

# BREW<sup>®</sup> Application Framework

**Mahesh Moorthy**

Sr Director - Engineering

*QUALCOMM Incorporated*



# Agenda

- **What is an Application in BREW ?**
- **Application Concepts (Top-visible, Suspend, Resume, etc.)**
- **Application History**
- **Background Applications**
- **Detailed call-flow sequence of starting & stopping Applications**
- **New APIs in BREW 3.x for application mgmt**
- **Screen Savers**

# Application - Introduction

- Any class that implements the IApplet interface in BREW is an “Application”
- Methods on IApplet interface
  - IAPPLET\_AddRef()
  - IAPPLET\_Release()
  - IAPPLET\_HandleEvent()
- Canned templates (such as AppGen, ModGen) can be used to build the basic support for an application
- Applet\_New() and HandleEvent() are key parts of an application code

# How Does BREW Recognize Apps ?

- **Each Application must belong to a Module**
- **Application information is stored in the MIF**
  - ClassID is the most important piece of App Info stored in MIF. Other info such as: Name, Icon, etc. also is in MIF
- **On startup of device (or Simulator), BREW environment is initialized. During this, the following steps are done:**
  - Enumerates all the MIFs in the system
  - Extracts information associated with all the applications specified in the MIFs
  - A single MIF can describe any number (0 or more) of applications
  - An in-memory (RAM) list of all the applications in the system is created. This list contains all the pieces of application-info extracted from the MIF (ex: ClassID)
  - All operations related to applications (enumeration, start, stop, etc.) use this list as a starting point

# Application - Key Concepts

- **Top-Visible**

- An application is said to be “Top Visible” if it has access to the primary display and has keypad focus
- Only one application can be “Top Visible” at any instant of time
- When an application (say, HelloWorld) is launched (say from AppMgr), that application (HelloWorld) becomes “Top Visible”. It takes control over the primary display and is responsible for drawing to it. It directly receives all the keypad events
- ISHELL\_StartApplet() and successful handling of EVT\_APP\_START is one of the means for an application to become Top Visible

# Application - Key Concepts (Continued)

- **Suspend**

- When the current Top-Visible application changes, the previous top-visible application is “Suspended”
- A suspended application does not have access to the Primary Display and does not have keypad focus (i.e. does not directly receive keypad events)
- All other operations (not relating to primary display or keypad) can be carried out by the Suspended application (ex: timers, sockets, etc.)
- No limit to the number of applications that can be in the “Suspended” state
- EVT\_APP\_SUSPEND is sent to the app by BREW, when it needs to be suspended
  - Not handling this event (i.e if FALSE is returned), will cause the application to be stopped i.e. application data is released from memory and app is potentially off-loaded from memory

# Application - Key Concepts (Continued)

- **Resume**

- When the current Top-Visible ceases to be top-visible, the most recently suspended application is “resumed”
- Resuming an application means, requesting a previously suspended application to become top-visible again
- EVT\_APP\_RESUME is sent to the app by BREW, requesting it to Resume. (If the app did not handle Suspend successfully, EVT\_APP\_START is sent instead)
- When an application resumes successfully (returns TRUE to the event), it becomes “Top Visible”)
- Unlike “Top Visible”, or “Suspend”, Resume is not an application state. Resume is an action sent to an app by BREW. It immediately transitions the app to the Top Visible state

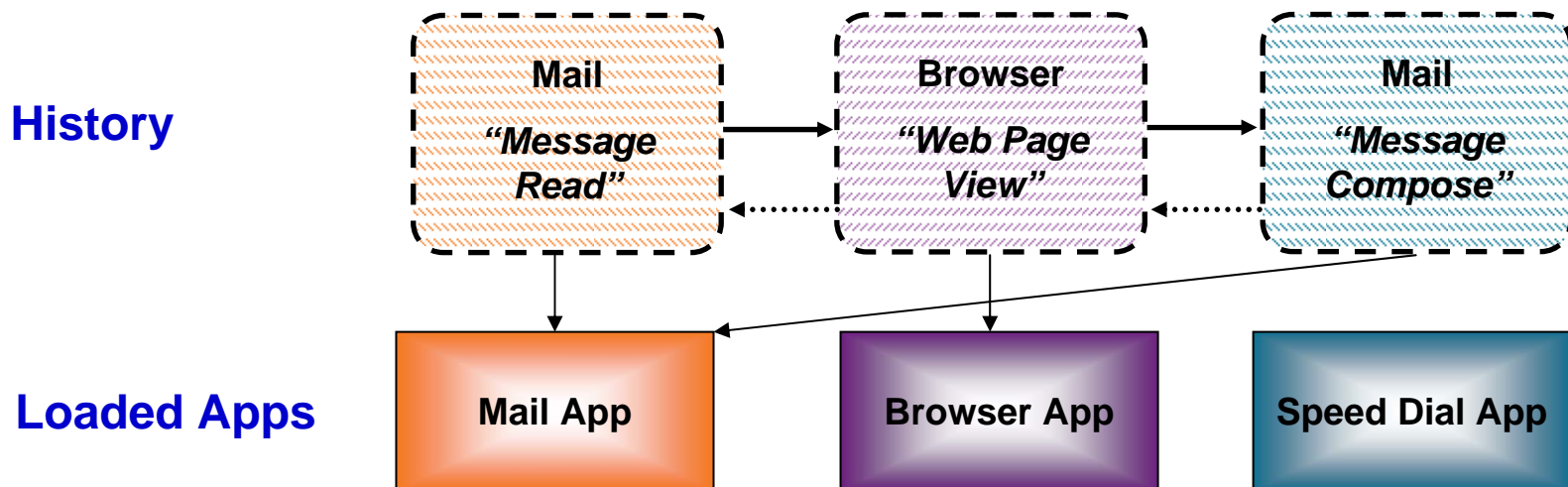
# Application - Key Concepts (Continued)

- **Application History**

- This is a list containing the Top Visible app and the set of suspended applications
- This list maintains the order (the application stacking order) in which apps were made top visible and suspended
- The Top Visible app is at the top of the list and the most recently suspended application is closest to the top and so on...
- When the current Top Visible application ceases to be top visible, this list is consulted for the next application to be Resumed
- While the concept of Application History and stacking order always existed in BREW, a new public API was introduced in BREW 3.x for accessing the history list. The API is called IAppHistory

# Application History

- A single application can occupy multiple spots in the History List
- Application History allows an applet to load once and maintain separate data for UI contexts
- Example
  - Email application module loaded once: History 1 maintains read state, History 2 maintains message composition state, etc.



# Application History

- **Allows State data to be stored along with the history entry**
- **The API IAPPHISTORY\_SetResumeData() can be used to store state data along with a history entry**
- **IAPPHISTORY\_GetResumeData() can be used to retrieve the state data**
- **Sequence of events to the Mail app in previous example**
- **Step 1: Launching the Mail app for “Mail – Message Read”**
  - New History entry created for Mail App
  - Mail App receives EVT\_APP\_START
- **Step 2: Transitioning from “Mail App – Message Read” to “Browser”**
  - New History entry created for Browser App
  - Mail App receives EVT\_APP\_SUSPEND
  - Mail App to use IAPPHISTORY\_SetResumeData() to store “Msg Read” related state information

# Application History

- **Step 3: Transitioning from “Browser” to “Mail – Message Compose”**
  - New history entry created for Mail App
  - Mail App receives EVT\_APP\_RESUME
  - Calling IAPPHISTORY\_GetResumeData() will return NULL since this is a new history entry
- **Step 4: Quitting “Mail – Message Compose”**
  - Browser App receives EVT\_APP\_RESUME
- **Step 5: Quitting “Browser”**
  - Mail App receives EVT\_APP\_RESUME
  - Mail App can use IAPPHISTORY\_GetResumeData() to retrieve the state data for “Msg Read”
  - Mail App becomes Top Visible
- **The key point to note is that there is only one copy of the application loaded in memory and there is only one copy of the applet data structures. State-specific information is stored in the history entry**

# Background Applications

- **Background Applications are not part of the Application History**
- **These applications are not part of the application stacking order**
- **No access to Primary Display and Keypad**
- **No limit to number of background applications**
- **Used primarily for performing non-UI related operations such as: message handlers, music downloads, etc.**

# Background Applications

- **How to cause an application to become background ?**

- **BREW 3.x:**

- ISHELL\_StartBackgroundApplet()
- On calling the above API, the event EVT\_APP\_START\_BACKGROUND event is sent to the background app by BREW. This event is sent instead of the normal start event EVT\_APP\_START

- **BREW 2.x:**

- Step 1: Call ISHELL\_CloseApplet() on the app that needs to be become background
- Step 2: On receiving EVT\_APP\_STOP, set \*(dwParam) to FALSE  
(\* (boolean \*)dwParam) = FALSE;

# ISHELL\_StartApplet() – Call Flow

- API used to cause an application to become top-visible
- This API has two portions to its execution – Synchronous Portion followed by Asynchronous portion
- The process of starting an applet is complete only after the Asynchronous part has finished
- Steps in Synchronous Portion of ISHELL\_StartApplet()
  - Check if the application Class ID is valid and is supported
  - Validate the license of this app and ensure it has not expired
  - Check with the Resource Arbiter to see if it is OK to allow this app to Start
  - If success, en-queue the asynchronous portion. No limit to the number of start-applet-requests that can be enqueued
  - Return error code (EBADCLASS, ENOMEMORY, ECLASSNOTSUPPORT, EEXPIRED, EFILENOEXISTS, SUCCESS)

# ISHELL\_StartApplet() – Call Flow (Continued)

- **Steps in Asynchronous Portion of ISHELL\_StartApplet()**
  - Suspend the current top-visible app
  - Associate a history entry for this app
    - If no entry exists for this app class ID in the history list, create a new history entry
    - If an entry already exists for this app and if the app did not SetResumeData(), move the history entry to top of the list. If it has set the ResumeData(), create a new history entry. This is done to preserve backward compatibility with BREW 2.x
  - Send EVT\_APP\_START to the new top-visible app
  - Verify if the new top-visible successfully handled EVT\_APP\_START. If not, close this app and resume the previous top-visible app
  - Notifications of type NMASK\_SHELL\_START\_STATUS to indicate successful or failed starting of apps

# Variants of ISHELL\_StartApplet

- **ISHELL\_StartApplet(pShell,clsApp)**
  - Sends EVT\_APP\_START to the app. dwParam is of type AEEAppStart\*
  - Makes the app Top Visible
- **ISHELL\_StartAppletArgs(pShell,clsApp,pszArgs)**
  - Sends EVT\_APP\_START. dwParam is of type AEEAppStart\* and contains the arguments pszArgs
  - Makes the app Top Visible
- **ISHELL\_BrowseURL(pShell,pszURL)**
  - Looks for Applet Class that handles the URL scheme specified by pszURL
  - Ex: ISHELL\_BrowseUrl(ps,"cmshop:browse"). Looks for Application that handles the "cmshop" URL scheme
  - Sends EVT\_APP\_START to the app
  - Sends EVT\_APP\_BROWSE\_URL to the app.dwParam points to the pszURL passed when ISHELL\_BrowseURL() is invoked
  - Make the app Top Visible

# Variants of ISHELL\_StartApplet

- **ISHELL\_BrowseFile(pShell,pszFile)**

- Looks for Applet Class that handles the File extension specified by pszFile
- Ex: ISHELL\_BrowseFile(ps,"foo.ini"). Looks for Application that handles the "ini" File Extension
- Sends EVT\_APP\_START to the app
- Sends EVT\_APP\_BROWSE\_FILE to the app. dwParam points to the pszFile passed when ISHELL\_BrowseFile() is invoked
- Make the app Top Visible
- pszFile must specify the complete name and path of the file

# Top Visible vs. Running Applet

- **Currently Running Applet**
  - Applet that is currently executing
  - May or may not be the same as the Top Visible applet
  - For ex: imagine a case where a timer Callback has just been dispatched to a background application (or a suspended application)
  - In the above example, while inside the timer callback, the bkgnd app (or suspended app) is the currently running app
  - All operations (such as: privileges, file-path-access, etc. are enforced based on the currently running application)
- **ISHELL\_ActiveApplet() returns the ClassID of the Top Visible app**
- **IAPPLETCTL\_RunningApplet() returns the ClassID of the currently running applet**

# ISHELL\_CloseApplet

- **ISHELL\_CloseApplet(pShell,bReturnToldle)**
  - API to close the currently running applet
  - Does not take a classID as parameter
  - Can be invoked on any running app (TopVisible, Suspended, Background)
  - The operation is Asynchronous
- **Closing an app that is in the History List**
  - The top most history entry for this application class ID is removed
  - If there are no more history entries for this app, the app receives EVT\_APP\_STOP
  - If there are more history entries for this application classID, the application is suspended i.e it receives EVT\_APP\_SUSPEND
    - wParam of this event is set to AEE\_SUSPEND\_CLOSE to indicate that ISHELL\_CloseApplet() has been called on this app and that there are pending history entries for this app

# ISHELL\_CloseApplet

- **Closing an app that is NOT in the History List**
  - Application receives `EVT_APP_STOP`
- **If `CloseApplet()` is called on the current top visible app, the next application in the history list is resumed and made top visible**
- **Note: Other wParam values to the Suspend Event `EVT_APP_SUSPEND` are:**
  - `AEE_SUSPEND_NORMAL` or `AEE_SUSPEND_EXTERNAL`
    - `AEE_SUSPEND_NORMAL` is used when this app is suspended due to another app starting
    - `AEE_SUSPEND_EXTERNAL` is used when an app is suspended due to an external event (such as: Suspending of BREW itself)

## ISHELL\_CloseApplet – Close All Applets

- **Calling ISHELL\_CloseApplet() with the second parameter (bReturnToldle) set to TRUE will send a request to BREW to close all the applications in the Application History**
- **PL\_SYSTEM Privilege is required to close all applications**
- **When closing all, there is no impact to background applications. Only applications in the Application History (i.e. top visible and suspended applications) are closed**
- **While BREW is in the process of processing “close all”, no requests to ISHELL\_StartApplet() will be entertained. Also, all pending requests to StartApplet() that are enqueued will be cancelled**

# Application States

- **The various states that any application can be in are:**
  - **APPSTATE\_TOP\_VISIBLE:** The app is currently top Visible
  - **APPSTATE\_SUSPENDED:** The app is currently suspended
  - **APPSTATE\_BACKGROUND:** The app is running in the background. It is not in the application history list
  - **APPSTATE\_STARTING:** The app is in the process of starting
  - **APPSTATE\_CLOSING:** The app is in the process of closing
  - **APPSTATE\_STOPPED:** The app has been stopped. This is the default state in which any app is created. From here, it migrates to one of the states mentioned above

# Summary of Application Events

<u>Event Name</u>	<u>Event Description</u>
EVT_APP_START	When an app tries to become top-visible
EVT_APP_STOP	The app calls ISHELL_CloseApp() on itself
EVT_APP_SUSPEND	Sent to current top-visible before a different app is made top-visible
EVT_APP_RESUME	Send to a suspended app to make it top-visible
EVT_APP_BROWSE_URL	Sent after EVT_APP_START with dwParam containing the URL with which ISHELL_BrowseURL() was invoked
EVT_APP_BROWSE_FILE	Sent after EVT_APP_START with dwParam containing the name of the file passed to ISHELL_BrowseFile()
EVT_APP_POST_URL	Event posted to the app without sending EVT_APP_START. dwParam contains the URL
EVT_APP_START_BACKGROUND	Sent to an app requesting it to start in the background
EVT_APP_NO_CLOSE	Sent to current top-visible before starting Screen Saver

# AEEAppInfo Structure

- **This is a data structure that contains application information**
  - Contains the ClassID of the app
  - Contains the name of the MIF of the app
  - Resource ID for locating Name, Icon, Settings, Version, etc. of the app
  - Standard Macros exist to determine offsets of the application resources
    - APPR\_NAME(ai)
    - APPR\_ICON(ai)
    - APPR\_IMAGE(ai)
    - .....
  - AEEAppInfo for an app can be obtained using ISHELL\_EnumNextApplet()
  - Applications can read from their MIF file, just like reading from any other BREW resource file
  - Example:
    - ISHELL\_LoadResString(pS, ai.pszMIF, APPR\_NAME(ai),...)
    - (ai is the variable holding AEEAppInfo information for this app)

# Looking for an Application

- **To check if a specific application (of a given ClassID) is supported in the system:**
  - ISHELL\_QueryClass(pShell,clsApp,&ai);
  - ai is AEEAppInfo
  - If the class is supported in the system and if it is an applet, this function returns TRUE and the AppInfo is filled in the structure passed as last argument to this function
- **Application Enumeration**
  - ISHELL\_EnumAppletInit()
  - ISHELL\_EnumNextApplet()
  - Has the limitation that if the application list changes between these two calls, accurate information is not returned
  - Listen to EVT\_MOD\_LIST\_CHANGED (or : NMASK\_SHELL\_MOD\_LIST\_CHANGED notification) to be notified of changes to available applications in the system

# Common Rules When Dealing with Applications

- **Never call ISHELL\_CreateInstance() on an application class ID**
- **Applications must be started only using the APIs provided for application management**
- **Directly creating an application classID will not provide any of the benefits of application management that BREW offers (ex: top-visible, application history, etc.)**
- **On Suspend, clean up as much of the resources as possible, so that they may be utilized by the newly top-visible app**
- **Return FALSE to any un-handled event in the HandleEvent() function of the app. This allows BREW to do default processing in the case of un-handled events (ex: CloseAppKey i.e. CLR key closes the currently running app)**

# Common Rules When Dealing with Applications

(Continued)

- **Avoid having huge background applications that are always running. They take up too much resources if the application size is big**
- **If “always running bkgnd apps” is a must for your design, consider splitting the bkgnd app into two or more portions and keep the portion that has to always run (say the one that listens to incoming messages) small**
- **If your application takes up lot of heap or file space, register for the useful system callbacks such as: `AEE_SCB_LOW_RAM`, `AEE_SCB_LOW_STORAGE` etc. that are delivered by BREW when the system goes low on RAM or file system space respectively. In these callbacks, do as much cleanup as possible**

# New APIs in BREW 3.x for Application Mgmt

- **IAppHistory**

- APIs for traversing the history list
- Insert / Move / Remove history entries
- Get / Set Resume Data
- Get / Set Data (can be used by the applications to store any arbitrary named-data with the history entry)
- Certain operations require special privileges (examples of privileged operations: Insert, Getting Info on history entries belonging to other apps, Moving history entries of other apps, etc.)
- Some operations are synchronous (ex: Get/Set Data, etc.) while some are not. For ex: Moving a history entry to top-visible is asynchronous while moving a history entry within the list without impacting the top-visible is asynchronous

# New APIs in BREW 3.x for Application Mgmt

- **IAppletCtl**

- APIs for Starting / Closing Apps
- ISHELL\_StartApplet() and ISHELL\_CloseApplet() continue to be supported for backward compatibility
- Get statistics for an app such as: application state, start time of the app, heap usage of the app, AppletInfo (AEEAppInfo) of the app, Class ID of currently running app, etc.

# Screen Savers

- **The idea is for BREW to cause a “designated” app to become top-visible when there is a period of inactivity on the device**
- **The designated app referred to above is the Screen Saver**
- **The default period of inactivity for SSaver to start is 30 seconds. This is configurable by OEMs**
- **Following operations reset the timer of inactivity period:**
  - **Key Events, Pen Events, Joystick Events, change in the current top-visible app**

# Screen Savers (Continued)

- **How does BREW identify the Screen Saver App**

- The app that is registered with the BREW registry as handler for Mime Type: “brew/ssaver” with AEECLSID\_SCREEN\_SAVER as the base class, is identified by BREW as the Screen Saver
- There can be only one registered Screen Saver App in the system
- The ScreenSaver flag in the MIF for the app only indicates that this app is capable of being a Screen Saver. It does not register with the BREW registry
  - This flag is used by the BREWAppMgr application to list all the available screen savers in the device, allowing the user to select the desired ScreenSaver
  - When user selects the desired Screen Saver, BREWAppMgr registers this app as the designated Screen Saver using the mime type and base class as described in the first point above

# Starting a Screen Saver

- **Here are the flow of events when a Screen Saver has to be started**
  - Step 1: BREW determines that the period of inactivity on the phone has exceeded the Screen Saver Timeout Value
  - Step 2: BREW sends the EVT\_APP\_NO\_CLOSE to the current top-visible app to check if it is OK to start the Screen Saver. Starting the Screen Saver will cause the current top-visible to be suspended
  - Current top visible can prevent the SSaver from starting by returning TRUE to EVT\_APP\_NO\_CLOSE
  - Step 3: If current top-visible returns FALSE to EVT\_APP\_NO\_CLOSE, it is suspended and the Screen Saver is started

# Stopping a Screen Saver

- **Here are the flow of events to stop a Screen Saver**
  - When a Key is pressed, the SSaver is stopped and the most-recent top-visible is made top-visible again
  - When a different app tries to become top-visible (by calling `ISHELL_StartApplet()`), the SSaver is stopped
  - There is no concept of Suspend for SSavers. They are stopped immediately in the two circumstances explained above

# Summary

- **Application is a class that implements IApplet**
- **Use ISHELL\_StartApplet() to become top-visible**
- **EVT\_APP\_SUSPEND / EVT\_APP\_RESUME: Events for Suspend / Resume respectively**
- **Use ISHELL\_StartBackgroundApp() for starting an app and going to the background directly**
- **Application History tracks the app stacking order. Applications should use ResumeData() in IAppHistory for storing state-specific information**
- **ISHELL\_StartApplet() and ISHELL\_CloseApplet() are asynchronous operations**
- **BREW 3.x introduced more APIs for app mgmt**
- **Screen Savers are BREW apps launched when there is a period of inactivity on the device**

**THANK YOU**