

Event Management

Apul Nahata, Staff Engineer
QUALCOMM Incorporated



Agenda

- **Events in BREW®**
- **Notifications in BREW**
- **Applet events**
- **Keypad events**
- **Dialog and Control events**
- **Alarm events**
- **Device events**
 - Flip and Keyguard events
 - PEN events
 - Joystick events
 - Screen Rotate events
- **User events**

Events in BREW

- BREW is event-driven
- Apps can send or post an event to the destination app using:
 - **ISHELL_SendEvent(pIShell, ClassID of Destination App, Event, wParam, lParam)** will send the event immediately and the control returns to the sender only after the event is processed by the receiving app
 - **ISHELL_PostEvent(pIShell, ClassID of Destination App, Event, wParam, lParam)** will queue the event to be sent. Event will be sent only after the sender finishes processing the current event or callback
- Several different types of events are supported in BREW

See “BREW Dispatcher” presentation for details on how BREW event dispatching works.

Notifications in BREW

- Apps can register for notification(s) issued by BREW or other classes by using:
 - **ISHELL_RegisterNotify**(plShell, ClassID of the app to notify, ClassID of the class that issues the event, Mask of events being registered)
- **EVT_NOTIFY** event is sent to the app that registered for notification(s)
 - **dwParam = AEENotify***
- Registration for notifications can be done in the app's MIF. i.e. Apps do not have to be executed to register for notifications
- In response to registered notifications, BREW loads the app passes it an event specific to the notification. e.g. **EVT_ALARM**

Applet Events

- **Commonly handled applet/application events:**
 - **EVT_APP_START:** When an application (app) is started or it calls **ISHELL_StartApplet**(plShell, App's classID) on itself
 - wParam = Bitmask of start codes
 - dwParam = AEEAppStart*
 - **EVT_APP_STOP:** When an app is stopped or it calls **ISHELL_CloseApplet**(plShell, bReturnToldle) on itself
 - dwParam = boolean*. App must set (*dwParam) to FALSE if it does not want to close and desires to become a background app
 - **EVT_APP_SUSPEND:** Sent to the current top-visible appl before a different app is made top-visible
 - wParam = AEESuspendReason
 - dwParam = AEESuspendInfo*
 - **EVT_APP_RESUME:** Sent to a suspended app to make it top-visible.
 - wParam = Bitmask of start codes
 - dwParam = **AEEAppStart***

See “BREW Application Framework” presentation for details on handling Applet events

Keypad Events

- **Commonly handled keypad events:**
 - **EVT_KEY_PRESS:** When a key is pressed
 - wParam = KEYCODE
 - dwParam = Bitflags for modifier keys (KB_LSHIFT, KB_RSHIFT, KB_LCTRL, KB_RCTRL, KB_LATL, KB_RALT etc).
 - **EVT_KEY:** While the key is pressed, this event is repeated
 - wParam = KEYCODE
 - dwParam = Bitflags for modifier keys
 - **EVT_KEY_RELEASE:** When a pressed key is released
 - wParam = KEYCODE
 - dwParam = Bitflags for modifier keys
 - **EVT_CHAR:** Character corresponding to the current key(s) pressed
 - wParam = AECHAR
 - dwParam = Bitflags for modifier keys

Keypad Events

- **Prior to BREW 3.1, only the top-visible apps can receive keypad events**
 - Starting with BREW 3.1., any app, even a background one, can register to receive key pad events. They must register for **NMASK_SHELL_KEY** notification
 - **NMASK_SHELL_KEY** Parameters
 - `dwParam = AEENotify*`. `pData` member of `AEENotify` points to `NotifyKeyEvent`
 - No change is required for top-visible apps

See “BREW Application Framework” presentation for details on Top-Visible and Background Apps

Control and Dialog Events

- Controls derived from IControl - IDateCtl, IMenuCtl, IStatic, ITextCtl, ITimeCtl - are examples of UI control interfaces in BREW
- Control related events that an app needs to be aware of begin with EVT_CTL_. For e.g.
 - EVT_CTL_TAB
 - EVT_CTL_ADD_ITEM
- Dialog(IDialog) related events that an app needs to be aware of begin with EVT_DIALOG_
- EVT_COMMAND is generated by UI controls often as a result of keypad events
- Please take a look at BREW API Reference for details on the above events

Alarm Events

- **There are a couple of ways to set alarms:**
 - **ISHELL_SetAlarm**(pIShell, ClassID of the App to be called when alarm expires, 16 bit user Code, Number of minutes to set the alarm for)
 - **IALARMMGR_SetAlarm**(pIAlarmMgr, ClassID of the App to be called when alarm expires, 16 bit user Code, Number of secs to set the alarm for)
- **When an alarm expires, EVT_ALARM event is passed to the app**
 - **EVT_ALARM**
 - wParam = nUserCode
- **If the app is not executing when the alarm expires, it is loaded by BREW and is passed EVT_ALARM**
- **Multiple expiring alarms are distinguished by user code**

Device Events – Flip and Keyguard Events




- **These events are delivered to the top-visible apps:**
 - **EVT_FLIP:** It is generated for clam-type/flip-type devices when the device's display accessibility is affected. This action is generally a consequence of opening and closing the device flip
 - wParam = TRUE: When the flip is open and display is accessible
FALSE: When the flip is closed and display is not accessible
 - **EVT_KEYGUARD:** This event is generated when any of the following happens:
 - A device's keypad is blocked when a clam-type device is closed
 - A device's keypad is re-exposed when clamshell is opened
 - Keys (Keypad) are locked by software and not by a piece of physical hardware
 - **EVT_KEYGUARD parameters**
 - wParam = TRUE: When the keyguard is on
FALSE: When the keyguard is off

Device Events – Flip and Keyguard Notifications

- **These notifications are delivered to any registered apps:**
 - **NMASK_DEVICENOTIFIER_FLIP:** This notification is generated for the same reason as EVT_FLIP
 - dwParam = AEENotify*. pData member of AEENotify points to AEEDeviceNotify
 - **NMASK_DEVICENOTIFIER_KEYGUARD:** This notification is generated for the same reason as EVT_KEYGUARD
 - dwParam = AEEDeviceNotify*. pData member of AEENotify points to AEEDeviceNotify


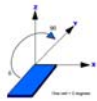

Device Events – Flip and Keyguard Behavior

Expected behavior on BREW 2.1, 3.x devices:

Item	Expected behavior on Slider 	Expected behavior on Clamshell 	Expected behavior on Rotator 
EVT_FLIP	Not sent on slider open or close	Sent to the top-visible app on flip open and close.	Not sent when rotate open or close
EVT_KEYGUARD	Sent to the top-visible app on slider open and close.	Sent to the top-visible app on flip open and close.	Sent to the top-visible app on rotate open and close
ISHELL_GetDeviceInfo() - AEEDeviceInfo.bFlip	bFlip = 0	bFlip = 1	bFlip = 0

Device Events – Flip and Keyguard Behavior

Expected only in BREW 3.x onwards devices:

Item	Expected behavior on Slider 	Expected behavior on Clamshell 	Expected behavior on Rotator 
Device Notifier	Notification sent to the registered app: NMASK_DEVICENOTIFIER_KEYGUARD	Notification sent to registered app: NMASK_DEVICENOTIFIER_FLIP and NMASK_DEVICENOTIFIER_KEYGUARD	Notification sent to the registered app: NMASK_DEVICENOTIFIER_KEYGUARD
ISHELL_GetDeviceInfoEx() - AEE_DEVICE_STATE_KEYGUARD_ON	Properly indicates when the keyguard is on (slider closed)	Properly indicates when the keyguard is on (flip closed)	Properly indicates when the keyguard is on (rotator closed)
ISHELL_GetDeviceInfoEx() - AEE_DEVICE_STATE_FLIP_OPEN	Returns EUNSUPPORTED	Properly indicates when the flip is open	Returns EUNSUPPORTED

Flip and Keyguard Events – How to Handle Them?

- If the carrier expects the app to *terminate* itself when the slider or clamshell is closed, then for devices exhibiting the expected behavior, do the following:

```
case EVT_FLIP: // wParam == FALSE when flip is closed
    if(!wParam){
        return FALSE; // return FALSE to close all apps
    }
    return TRUE; // flip was opened, don't terminate
case EVT_KEYGUARD: // wParam == TRUE when keyguard is
    enabled
    if(wParam){
        return FALSE; // return FALSE to close all apps
    }
    return TRUE; // flip/slider was opened, don't
terminate
```

Flip and Keyguard Events – How to Handle Them?

- If the carrier requirements state that all apps should terminate if the top-visible app returns **FALSE** to **EVT_FLIP** and **EVT_KEYGUARD**, it takes a little more code to handle the requirement
- Many existing 2.1 and 3.x devices have been commercialized with differing behavior
- In such cases, within the **EVT_FLIP** and **EVT_KEYGUARD** handler, the top-visible app has to call **IShell_CloseApplet(IShell *pIShell, boolean bReturnToldle)**.
- For 2.1, set **bReturnToldle** to **TRUE** so that all active apps are terminated
- For 3.x, you cannot set **bReturnToldle** to **TRUE** since it requires “System” privilege. Setting it to **FALSE** to terminate the current app is acceptable

Flip and Keyguard Events – How to Handle Them?

- Hence, a generic solution for handling EVT_FLIP and EVT_KEYGUARD on 2.1 and 3.x devices to *terminate* the app is:

```
#include "AEEStdlib.h"
// version number for BREW 3.0.0.0
#define BREW_V3 (0x030000001)
// macro to determine if BREW ver >= 3.0.0.0
#define IS_BREW_V3_OR_LATER() (GETAEEVERSION(NULL,0,0) >= BREW_V3)
case EVT_FLIP:
    if(!wParam){ // wParam == FALSE when flip closed
        if(IS_BREW_V3_OR_LATER()){ // BREW V3 - cannot return to idle
            // close applet for legacy flip support, bReturnToIdle = FALSE
            ISHELL_CloseApplet(pIShell, FALSE);
        }
        else{ // BREW V2 - return to idle
            // close applet for legacy flip support, bReturnToIdle = TRUE
            ISHELL_CloseApplet(pIShell, TRUE);
        }
        return FALSE; // return FALSE to close all apps
    }
    return TRUE; // else flip was opened, don't terminate
```

Flip and Keyguard Events – How to Handle Them?

```
case EVT_KEYGUARD:
    // wParam == TRUE when keyguard is enabled
    if(wParam){
        // BREW V3 - cannot return to idle
        if(IS_BREW_V3_OR_LATER()){
            // close app for legacy flip and slider support,
            // bReturnToIdle FALSE
            ISHELL_CloseApplet(pIShell, FALSE);
        }
        // BREW V2 - return to idle
        else{
            // close applet for legacy flip support, pass TRUE
            ISHELL_CloseApplet(pIShell, TRUE);
        }
        return FALSE; // return FALSE to close all apps
    }
return TRUE; // else flip/slider was opened, don't terminate
```

Flip and Keyguard Events – How to Handle Them?

- For apps intended to *continue execution* while the clamshell or slider is closed, the handling of EVT_FLIP and EVT_KEYGUARD will differ from previous case
- From what we have seen, these apps require special approval from the carrier since they differ from the typical carrier app requirement
- Important to note is that apps should explicitly return TRUE when handling these events

```
case EVT_FLIP:  
    // app-specific code to handle necessary operations  
    // such as drawing to the secondary display  
    ...  
    return TRUE;
```

Flip and Keyguard Events – How to Handle Them?

contd ...

```
case EVT_KEYGUARD:
    // app-specific code to handle necessary operations
    // when not all keys are accessible
    ...
    // app may choose to use IFLIP APIs to retrieve
    // the key codes of the keys accessible to the user based
    // on the current positions of all flips
    ...
    return TRUE;
```

NOTE: In BREW 3.1, IFLIP interface was introduced

- In addition to flip status (open/close), it provides additional information regarding rotation, angle of flip etc.
- It provides a means to retrieve the key codes of the keys accessible to user based on the current positions of all flips
- It provides a means to retrieve the display class IDs of the displays accessible to the user based on the current positions of all flips

Flip and Keyguard Events – How to Handle Them?

- **For apps that intend to recognize flip and keyguard state even when they are not top-visible, following needs to be done**
 - In EVT_START handler, register for NMASK_DEVICENOTIFIER_FLIP and NMASK_DEVICE_NOTIFIER_KEYGUARD
 - Handle the EVT_NOTIFY for the state transitions
 - Deregister for the above notifications on EVT_STOP
 - In this case, it's important to register and deregister for notifications programmatically than the MIF because the app may only want to be notified when it is executing. App isn't interested in being loaded just to handle the EVT_NOTIFY

Flip and Keyguard Events – How to Handle Them?

contd ...

```
#include "AEEDeviceNotifier.h"
...
// App is told it is starting up
case EVT_APP_START:
    // register for flip and keyguard notification (if necessary)
    // Call ISHELL_RegisterNotify() once with a single notification mask
    if(SUCCESS != ISHELL_RegisterNotify(pMe->a.m_pIShell, AEECLSID_MYAPPID,
    AEECLSID_DEVICENOTIFIER, NMASK_DEVICENOTIFIER_FLIP |
    NMASK_DEVICENOTIFIER_KEYGUARD)){
        // error handling code here
    }
    return TRUE;
...
// App is told it is exiting
case EVT_APP_STOP:
    // deregister for device notification by registering for mask of 0
    if(SUCCESS != ISHELL_RegisterNotify(pMe->a.m_pIShell, AEECLSID_HELLOWORLD,
    AEECLSID_DEVICENOTIFIER, 0)){
        // error handling code here
    }
    return TRUE;
```

Flip and Keyguard Events – How to Handle Them?

```
...
// App receives notification
case EVT_NOTIFY:
    // verify that notification is from device notifier
    if(AEECLSID_DEVICENOTIFIER == ((AEENotify *)dwParam)->cls){
        // accessing a temporary pointer to simplify typecasting
        AEEDeviceNotify *pDevNot = (AEEDeviceNotify *)((AEENotify *)dwParam)->pData;
        // check if this notification is for flip
        if(NMASK_DEVICENOTIFIER_FLIP & ((AEENotify *)dwParam)->dwMask){
            // AEEDeviceNotify.wParam == FALSE when flip closed
            if(!pDevNot->wParam){
                // app logic to handle flip close state
            }
            return TRUE;
        }
        // check if this notification is for keyguard
        if(NMASK_DEVICENOTIFIER_KEYGUARD & ((AEENotify *)dwParam)->dwMask){
            // AEEDeviceNotify.wParam == TRUE when keyguard is enabled
            if(pDevNot->wParam){
                // app logic to handle keyguard on state
            }
            return TRUE;
        }
    }
return TRUE;
```

Flip and Keyguard Events – Things to remember

- **A carrier might have additional default requirements for handling these events which the app developers should be aware of**
- **Identify how your app handles flip and keyguard events in the specifications document when submitting for TRUE BREW® Testing**
- **Depending on the kind of app, a carrier might have special requirements for approval**
 - For e.g. even if the default requirements mandate an app to terminate when flip is closed, a carrier might want some media playing apps and LBS apps to continue executing and take advantage of secondary display when flip is closed

Device Events – Pen events

- From BREW 3.1 onwards, touchpad input is supported via Pen events
- Following Pen events are delivered to top-visible apps:
 - **EVT_PEN_UP**: When the pen is lifted up from the pad
 - dwParam = position: signed x in upper 16 bits, y in lower
 - **EVT_PEN_DOWN**: When the pen touches the pad
 - dwParam = position: signed x in upper 16 bits, y in lower
 - **EVT_PEN_MOVE**: This event is repeated when the pen is moving on the surface of the pad
 - wParam = 15 most significant bits represent event generation time and thus can be used to calculate the time lag between events. The LSB in wParam is 1 if an EVT_PEN_MOVE event was dropped before the current EVT_PEN_MOVE event. If the previous EVT_PEN_MOVE event is in the event queue, then the LSB of wParam will be 0
 - dwParam = position: signed x in upper 16 bits, y in lower
 - **EVT_PEN_STALE_MOVE**: Stale move events are reported when event queue contains additional unprocessed moves
 - dwParam = position: signed x in upper 16 bits, y in lower

NOTE: For position values, 0, 0 is the upper left corner of the active frame

Device Events – Joystick Events

- **Support for joystick events has been added in BREW 3.1**
- **Following joystick events are delivered to top-visible apps:**
 - **EVT_JOYSTICK_POS:** This event is repeated when the joystick is being moved
 - wParam = 15 most significant bits represent event generation time and thus can be used to calculate the time lag between events. The LSB in wParam is 1 if an EVT_JOYSTICK_POS event was dropped before the current EVT_JOYSTICK_POS event. If the previous EVT_JOYSTICK_POS event is in the event queue, then the LSB of wParam will be 0
 - dwParam = position: signed x in upper 16 bits, y in lower
 - **EVT_JOYSTICK_STALE_POS:** Stale move events are reported when event queue contains additional unprocessed moves
 - dwParam = position: signed x in upper 16 bits, y in lower

NOTE: For position values, 0, 0 is the upper left corner of the active frame

Device Events – Headset Events

- **EVT_HEADSET** is passed to the top-visible app when the headset is plugged in or out
 - **wParam = TRUE**: if headset is plugged in
FALSE: if the headset is plugged out
- **Non top-visible apps can register for NMASK_DEVICENOTIFIER_HEADSET notification**
 - **dwParam = AEENotify***. **pData** member of **AEENotify** points to **AEEDeviceNotify**
- **Related to headset handling, at any time, apps can use ISound_Get() API to return the current audio device (AEESoundInfo. AEESoundDevice)**

Device Events – Screen Rotate Events

- Screen rotate event or notification is generated when the screen orientation is changed
- **EVT_SCR_ROTATE** event is delivered to the top-visible app
 - **wParam = AEEScrOrientation** (Portrait or Landscape)
- Non top-visible apps can register for **NMASK_DEVICENOTIFIER_SCR_ROTATE** notification
 - **dwParam = AEENotify***. **pData** member of **AEENotify** points to **AEEDeviceNotify**
- At any time, apps can use **ISHELL_GetDeviceInfoEx()** API with **AEE_DEVICESTATE_SCR_ORIENTATION** item to query the current screen orientation

User defined events

- User defined events in BREW begin with **EVT_USER**
- **EVT_USER** constant should be used by apps to help define private messages, usually in the form of **EVT_USER + y**, where **y** is an positive integer
 - `#define EVT_MYAPP_MSG` `EVT_USER + 100`

Thank You