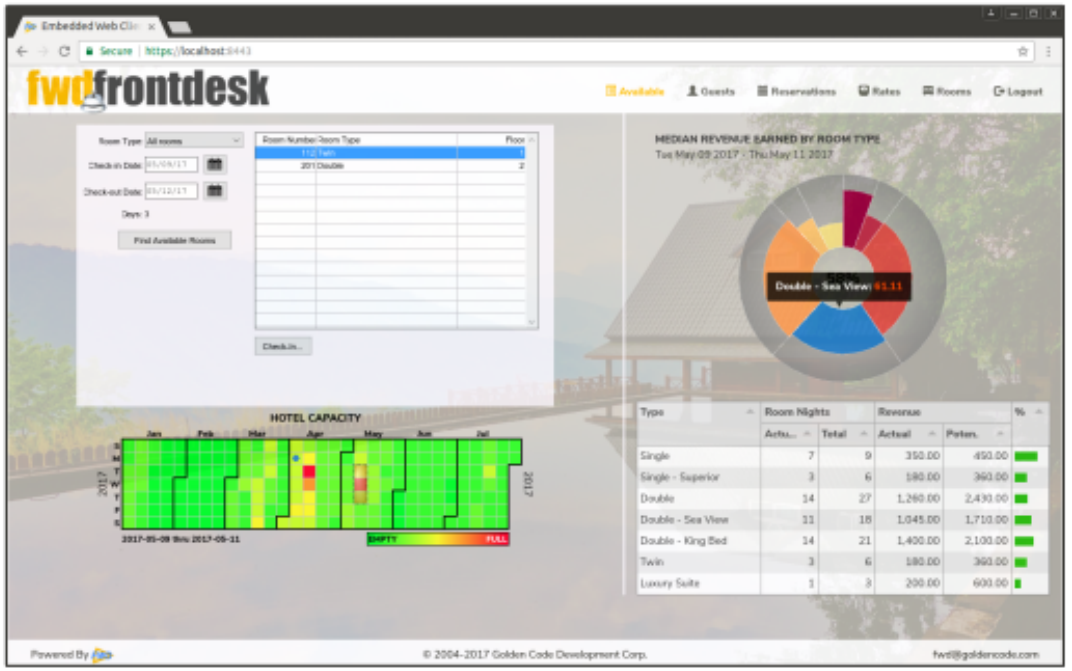
- Last significant investment in the core 4GL UI capabilities was 1998 (Progress 4GL v9 stabilized GUI code, added ADM2)
- In 2017, 4GL GUI may best be described as deprecated
- Progress recommends using non-4GL technology for UI modernization
- There is no future proofing of GUI code
  - no evolution path
  - existing investment is devalued
- Recommended rewrite path is very hard to achieve and involves large, unnecessary costs

Extending 4GL GUI to the Web

- What if existing 4GL GUI could run in the web, without a rewrite?
- What if 4GL GUI features were enhanced to add new capabilities, new functionality, a modern UI approach?
  - some improvements could be implemented without code edits
  - other improvements require some new code (or changing existing code to use new features)
- Clean, evolutionary path that starts with existing GUI investment and allows developers to achieve fully modern UI without a rewrite
- Developers focus on high value screens; modernization effort is additive/incremental
- Simultaneously avoids long tail problem while future proofing existing investment
- Would require re-imagining/re-implementing core 4GL UI capabilities, from the bottom up
- This is exactly what many 4GL developers would have wanted Progress to do

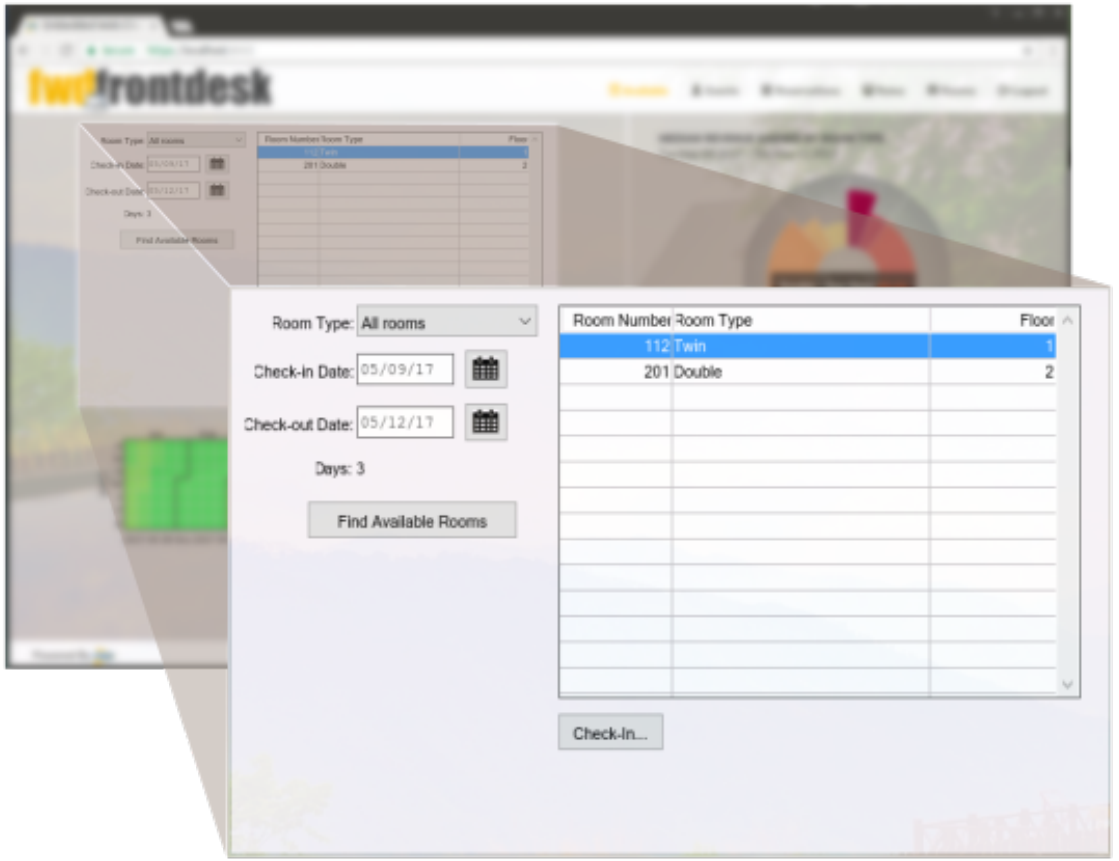
FWD is this re-imagined, enhanced 4GL.

Custom Web Application



- HTML5/CSS/Javascript
- Single page application (no reloads)
- Javascript controls bound to ABL data
- FWD client as separate document in an IFrame (rectangular, visual element)
- Main (containing) document and IFrame document have different "origins"
- Cross-Document Messaging allows communication between the two

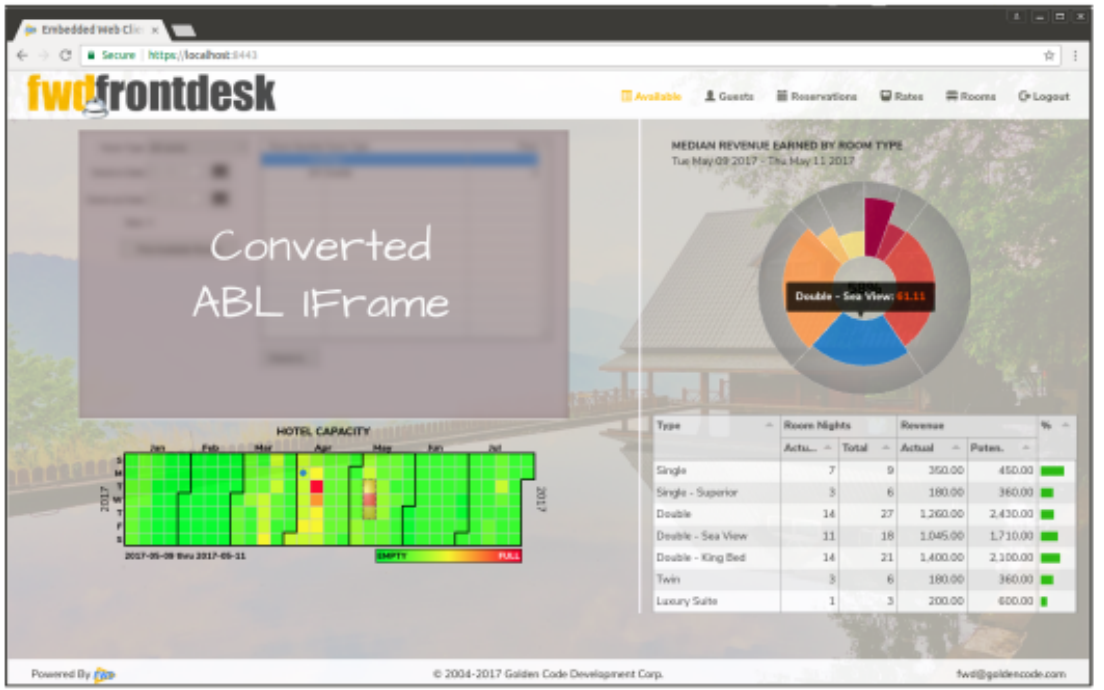
FWD Embedded Web Client



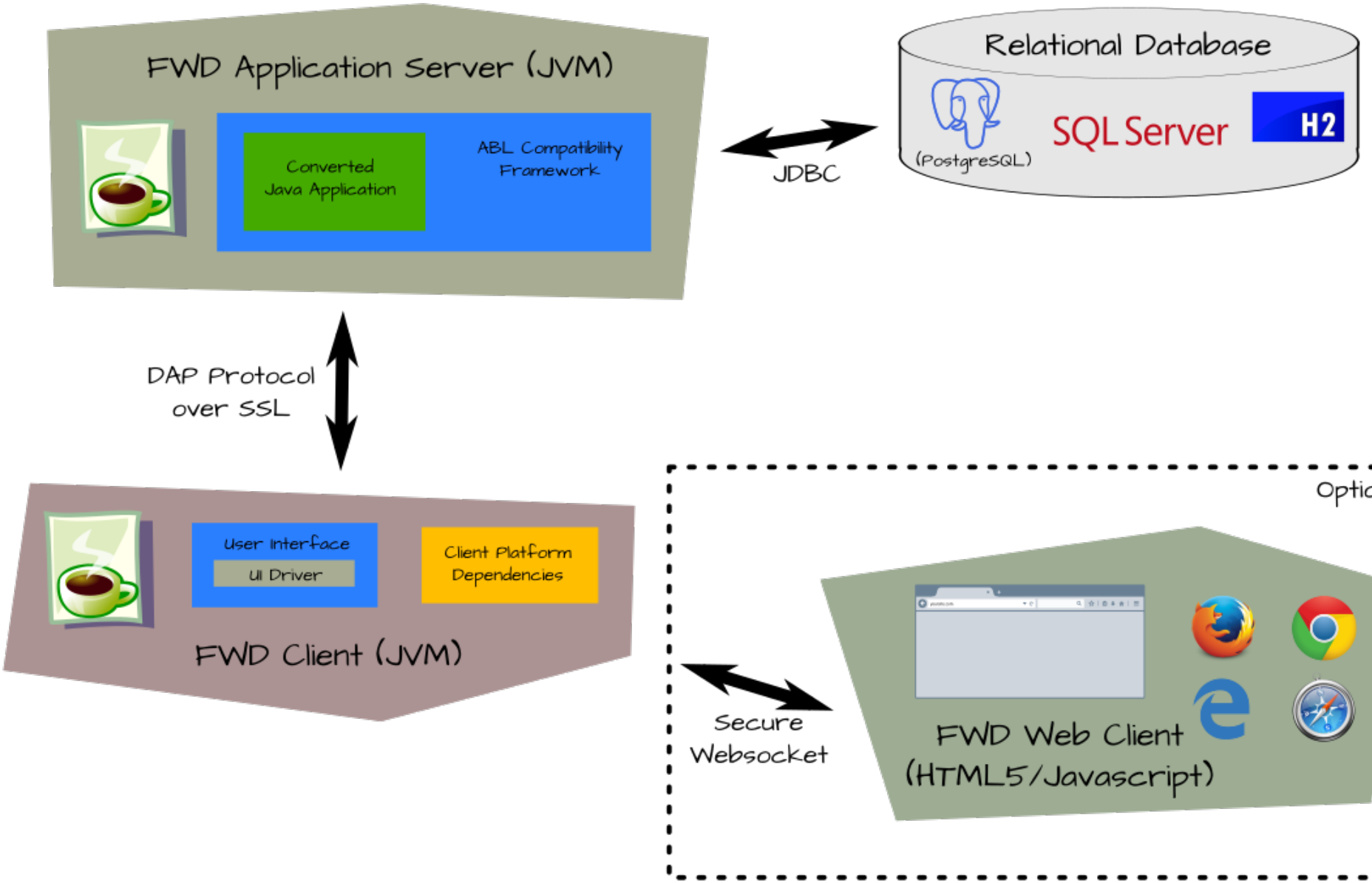
- HTML5/CSS/Javascript
- Converted ABL code runs in the Java application server
- ABL UI renders in an IFrame
- Non-modal windows "chromeless"
- Dialogs are modal with a shading overlay
- API for control of screens by containing web page (open, close, hide, surface...)
- Javascript appserver API
- Javascript PUBLISH/SUBSCRIBE integration with converted ABL

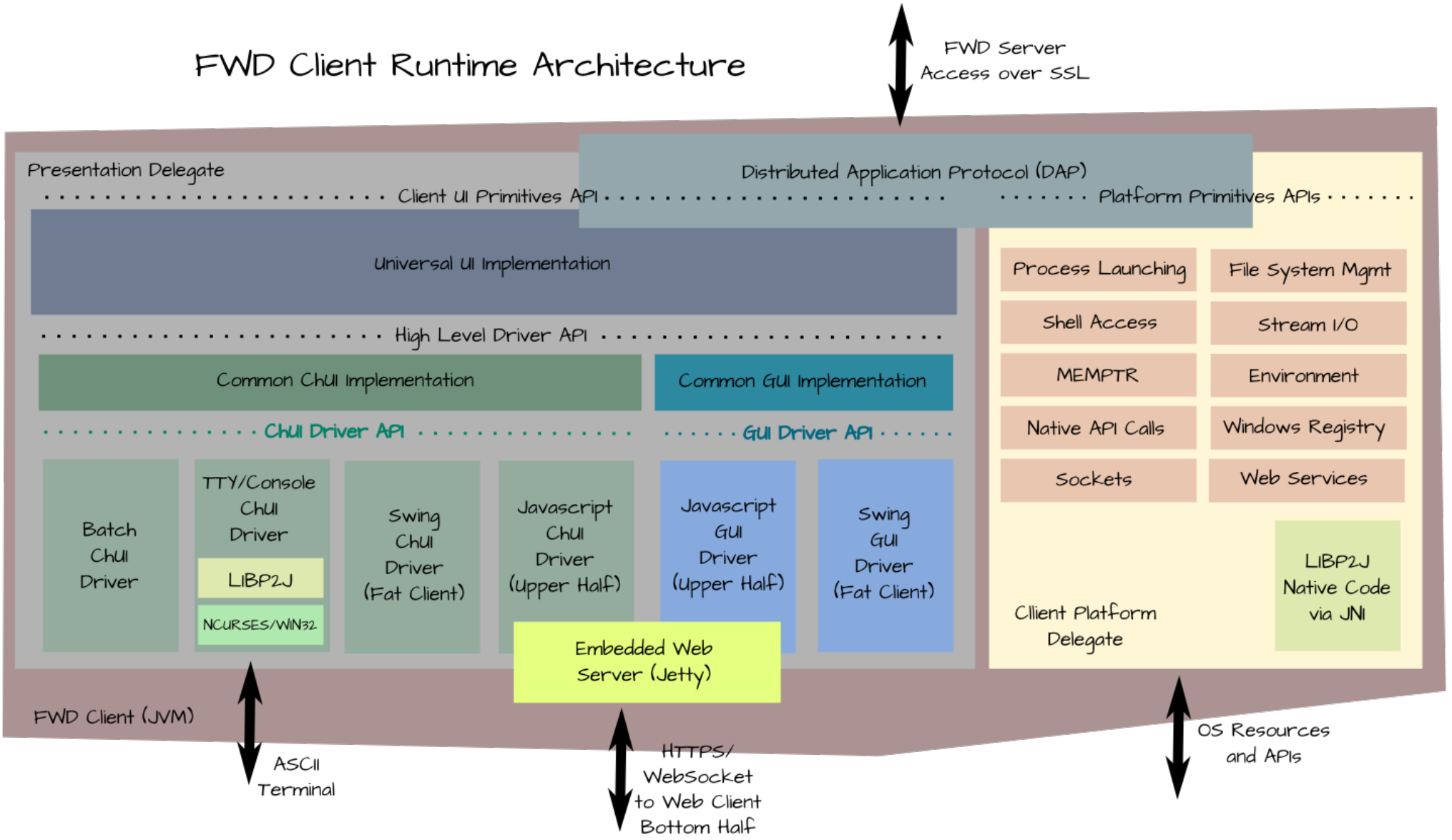
Javascript "Up-Calls"

- Create a custom web UI
- Use any browser-side Javascript tooling, controls or frameworks
- Direct calls from Javascript code to converted ABL
- Call external procedures, optionally using persistent procedures
- Internal procedures and functions can be invoked using a persistent procedure handle
- Data returned as JSON
- Easy to use data binding techniques (d3 is shown here)

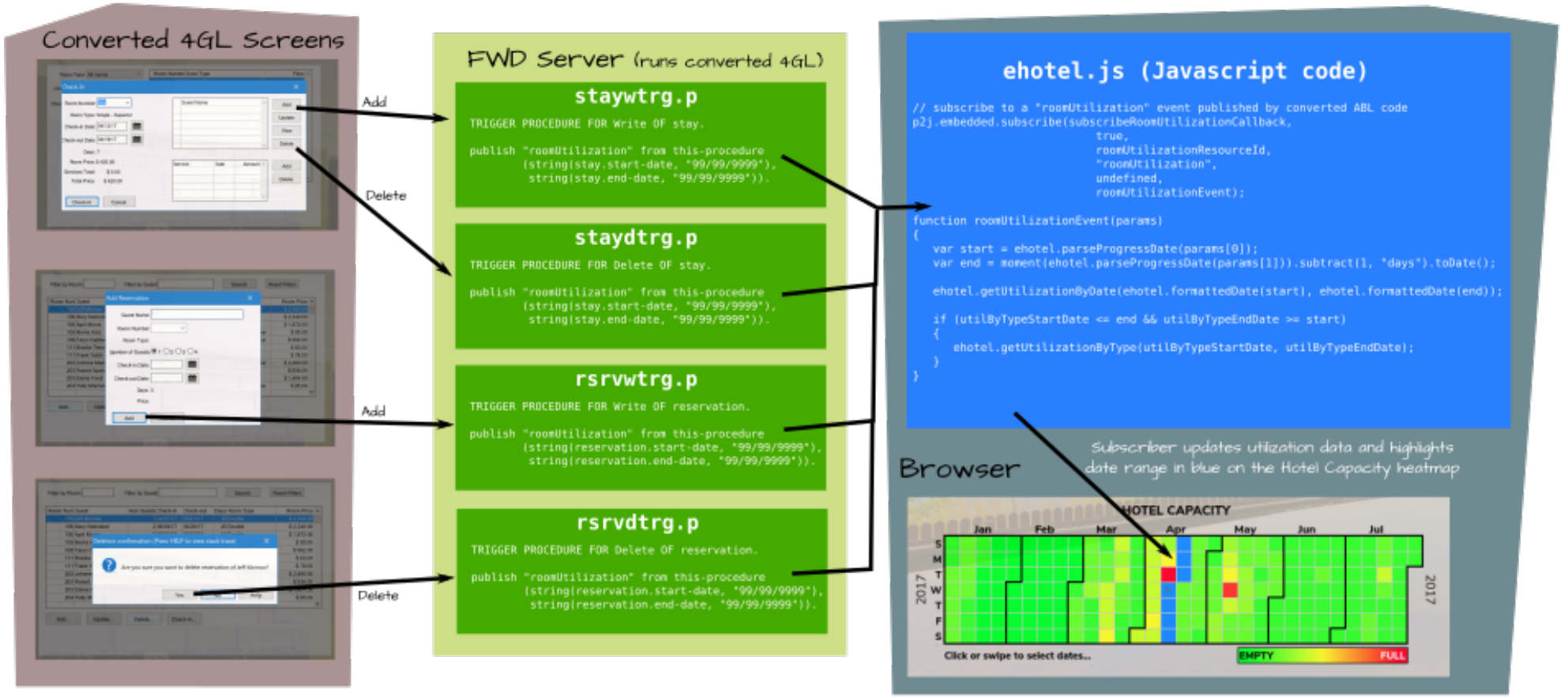


FWD Runtime Architecture

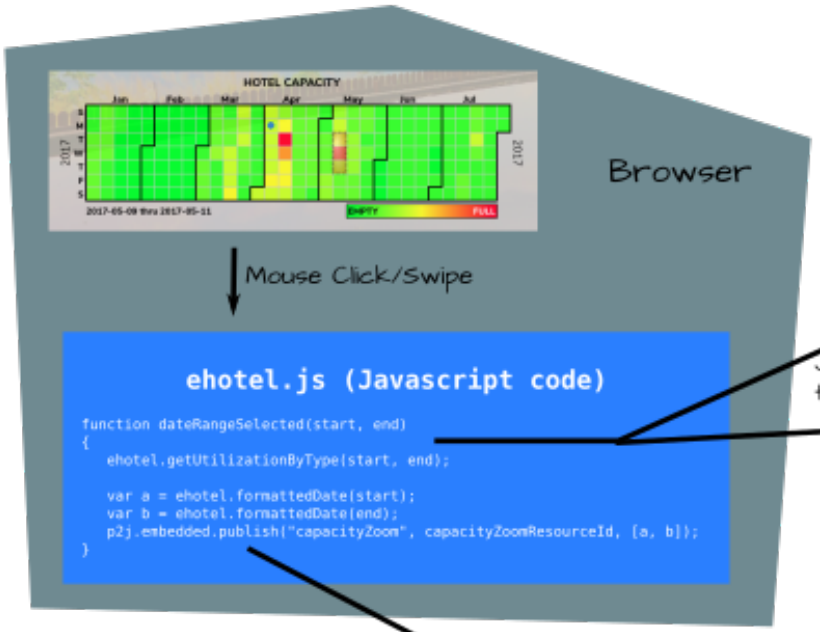



FWD Client Runtime Architecture
FWD Server
Access over SSL
Presentation Delegate
Distributed Application Protocol (DAP)
Client UI Primitives API
Platform Primitives APIs
Universal UI Implementation
High Level Driver API
Common ChUI Implementation
Common GUI Implementation
ChUI Driver API
GUI Driver API
Batch
ChUI
Driver
TTY/Console
ChUI
Driver
LIBP2J
NCURSES/WIN32
Swing
ChUI
Driver
(Fat Client)
Javascript
ChUI
Driver
(Upper Half)
Javascript
GUI
Driver
(Upper Half)
Swing
GUI
Driver
(Fat Client)
Embedded Web
Server (Jetty)
Process Launching
File System Mgmt
Shell Access
Stream I/O
MEMPTR
Environment
Native API Calls
Windows Registry
Sockets
Web Services
LIBP2J
Native Code
via JNI
Cllient Platform
Delegate
FWD Client (JVM)
ASCII
Terminal
HTTPS/
WebSocket
to Web Client
Bottom Half
OS Resources
and APIs

Javascript SUBSCRIBE "roomUtilization"



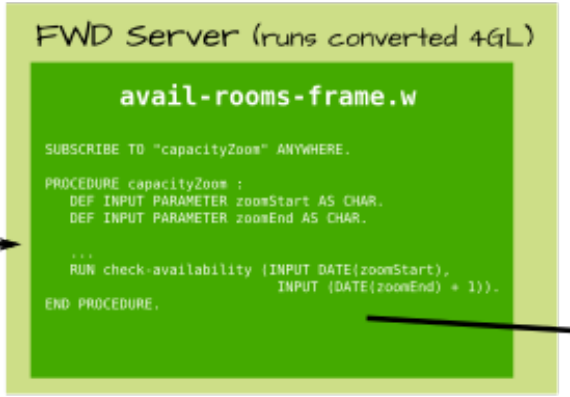
Javascript PUBLISH "capacityZoom"

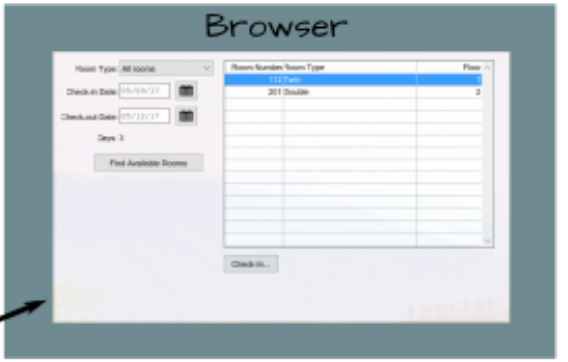


Java Up Call to get data to refresh aster plot

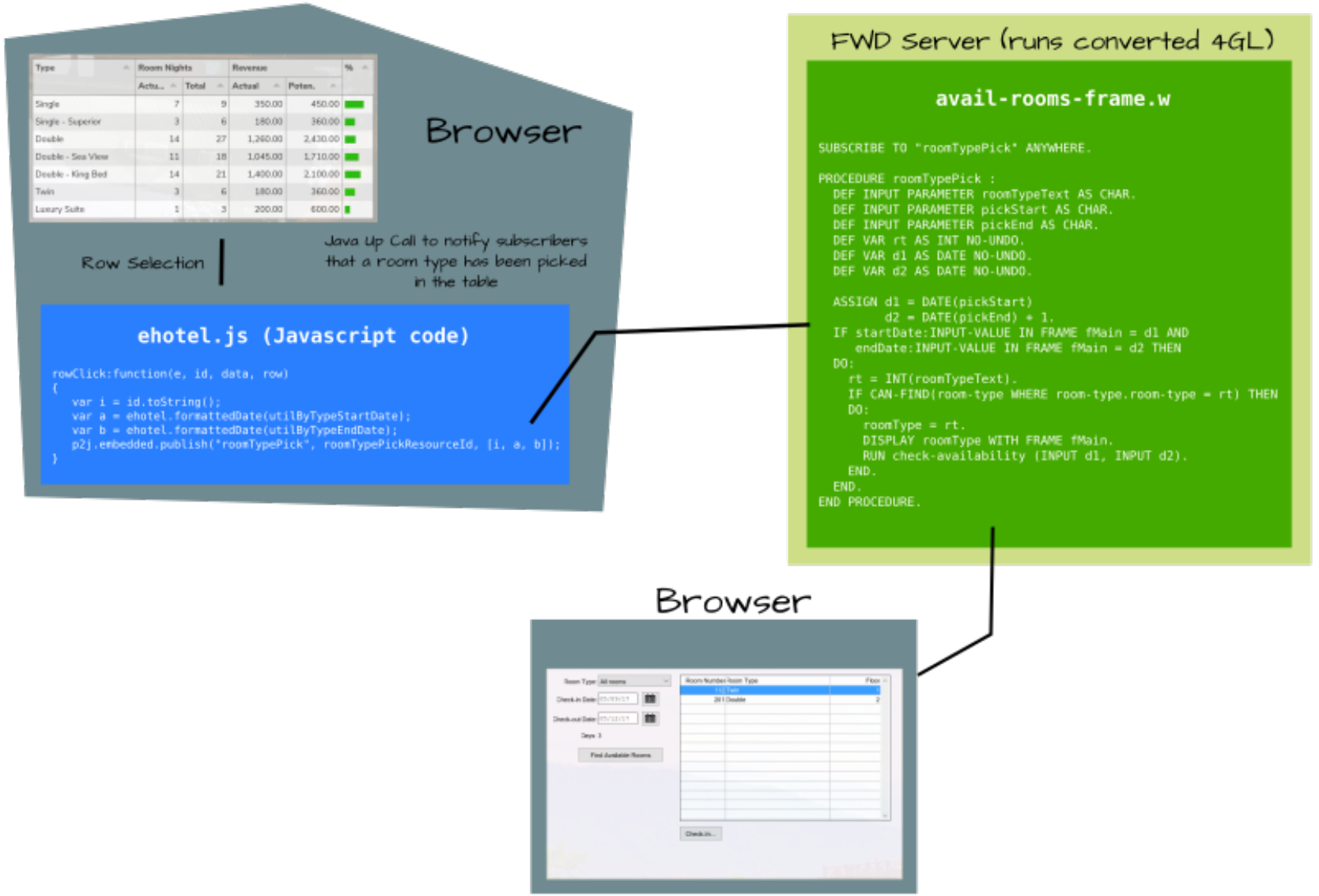
Java Up Call to get data to refresh Room Utilization table

Java Up Call to notify subscribers that a date range on the Hotel Capacity heatmap was selected

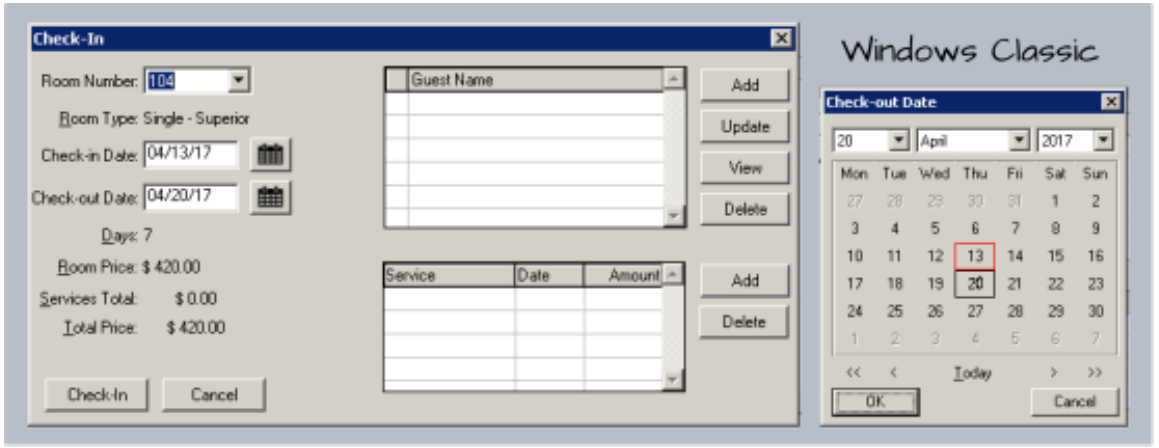


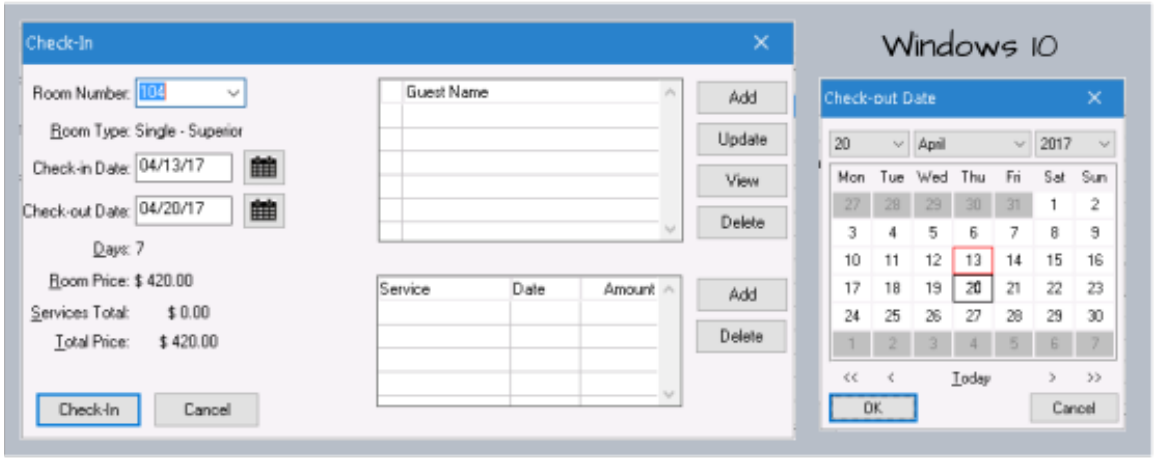


Javascript PUBLISH "roomTypePick"



Custom Themes





- Complete control over widget drawing, at the pixel level
- Control over colors and fonts
- Each widget type has its own drawing routines, override at a very "granular" level
- Pluggable Java theme classes
- Subclass the built-in themes and override the parts you need
- Build your own from scratch
- Choose the theme at runtime
- Different FWD client sessions can run different themes

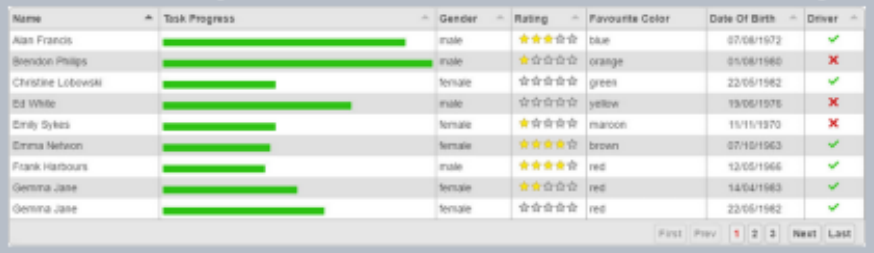
Better Widgets

Improve Existing Widgets (browse is top priority)

Widget Managed Column Sorting

Header Formatting and Control

Widget-Managed Column Filtering



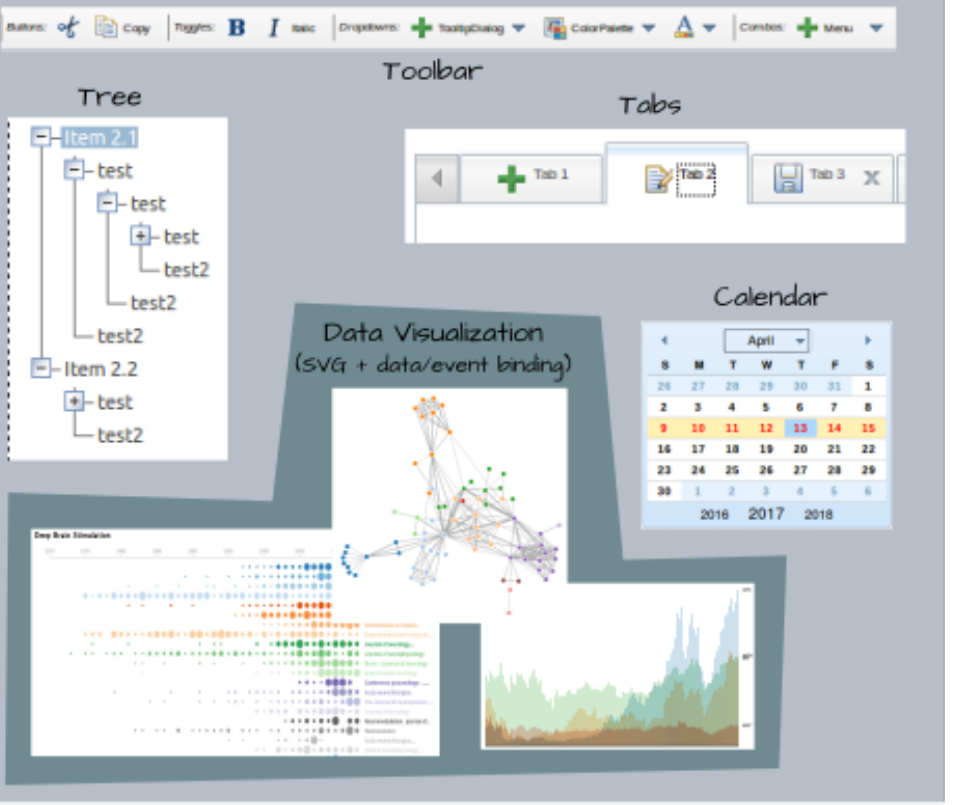
Cell-Level Hyperlinking

Pagination Controls

Direct Export to PDF/CSV/Excel

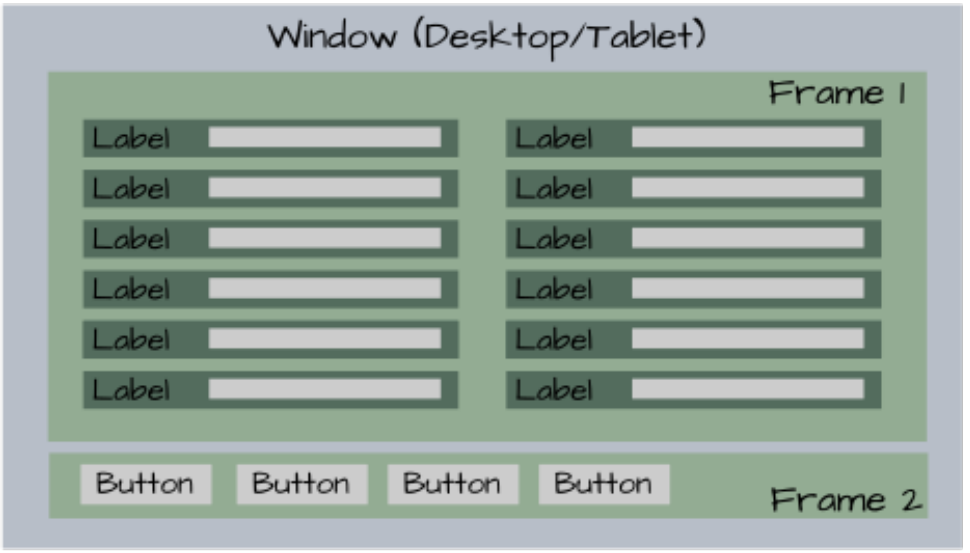
- Direct use as 4GL syntax extensions (e.g. VIEW-AS CALENDAR or CREATE TREE-VIEW)
- New widgets are first class 4GL widgets
- Enhanced versions of widgets optionally enabled at runtime (by cfg) or by use of new 4GL keywords
- Not yet available (this work is in process now), the images are for illustration only

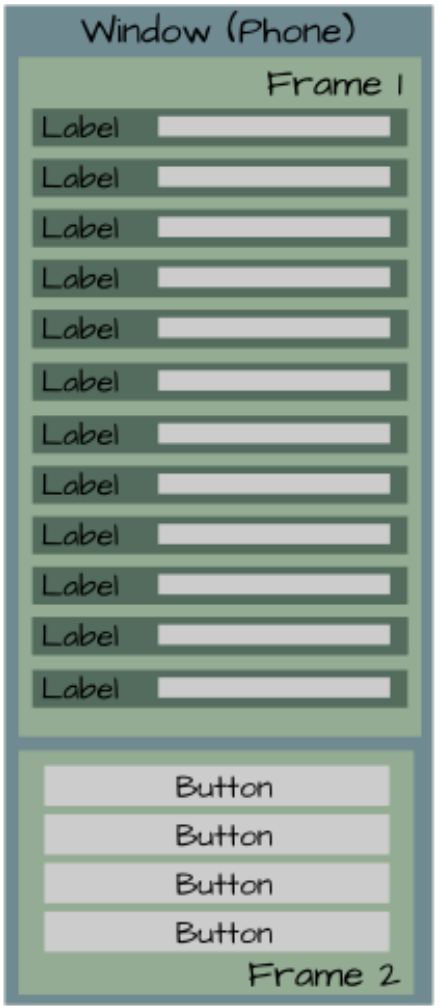
Add New Widgets



Dynamic Layout

- Encode a layout strategy at the window and frame level in 4GL code (e.g. LAYOUT EQUAL-COLUMNS or LAYOUT CARD)
- Dynamic layout and resizing of a window (and its contained frames) in response to device media queries (the screen size/resolution determines how the layout renders)
- In this example, a multi-column layout on a wider screen will naturally render in a single column on a phone screen - same 4GL code, different runtime result
- Not yet available (this work is in process now)





Other Benefits

- Portability (platform and database)
- Easy integration with any Java technology
- 4GL syntax enhancements, e.g.:
  - CREATE TIMER (instead of PSTimer OCX)
  - CREATE REPORT (JasperReports integration)
  - CREATE SMTP-EMAIL
  - Multithreading (coming soon)
  - ...
- Cloud-friendly

Status

- Everything demonstrated today available now
- Themes support available today
- Enhanced browse available by the end of 2017
- Other widget improvements starting early 2018
  - new widgets
  - dynamic layout
  - etc…

Open Source

- Solve your own problems
- See how it all works
- Collaborate / contribute
- Available now and forever
- Affero GPL (reciprocal)
- Dual licensing available

Find Us On the Web!

- www.beyondabl.com
- facebook.com/beyondabl
- twitter.com/beyondabl
- plus.google.com/+beyondabl
- linkedin.com/company/fwd-project
- youtube.com/channel/UCk3pga7EKxAQVOV_CiYOR7g