



BREW™ Development Best Practices

Bob Fitterman, CTO

Vindigo Studios

June 9, 2004

Overview

- › **Specific development ideas**
- › **Staying on schedule**
- › **Strategies for testing applications**
- › **Cross-device support**



I Wish I Knew This 3 Years Ago

The Illusion of Speed

- › **Pre-fetch anticipated content**
- › **Download bundle of data to process locally**
- › **Cache responses, especially for hierarchical navigation**
- › **Process incoming data as streams rather than waiting for the last byte to arrive**
- › **Until EV-DO is pervasive, this will matter**





Squeezing More into Less Space

- › **Consider using THUMB instruction set**
- › **Our experience**
 - **Code size reduced by 30%**
 - **No significant performance hit**
- › **Can use on individual source files**
 - **May want to use ARM for compute-intensive portions**
- › **Need to specify compile and link options**



Consider Using C++

- › **Modularity supports code reuse**
 - OO approach encourages this
 - Yes, you can do this without C++
- › **Exceptions are not supported**
- › **Templates and virtual functions will bloat code**
- › **Overload new, new[], delete, delete[] operators**
- › **You can migrate to this gradually: Mix C/C++**



Code Reuse Strategies

- › **Classes/libraries of software you develop**
 - Depends on the range of applications
 - Need to structure code into separate modules
- › **Small object modules can be beneficial**
 - Use linker options to drop out unreferenced code



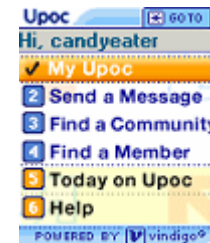
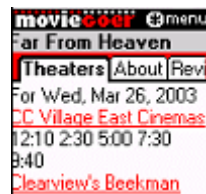
Private Extensions

- › **Takes reuse to an extreme case**
- › **Reduces apparent size when consumer installs second application using same extension**
- › **Enforces modular structure because you need to define an interface**
 - **We have focused on event-driven models**
- › **TRUE BREW[®] Testing submission is more involved**
 - **Simpler than public extension requirements**
 - **Be sure you are explicit in your directions when you submit for TRUE BREW Testing**



Code Reuse to the Max

- › For classes of applications, you can build an extension that will support multiple products
 - Information apps might use a browser-like client
 - Classes of games may all run via a common engine
 - You must analyze problem set you want to address





Code Reuse to the Max

- › **Trade-offs to consider**
 - **Data-driven clients are harder to create and debug**
 - **Big ROI when you release multiple applications**
 - **Reduced likelihood of TRUE BREW Testing failures**
- › **Challenge: Take the oath to freeze your code**



Handset Debug Tools

- › **### sequences**
 - ### 2 net debug tool
 - ### 3 available memory
 - ### 0 Turns off debug displays
- › **Key in codes *after* BREW environment starts**
- › **Some handsets use * * * or other sequences to activate**

RTFM

- › **Read the documentation**
- › **Read the release notices**
- › **Search the FAQs and Knowledge Base**
- › **Look in the BREW Forums**
- › **Reread the documentation**
- › **This will save you time and tell you things your competitors don't know**



Leverage BREW Support

- › **Responsive resource at great price**
 - Free for authenticated BREW developers

- › **Need to be specific about the problem**
 - Specify the hardware and software versions
 - Include a demo application or precise steps to replicate the problem

- › **Research first, then write**



The Clock is Ticking...



Our Approach

- › **We deliver software every 3 weeks**
 - **Version “n” under development**
 - **Version “n-1” in QA**
 - **Version “n-2” is released**
 - **We work on 1-week cycles scheduling work**

- › **We build client-server applications**
 - **Continue to update server code against frozen client**
 - **Typically build server for new app in 3-9 weeks**
 - **Release new client every 4-6 months**
 - **Client supports plug-ins**
 - **New features without complete rebuild of client**



Our Approach (cont.)

- › **We are proponents of agile development**
 - Our discipline is Extreme Programming (XP)
 - See www.extremeprogramming.org
- › **Frequent releases checkpoint working code**
 - Small incremental feature changes
- › **Heavy use of automated testing**
 - Far beyond what's described here
- › **Continuous code integration**



Use Common Sense

- › **Normal project management strategies apply:**
- › **Look for high risks and address them early**
 - **Don't leave the hard stuff to the end**
- › **Have regular interaction with your customer**
- › **Don't assume everything works as it should**



Test on Handsets Regularly

- › **Emulator/Simulator is a great tool, but**
 - Too optimistic in its behavior
 - Everything works
 - First place you should test

- › **Test on several different handset models**
 - Minor handset variations can be significant

- › ***Do this throughout the cycle***
 - ***Avoid surprises at the end of your project***



Testing Zealots on the Loose



Testing is in Our Blood

- › **Agile development encourages automated testing**
- › **Thousands of automated unit tests for our server code developed over 4+ years**
- › **Always looking for ways to build automated tests**
 - **Unit tests, load tests, regression tests, even tests of vendor APIs**



Acceptance/Regression Tests

- › **Server changes can break deployed product**
 - Choice 1: Freeze server code
 - Choice 2: Test the interaction constantly
- › **Wanted scripted acceptance/regression tests**
 - Sought end-to-end testing
- › **Poor man's solution**
 - Scripted playback of key presses via Emulator
 - Save screenshots and compare to baseline images
 - Breaks easily: very sensitive to data changes



Custom Testing Solution

- › **Selected Fitnesse, an open-source acceptance testing framework**
 - Tests are written in simple syntax
 - One test = one web page
 - See www.fitnesse.org

- › **Created “mini-implementation” of needed BREW APIs**
 - Do not update a display
 - Support inspection methods
 - Inspection methods linked to scripting
 - Only need small subset of BREW calls

Test Steps



› Typical Test

1. Start application
2. Press zero
3. Assert you have a menu with 5 items
4. Assert title says “Pick location”
5. Assert third menu item says “By neighborhood”



› Initial investment: 2 people for one month

› ROI is immeasurable

› We run over 750 automated regression tests nightly across 15 released apps



TRUE BREW Testing

- › **You must go through the entire test plan**
 - Tedious, but there is no substitute
- › **If you change any code, revisit the test plan**
- › **Use a QA team to keep yourself honest**
- › **Be thorough**



Too Many Handsets, Too Little Time



Cross-Device Considerations

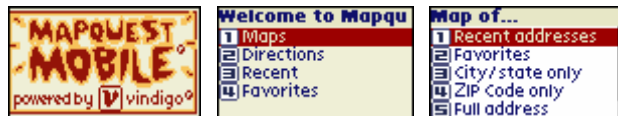
- › **Leverage your development investment by supporting multiple handsets**
- › **Compute screen layout dynamically**
 - Consider font metrics, screen dimensions, etc.
- › **Test your implementation across handsets**
 - Simulator/Emulator will not expose device-specific nuances
 - Devices show variation
 - Test on the earliest commercial release of handset software if you can



Two Handsets, One Version

› Low-end phone

- 96 W x 54 H (landscape)
- 12-bit color depth
- No soft keys
- RAM: 400 kb
- Single key press only
- IPOSEDET unavailable



› High-end (U.S.) phone

- 240 W x 292 H (portrait)
- 16-bit color depth
- 2 soft keys
- RAM: 600 kb
- Simultaneous key press
- IPOSEDET support





Handset Variations

- › **Support device-specific capabilities as needed**
- › **Examples**
 - **ITAPI_MakeVoiceCall()**
 - **Missing glyphs in fonts**
 - **Soft key labels**
 - **Wallpaper directory**
 - **Simultaneous key press support**
 - **Handset-specific implementation differences (bugs)**
- › **Review device data sheets (DDS) and test on handsets**



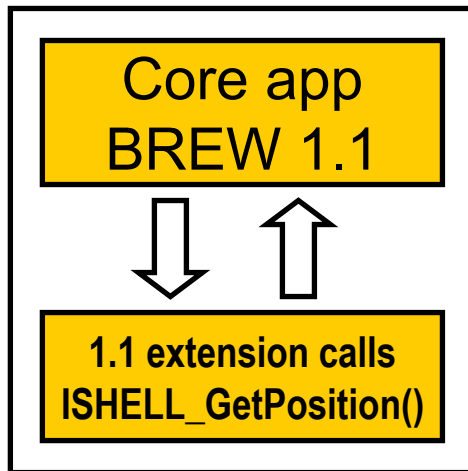
Cross-Device Strategy

- › **Use flags to configure features**
 - Set flags from server on first use, or
 - Set flags on device via resource file or table keyed on Platform ID
 - Maintain one executable across many handsets
- › **Degrade gracefully in absence of capability**
- › **Benefit: one set of source code, one executable, and potentially one package across all devices**

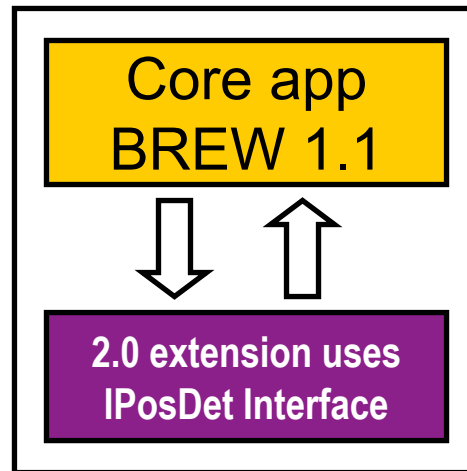


Cross-Device Example

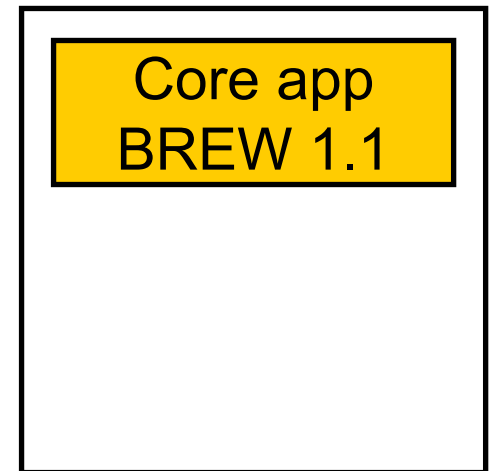
- › **GPS functionality in private extension**
 - Using app and extension communicate via events
 - Degrades gracefully in absence of extension



**Deprecated
BREW 1.1 APIs**



**Leverages
BREW 2.0 interface**



**Lacks auto-
location support**

Summary

- › **Code reuse reduces bugs**
- › **Testing on handsets is crucial**
- › **Leverage available resources**
- › **Use a QA team to keep yourself honest**
- › **Be thorough**