Displaying Springboard Applications in Custom Launchers

Table of Contents

1. Introduction

This application note describes the modifications required to properly display Springboard module-resident applications in a Launcher application. Other third party Launcher applications or OS overlays can leverage this technique to properly display Springboard module applications.

Handspring has modified the Launcher application to be able to display applications on the Springboard module (internally implemented as card 1). Handspring has also modified the Launcher application to uniquely distinguish module-resident applications by adding a bullet dot to the name of the application. (See Figure 1.0 below).

Figure 1.0: Screenshot of Visor displaying the Springboard PhoneUI and SMSUI application

2. Displaying Springboard Applications with a Bullet Mark

Displaying the bullet mark is very straightforward. The developer modifies the application software source code at which the application draws the application's icon bitmap and program name.

To display a bullet mark, first create a string resource definition containing the bullet character. The bullet character can be any ASCII character that the developer deems appropriate. Handspring recommends the ASCII character value `0x95` as the optimal bullet character (except for the Japanese-localized Visor), filling 3 x 3 pixels. For the Japanese-localized Visor, Handspring recommends the ASCII character value `0xA5` as the optimal bullet character, filling 2 x 2 pixels. This resource can be cached to a global variable for faster access.

Next, while traversing the Launcher's database records to display the application names, verify that the card number field is card 1 (Springboard module). If it is, draw the bullet character, then draw the icon name.

2.1. Sample Code

The below sample code demonstrates how to add the bullet mark to Springboard applications. This sample code is extracted from Handspring's Launcher application where the software draws the application icon bitmaps and program names in "icon view" or "list view". Developers will identify the appropriate drawing code in their software and modify the code accordingly.

```
// Bullet character string resource definition for .rcp file
STRING ID rscBulletStrID "•"

// Global variables for caching bullet character
#define  bulletStrSize  4
Char     BulletStr[bulletStrSize];
UInt16   BulletStrLen;


StartApplication (UInt16 cmd)
{

.............
.............

  // Cache the bullet string (character)
  GetStringResource (rscBulletStrID, BulletStr, bulletStrSize);
  BulletStrLen = StrLen (BulletStr);
}
```

```
/***********************************************************************
 *
 * FUNCTION:    AppsViewDrawListItem
 *
 * DESCRIPTION: This routine the draw the applications's small icon and
 *              name in the list view.
 *
 * PARAMETERS:  itemNum          - record number that corresponds to the
 *                                 table item to draw
 *              bounds           - bound to the draw region
 *              transBitMapOnly  - if true, only draw bitmap part of item
 *                                 hasTransparency
 *
 * RETURNED:    errNone if no error
 *
 *
 ***********************************************************************/
static Err
AppsViewDrawListItem (UInt16 itemNum, RectanglePtr bounds, Boolean
transBitMapOnly)
{

.............
.............

  // Get ptr to the app entry
  recP = GetLauncherItem (itemNum);

  // Draw name if desired
  if (!transBitMapOnly)
       {
         // Draw app icon name
         currFont = FntSetFont (launcherFont);


         // Reset X, Y coordinates
         x = bounds->topLeft.x + 1 + listViewLabelMargin;
         y = bounds->topLeft.y + ((bounds->extent.y - FntLineHeight ()) / 2);

         // Draw the bullet indicator if this app is on the removable card
         if (recP->cardNo != 0)
              {
                // Use new bullet string
                WinDrawChars (BulletStr, BulletStrLen, x, y);
              }
         // Advance to next column
         x += listViewCardIndWidth;

         // Calc max string width that fits in item bounds
         appNameLenWidth = bounds->extent.x - iconP->width - 1;

         // Draw app name
         DrawCharsWithEllipsis ((char *) recP->iconName, x, y, appNameLenWidth);
         FntSetFont (currFont);
       }

  ReleaseLauncherItem (recP, itemNum, false);

.............
.............

}
```

```
/***********************************************************************
 *
 * FUNCTION:    AppsViewDrawIconItem
 *
 * DESCRIPTION: This routine draws the applications's small icon and
 *              name in the icon view.
 *
 * PARAMETERS:  itemNum          - record number that corresponds to the
 *                                 table item to draw
 *              bounds           - bound to the draw region
 *              transBitMapOnly  - if true, only draw bitmap part of item
 *                                  hasTransparency
 *
 * RETURNED:    errNone if no error.
 *
***********************************************************************/
AppsViewDrawIconItem (UInt16 itemNum, RectanglePtr bounds, Boolean
transBitMapOnly)
{
.............
.............

  // Get ptr to the app entry
  recP = GetLauncherItem (itemNum);


  // Draw app icon name if desired
  if (!transBitMapOnly)
       {

         // ------------------------------------------------------------------
         // Draw app icon name
         // Put a bullet in front of the name if it's on
         // the removable card.
         // ------------------------------------------------------------------
         if (recP->cardNo != 0)
              {
                // Use new bullet string
                StrCopy (name, BulletStr);
              }
         else
              name[0] = 0;
         StrCat (name, recP->iconName);

         currFont = FntSetFont (launcherFont);

         // Calc max string len and width that fits in item bounds
         appNameLenWidth = bounds->extent.x - 1;
         appNameLen = StrLen (name);
         FntCharsInWidth (name, &appNameLenWidth, &appNameLen, &fits);

         x = bounds->topLeft.x + (Int16) (((Int16) bounds->extent.x - (Int16)
                                                        appNameLenWidth) / 2);
         y = bounds->topLeft.y + (bounds->extent.y - FntLineHeight ());
         DrawCharsWithEllipsis (name, x, y, appNameLenWidth);
         FntSetFont (currFont);
       }
  ReleaseLauncherItem (recP, itemNum, false);

.............
.............

}
```

```
/***********************************************************************
 *
 * FUNCTION:        GetLauncherItem
 *
 * DESCRIPTION:     Provides access to the Launch database.  This function returns
 *                  a locked ptr to the record of the given index.
 *                  Call ReleaseLauncherItem when you're done with it, and
 *                  it will be unlocked & released there.
 *
 * PARAMETERS:
 *                  itemIndex    -> index of item requested.
 *                  update -> True to update the item if its dirty
 *
 * RETURNED:        Ptr to requested item
 *
 * REVISION HISTORY:
 *                  Name   Date         Description
 *                  ----   ----         -----------
 *
 *
 ***********************************************************************/
static DmLaunchDBRecordPtr
GetLauncherItem (UInt16 itemIndex)
{
  MemHandle       recH;
  DmLaunchDBRecordPtr recPtr;


  recH = DmGetRecord (LaunchDBRef, itemIndex);
  ErrNonFatalDisplayIf (recH == 0, "can't get record");

  recPtr = (DmLaunchDBRecordPtr) MemHandleLock (recH);

  return recPtr;
}
```

```
/***********************************************************************
 *
 * FUNCTION:        ReleaseLauncherItem
 *
 * DESCRIPTION:     Unlock and clear the busy bit on a record from the Launch
 *                  database.
 *                  Also sets the dirty bit if 'dirty' is true.
 *
 * PARAMETERS:
 *                  recP         -> Ptr to Launch database record, obtained
 *                                  via GetLauncherItem()
 *                  itemIndex    -> index of item requested.
 *                  dirty        -> Pass 'true' if the record was changed.
 *
 * RETURNED:        none
 *
 * REVISION HISTORY:
 *                  Name   Date         Description
 *                  ----   ----         -----------
 *
 *
***********************************************************************/
static void
ReleaseLauncherItem (DmLaunchDBRecordPtr recP, UInt16 itemIndex, Boolean dirty)
{

  MemPtrUnlock ((MemPtr) recP);
  DmReleaseRecord (LaunchDBRef, itemIndex, dirty);

  CloseLaunchDatabase ();

  return;
}
```

```
/***********************************************************************
*
* FUNCTION:         DrawIndicatorAndCharsWithEllipsis
*
* DESCRIPTION:      Draw a string in given bounds using ellipsis clipping
*                     if necessary. If 'onCard' is true, then a bullet
*                     character will be drawn in the column to the left
*                     of the name.
*
* PARAMETERS:
*             nameP         IN     name to draw
*             indicator     IN     if true, draw bullet in left column. If false,
*                                    left left column blank
*             x             IN     x coordinate to start drawing
*             y             IN     y coordinate to start drawing
*             maxWidth      IN     max width
*
* RETURNED:         nothing
*
*
***********************************************************************/
static void
DrawIndicatorAndCharsWithEllipsis (const char* nameP, Boolean indicator,
                                           Int16 x, Int16 y, UInt16 maxWidth)

  UInt16          colWidth;


  // Draw the indicator, if on
  colWidth = FntCharsWidth (BulletStr, BulletStrLen) + 2;
  if (indicator)
       WinDrawChars (BulletStr, BulletStrLen, x + 1, y);
  x += colWidth;
  maxWidth -= colWidth;

  // Draw the name
  WinDrawTruncChars (nameP, StrLen (nameP), x, y, maxWidth);
}
```

3. Springboard Applications Category Management

Category management for Springboard module-resident applications is more complicated. Handspring has encapsulated the APIs for category management into a library called `LdbMgr`.

The common `LdbMgr` APIs use for category management are:

- `LdbOpen()` to open the `LdbMgr` library and cache the category information for faster access
- `LdbInterfaceGet()` to get the address pointers for the `LdbMgr` APIs
- `LdbClose()` to close the `LdbMgr` library and write out the category information to the Launcher's database
- `LdbCategoryLabelGet()` to get the category label based on the id
- `LdbCategoryLabelSet()` to set the category label based on the id.

The APIs prototypes are:

```
Err LdbOpen (UInt16 refNum, UInt32 openFlags, UInt8* alreadyOpenP);

Err LdbInterfaceGet (UInt16 refNum, UInt32 interfaceId,
                                    void* interfaceP, UInt32 interfaceSize);

Err LdbClose (UInt16 refNum, UInt32 closeFlags, UInt8* stillOpenP);

Err LdbCategoryLabelGet (LdbInterfaceRef interfaceRef, UInt16 catId,
                         Char* labelBufP, UInt16 bufSize);

Err LdbCategoryLabelSet (LdbInterfaceRef interfaceRef, UInt16 catId,
                             Char* labelP);
```

Review the `LdbMgr.h` file for the full list of available APIs.

4. History

Date	Revision #	Description of changes
15 Jan 01	1.00	Initial release