



BREW 2005
conference

Writing an Efficient BREW® Porting Layer

Viswanathan “Vishy” Kalyanakrishnan
Staff Engineer
QUALCOMM Incorporated

June 2, 2005

Overview

- **OEM Layer**
 - Components & Function
 - Where it fits in
 - Interfaces it deals with
- **Need for optimizing the OEM layer**
- **Operator requirements & importance**
- **Optimization techniques**
 - Task prioritization
 - Display layer
 - File System layer
 - Network operations
 - Resource files
- **Handling Memory constraints**
- **Q & A**

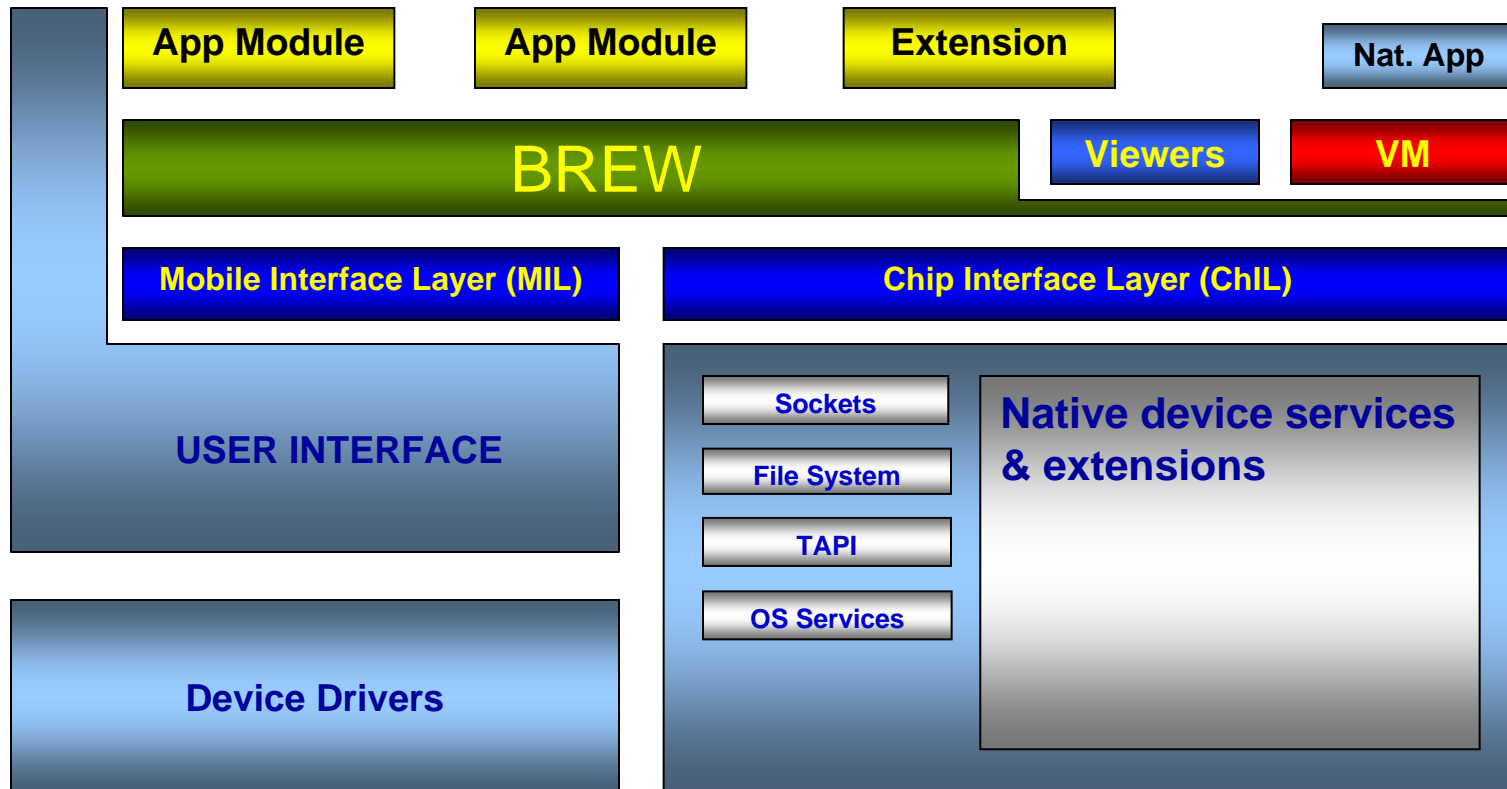


What is the OEM Layer?

- **Glue between the BREW APIs & native device functions**
- **Contains a set of portability APIs**
- **Reference version provided with the BREW Porting Kits**
- **Customizable by OEMs per device features & operator requirements**



Where it Fits



Components of the OEM layer

- **BREW AEE header Files**
- **OEM Layer header files**
- **OEM Layer source files**
- **Resource files (strings, images, themes etc.)**
- **Configuration items**
- **Libraries**
- **Data files**



OEM Layer Interfaces

- **Display**
- **File System**
- **Networking/TAPI**
- **Device configuration**
- **Media**
- **Addressbook**
- **Database**
- **Security**
- **Etc...**



Need for OEM Layer Optimization

- **Improve BREW API performance**
- **Meet operator requirements**
- **Enhanced end user experience**
- **Obtain optimal performance given device constraints**
- **Expose additional functionality unique to the device**
- **Maintain backward compatibility**



Operator Requirements & Importance

- **Features required**
- **Memory requirements**
- **Benchmark tests**
- **Hardware**
- **File system performance**
- **Networking performance**



Optimization Techniques (1 of 5)

- **Task Prioritization**

- Pick a task context
- Identify the task priority
- If no separate BREW task, recommend using UI task context
- If running as a separate task, recommend a priority that is equal to or just lower than the UI task
- Very low priority could cause BREW task starvation
- Very high priority could cause other important tasks to starve
- Analyze task interactions
- Prevent CPU hogging – potential for watchdogs to be set off
- Periodically give up CPU to other tasks



Optimization Techniques (2 of 5)

- **Display Layer**

- One of the first layers to be ported for BREW
- Significant impact on end user experience
- Hardware
 - TFT Active Matrix, TFD, DSTN, CSTN etc.
 - Display Technology & Bus Architecture affect performance
- Hardware refresh rate & device buffer refresh rates
- Optimization in display device driver software
- Support the right bit depth
- Exploit processor caches
- Dirty rectangle optimizations
- Exploit hardware decoders
- BREWStone[®] can help



Optimization Techniques (3 of 5)

- **File System Layer**

- Significant impact on device performance
- Choosing the right flash part
 - NAND – Fast erase, slow single byte write, fast multibyte write, slow read, low power consumption, needs additional RAM for code execution
 - NOR – Slow erase, fast single-byte write, slow multibyte write, fast read, high power consumptions, does not require additional memory for code execution
- Take advantage of BREW's file caching mechanism
- File system block size variations

- **Network Operations**

- Optimize TCP window size
- Implement OEMSocket_SetSndBuf() & OEMSocket_SetRcvBuf() APIs
- Use OATWeb & OATDownload tests to obtain performance statistics



Optimization Techniques (4 of 5)

- **Configuration Items**

- Provide device information & OEM configuration
- Frequently accessed by BREW (OEM_GetConfig/OEM_GetDeviceInfo)
- Typically maintained in persistent storage (NV/FS)
- Cache data for faster access
- Send out notifications on device item changes
- Maintain persistent storage & cache in sync

- **Resource File Management**

- BREW Resource files typically contain images, strings & dialogs
- New uiOne Toolkit files contain theme information
- Loaded using ISHELL_LoadResXXX() APIs
- Exploit CFGI_CACHED_RESOURCES item
- Pros – Faster access to resources
- Cons – Increased heap usage



Optimization Techniques (5 of 5)

- **Resource File Management**

- Use BMPClean tool to reduce bitmap image sizes
- Preload resources at startup
- Move resource files to code space

- **Event Notifications**

- If registering for BREW notifications, make sure they are consumed as needed to prevent unwanted additional notifications
- Investigate potential to pre-launch applications in the background to reduce application load time.



Handling Memory Constraints (1 of 4)

- **Devices such as handsets constrained on memory (RAM, ROM)**
- **ROM contains Read Only Data, Read Only Code & non zero initialized data**
- **RAM contains Read/Write data & Zero initialized data**
- **BREW libraries & OEM layer consume RAM & ROM**
- **Identify minimum set of BREW features required**
- **Typically enabled/disabled via OEMFeatures.h**
- **Unlink libraries & remove code that is not required**
- **Re-use libraries (e.g., a PNG image decoder on the native side can also be used by BREW OEM layer)**



Handling Memory Constraints (2 of 4)

- **Inspect OEM Layer code & re-write more efficient code if needed**
- **Low File System Space & Ample ROM**
 - Re-zone the memory banks & access the RO data as files from the file system rather than directly from an array pointer
 - Consumes additional ROM but frees up file system
 - Upgrade ROM chip if a PCB rework is not needed & the device is still in early development stage



Handling Memory Constraints (3 of 4)

- **Low File System Space & Ample ROM Space**
 - Place data files in ROM constant data buffers
 - Rework OEM layer code to access the data as an array. This also improves the speed of execution
 - If possible, place files in ROM constant data buffers & setup the device initialization routines to create these as remote link files
 - Allows a named file access to the array data with file system commands
 - OEM layer code uses standard file system calls to access data & data comes faster from ROM



Handling Memory Constraints (4 of 4)

- **Low ROM, Ample File System Space & RAM**
 - Reverse the steps mentioned in the previous slide
 - Take data array files, convert to binary & load them onto the file system directly
 - OEM layer code needs to be modified to access this data from the file system
 - Substantial ROM space saving can be achieved
- **Low ROM, Low EFS & Low RAM**
 - Cut down on BREW features & capabilities to fit in
 - Analyze BREW feature interdependencies to identify all possible candidates for removal



Other Items

- **Exploit any internal cache provided by the processor**
- **Use applications like BREWStone[®], OATWeb & OATDownload during development to measure performance & optimize**
- **Understand operator requirements clearly so that you can actually take advantage of them**
- **Remember to maintain backward compatibility on OEM extensions also**





Q & A