



ZOOM OUT BREW 2008 CONFERENCE

**QWERTY Keypad Support – BREW®
Standardized Approach**

Alhad Purnapatre, Senior Engineer

Malay Prajapati, Engineer

Qualcomm Incorporated

QUALCOMM



Outline

- Need for standard approach
- Details of the approach – key events, modifiers, IKeysMapping
- Porting steps for OEMs
- Application development considerations
- Examples



Need for standard approach

- No standard approach in BREW before this
- Keypad formats are not standard
- This means [Function] + [A] can mean ‘#’ on one handset and ‘*’ on another, as an example
- Apps need to be re-written for individual handsets because of these variations



The standard approach – Key events

- Standard EVT_KEY_PRESS, EVT_KEY, EVT_KEY_RELEASE events
- For alphabet keys i.e. keys from A to Z, wParam = unicode value of lower case letter
- Ex.: Pressing 'A' will generate EVT_KEY events with wParam = 97
- New AVK codes AVK_A through AVK_Z, equal to corresponding unicode values



Modifiers – Shift and Caps Lock

- Will not change the AVK value sent in wParam
- dwParam will contain the correct modifier flag i.e. KB_LSHIFT, KB_RSHIFT, KB_CAPSLOCK
- Ex.: Pressing [SHIFT] + [A] will generate key events with wParam = AVK_A and dwParam = KB_LSHIFT



Modifiers – Function, symbol

- Keys such as Function and Symbol that overload other keys
- Ex.: [Symbol] + [A] can mean ‘#’
- App will still get EVT_KEY with the original keycode i.e. AVK_A in the above example
- dwParam will contain the modifier for ex. KB_SYMBOL
- Apps will need a mapping mechanism to get interpretation of keys



Mapping mechanism - IKeyMapping

- New Interface defined: IKeyMapping
- IKeyMapping_GetMapping API
- Maps an AVK code and modifier combination to an interpreted Unicode value



IKeysConfig

- Allows applications to set and get the state of ‘sticky’ keys – Caps Lock, Num Lock, Scroll lock etc
- The setting affects the global state of the key
- Useful for turning sticky modifiers on and off as desired by an app



OEM Porting – Key Events

- Use new defines AVK_A through AVK_Z to send alphabet key events
- Include any modifiers in the key events as dwParam
- Use AEE_Event rather than AEE_Key since modifiers need to be sent
- Use AVK_SPACE and AVK_ENTER for a space bar and a return key.
- For other QWERTY keys without alphabet characters, send unicode value of the primary character as wParam

OEM Porting – Key Events

Ex.: User presses [Left Shift] + [A]

```
/* User pressed left SHIFT */
AEE_Event(EVT_KEY_PRESS, AVK_LSHIFT, 0);
AEE_Event(EVT_KEY, AVK_LSHIFT, 0);

/* User pressed A (and is still holding SHIFT) */
AEE_Event(EVT_KEY_PRESS, AVK_A, KB_LSHIFT);
AEE_Event(EVT_KEY, AVK_A, KB_LSHIFT);

/* User released A */
AEE_Event(EVT_KEY_RELEASE, AVK_A, KB_LSHIFT);

/* User released left SHIFT */
AEE_Event(EVT_KEY_RELEASE, AVK_LSHIFT, 0);
```



Porting IKeysMapping

- Reference implementation available in BREW Porting Kit (3.1.5 SP01 and further)
- OEMKeysMapping.c and FEATURE_KEYS_MAPPING
- IKeysMapping requires a mapping text file to be generated and available on the handset



KeysMapping.ini

- List out the key mappings available on the handset
- Example format:

```
AVK_A + KB_LSHIFT = 'A'
```

```
AVK_A + KB_RSHIFT = 'A' // Provide both shift keys
```

```
AVK_A + KB_SYMBOL = '#'
```

```
AVK_A + KB_LALT | KB_LSHIFT = 0xA9
```

- File is free-form and supports C++ style single line comments



KeysMappingParser

- Utility to convert KeysMapping.ini into a 'handset-readable' format – map.csv
- Available in pk\utils
- Uses AEEVCodes.h as pointed to by BREWDIR to parse KeysMapping.ini
- Also possible to specify a different AEEVCodes.h through command line



map.csv

- Generated by KeysMappingParser
- Load this on handset and change the MAPPINGS_FILE define in OEMKeysMapping.c to point to the ported location
- Submit with device pack – useful in simulation and PEK



Device pack and PEK

- New DPK items
 - IDS_DD_EXTENDED_KEYPAD_SUPPORT (Yes/No)
 - IDS_DD_EXTENDED_KEYPAD_MAP_FILE_LOCATION
 - Path to this file on device
- PEK
 - Run OATKeysMapping PEK module
 - Run OATKey ExtendedKeys.n test cases



ITextCtl support

- Available as a new text mode (AEE_TM_EXTENDED_KEYS)
- Reference implementation available in OEMText.c
- Auto-interpretation of key combinations – calls IKeysMapping_GetMapping internally



Application Development

- Expect standard EVT_KEY_PRESS, EVT_KEY and EVT_KEY_RELEASE events
- Call IKEYSMAPPING_GetMapping to get key combination interpretation
- Call IKEYSCONFIG_GetStickyKeys to get sticky keys available on device
- Call IKEYSCONFIG_GetKeyState to get sticky key state
- Call IKEYSCONFIG_SetKeyState to set sticky key state
- Use ITextCtl AEE_TM_EXTENDED_KEYS mode for text entry

IKeyMapping Example

```
boolean App_HandleEvent(...)
{
    switch( eCode )
    {
        case EVT_KEY:
            if ( dwParam ) // dwParam contains modifier keys
            {
                AECHAR mapping;

                /* Get mapping */
                if ( IKEYSMAPPING_GetMapping(pKeyMapping,
                    wParam, dwParam, &mapping) != SUCCESS )
                    return FALSE; // Or use application specific logic
                                    // for multiple key presses
                else
                    /* Use mapping */
                    .....
            }
            /* Code to handle standard AVK keys such as CLR, SELECT, UP,
            DOWN, LEFT, RIGHT, primary keys (A-Z, 0-9) */
            .....
        }
    }
}
```

IKeysConfig Example – Obtaining interface

```
/* First create an instance of IKeysMapping */
if((SUCCESS == ISHELL_CreateInstance(pIShell,
    AEECLSID_KEYSMAPPING, &pIKeysMapping)){

/* Then obtain IKeysConfig as a QI on IKeysMapping */
if(IKEYSMAPPING_QueryInterface(pIKeysMapping,
    AEEIID_KEYS_CONFIG, &pIKeysConfig) == SUCCESS) {

    /* Use Interface */
    // ...

```

IKeysConfig Example – Use API

```
//Exercise IKeysConfig interface
if(SUCCESS == IKEYSCONFIG_GetStickyKeys(pIKeysConfig,
                                         &dwStickyKeys)){

    boolean keystate;

    if (dwStickyKeys & KB_CAPSLOCK){
        //If caps lock is present, read its state
        IKEYSCONFIG_GetKeyState(pIKeysConfig, KB_CAPSLOCK,
                                &keystate);
        /* Logic to handle Caps Lock state */
        /* IKEYSCONFIG_SetKeyState() may be used to change
           key states if desired */
    }
}
```



TextCtl example

```
//Create ITextCtl
ISHELL_CreateInstance(pIShell, AEECLSID_TEXTCTL,
                    &pTextCtl);

//Other initialization - set the rectangle etc.
// ...
//Set input mode
ITextCtl_SetInputMode(pTextCtl, AEE_TM_EXTENDED_KEYS);

// In HandleEvent, if ITextCtl is active
return ITEXTCTL_HandleEvent(pTextCtl, eCode,
                            wParam, dwParam);
```



API Support

- Prior to BREW 3.1.5 SP01
 - Available as dynamic extension
 - Use same header file as BREW SDK 3.1.5 SP01
 - ITextCtl has partial or no support
 - No simulator support
- From BREW 3.1.5 SP01
 - IKeysMapping and IKeysConfig features are optional
 - If create instance fails then API is not supported
 - Check device pack for support information
(IDS_DD_EXTENDED_KEYPAD_SUPPORT)



TRUE BREW[®] Testing (TBT)

- New test cases will be added to TBT test plan
- Modifier keys have desired effect on application
- Data entry screen reflects changes in modifier keys to entered text
- Application behavior reflects state of sticky key



Thank you

- Documentation resources
 - PK Porting guide in 3.1.5 SP01
 - PEK user guide, test case descriptions
 - BREW API Reference guide
 - Extended keyboard KB article on Developer Extranet