ePad-vision Firmware 4.3 R11403
Release Notes

Firmware Supported SIF Fonts

A. Adding following reports to support SIF Fonts functions:

F_FONT_INFO_FLASH: to get font info from Flash F_FONT_INFO_RAM: to get font info from RAM F_FONT_BODYRAM: to transfer SIF font data from Flash to RAM F_RAM_INFO: to get available storage info in RAM F_PURGE_ALL_FONTS: to delete all the SIF fonts in RAM F_SIF_OPEN_FILE: to create a SIF file in the Flash
F_SIF_DATA: to transfer SIF font file data from host to device

B. Simplified the downloadFonts structure in the ePadVisionApp.h to reduce number of pointers created from earlier version.

```
struct {
        U8 * pNextFont;                 // The address for the next download font
        U8 * pCurrentFontData;          // Points to the currnet transfering font
        data; U32 fontSize[MAX_DOWNLOADFONTS];          // The font size
        U8 fontID;                                      // The curent font ID
        U8 fontNameLen[MAX_DOWNLOADFONTS];// font name length
        GUI_FONT fontGUIData[MAX_DOWNLOADFONTS]; //place holder for
        created font char
        fontName[MAX_DOWNLOADFONTS][MAX_SIFNAMELEN];
        // The font name corresponding to each font entry
} downloadFonts;
```

C. Modified the external RAM size to 8M. This reflects the current hardware implementation in storageManager.h

D. Modified the WorkingArea structure to hold HID report data sending from host in HIDComm.h

   a. Remove FONT_LOAD_REPORT
   b. Add RAMSPACE_REPORT, OPEN_SIF_REPORT

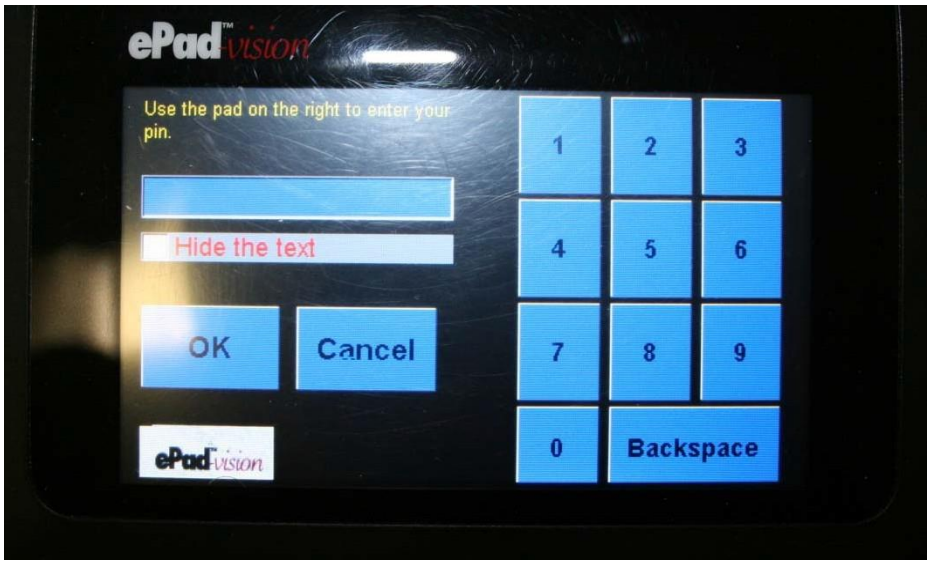
The following table lists fonts supported in the ePad-vision firmware before R11309:

SAN_R_13	MON_B_08	COM_B_18
SAN_B_13	MON_B_09	COM_B_24
SAN_R_16	MON_B_10	
SAN_B_16	MON_B_12	
SAN_R_20	MON_B_13	
SAN_B_20	MON_B_15	
SAN_R_24	MON_B_16	
SAN_B_24	MON_B_17	
SAN_R_32	MON_B_18	
SAN_B_32	MON_B_48	

In ePad-vision Firmware 4.0 R11309, all the MON and COM fonts will be mapped to SAN fonts which have extended ASCII characters included, as the following table shows:

Non-supported fonts	Supported fonts
MON_B_08 MON_B_09 MON_B_10 MON_B_12 MON_B_13	SAN_R_13
	SAN_B_13
MON_B_15 MON_B_16 MON_B_17 MON_B_18	SAN_R_16
	SAN_B_16
	SAN_R_20
COM_B_18	SAN_B_20
	SAN_R_24
COM_B_24	SAN_B_24
MON_B_48 SAN_B_32	SAN_R_32

PinPad Feature



A new API has been added to ePad-vision software to support the PinPad feature:

- void DisplayPinPad(string XMLData, int timeout, out string output)

XMLData:
The XML string defines the Pin Pad layout. The XMLData for the Pin Pad layout is similar to the XML data used in method LoadWidgetLayout(string XMLDocument). For the details, please refer the XML Schema section.

The default PinPad XML file, WPinPad.xml, is located at the ePadLink\ePad\Images\ePad-vision\XML folder.

timeout:
Controls the Pin Pad session duration. The Pin Pad will be forcibly terminated and an ePadexception will be raised if timeout period expires. The value is expressed millisecond.

output:
Returns the user's input entered on the ePad-vision device. The output has the value "null" if the user terminates the Pin Pad session by clicking the Cancel button. If the user doesn't enter any digits or deletes all inputted digits and then clicks the OK button, it has the empty string (zero-length string).

The XML Schema

The XML schema is similar to that used in LoadWidgetLayout(…). It has the root element "PinPad" instead of "WidgetLayout". The following is the definition in XSD format. It may also be found in the file PinPad.XSD bundled in the ePad-Vision SW codebase.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name ="PinPad">
      <xs:complexType>
                                          <xs:sequence>
            <xs:element name="Widget" maxOccurs ="unbounded">
                                             <xs:complexType>
               <xs:sequence>
               </xs:sequence>
               <xs:attribute name="ID" type="xs:string" />
               <xs:attribute name="FontName" type="xs:string" />
               <xs:attribute name="BGColor" type="xs:string"/>
               <xs:attribute name="FGColor" type="xs:string"/>
               <xs:attribute name="Width" type="xs:unsignedShort" />
               <xs:attribute name="Height" type="xs:unsignedShort" />
               <xs:attribute name="X" type="xs:unsignedShort"/>
               <xs:attribute name="Y" type="xs:unsignedShort"/>
               <xs:attribute name="Effect" type="xs:string" />
               <xs:attribute name="Text" type="xs:string" />
                                             </xs:complexType>
            </xs:element>
                                          </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

To make the XML data simpler and compatible with the predefined XML data, the following changes have been implemented.

Widget ID Attribute

To make the XML data more readable, the widget ID attribute has the type "string" instead of "integer". All valid Pin Pad widgets (controls inside the Pin Pad) have their own name defined in an enumeration type PinPadControls. Instead of directly giving the widget ID number, the relevant enumeration item name should be given. The following code snippet gives the definition.

```
/// <summary>
/// Defines the controls of Pin Pad Wwdget.
/// </summary>
public enum PinPadControls
{
    /// <summary>
    /// The numberic key for number 0.
    ///
    </summary
    > NUM_0 =
    10,
    /// <summary>
    /// The numberic key for number 1.
    ///
    </summary
    > NUM_1 =
    1,
    /// <summary>
    /// The numberic key for number 2.
    ///
    </summary
    > NUM_2 =
    2,
    /// <summary>
    /// The numberic key for number 3.
    ///
    </summary
    > NUM_3 =
    3,
    /// <summary>
    /// The numberic key for number 4.
    ///
    </summary
    > NUM_4 =
    4,
    /// <summary>
    /// The numberic key for number 5.
    ///
    </summary
    > NUM_5 =
    5,
    /// <summary>
    /// The numberic key for number 6.
    ///
    </summary
    > NUM_6 =
    6,
    /// <summary>
    /// The numberic key for number 7.
    ///
    </summary
    > NUM_7 =
    7,
    /// <summary>
    /// The numberic key for number 8.
    ///
```

```
        </summary
        > NUM_8 =
        8,
        /// <summary>
        /// The numberic key for number 9.
        ///
        </summary
        > NUM_9 =
        9,
        /// <summary>
        /// The functional key to backspce.
        /// </summary>
        FUN_BACKSPACE
        = 11,
        /// <summary>
        /// The fuctional checkbox to show the entered numbers as
        asterisks (optional item).
        ///
        </summary
        > FUN_SEC
        = 30,
        /// <summary>
        /// The functional botton to accept and dismiss the Pin Pad.
        ///
        </summary
        > FUN_OK
        = 40,
        /// <summary>
        /// The functional botton to cancel and dismiss the Pin Pad.
        ///
        </summary>
        FUN_CANCEL
        = 50,
        /// <summary>
        /// The text field to show the customized caption (optional
        item).
        ///
        </summary
        > CAPTION
        = 19,
        /// <summary>
        /// The text filed to display the entered numberic string.
        ///
        </summary
        > CONTENT
        = 20,
        /// <summary>
        /// The customized image (optional item).
        ///
        </summary
        > IMAGE =
        60,
}
```

Defaults and Omitted Attributes

The widget type attribute has been omitted because we use the enumerate item name as the widget ID and the relevant widget type is implied by the widget enumerate item name.

For each widget, its default text will be used if the relevant text attribute is not given in the XML data. The following list shows the defaults.

Numeric buttons:	*The corresponding numeric character*
Backspace button:	BACKSPACE
The OK button:	OK
The Cancel button:	CANCEL
The password option:	Hide the input

For the widget that displays the entered numeric string, the text attribute will be ignored.

The Image

The ID enumeration item “IMAGE” is designed to show the images.

The image data are given via the Text attribute. It can be any image format supported by the host Windows platform, for example, BMP, JPEG, JIF, …. The image data must be in Base64 format and assigned the “Text” attribute. This could be done through following C# sample code.

```
FileStream fs = new FileStream(ImageFileName, FileMode.Open, FileAccess.Read);
Image image = Image.FromFile(ImageFileName);

Byte[] _imageData = (byte[])(new BinaryReader(fs)).ReadBytes((int)fs.Length);

FileInfo f = new FileInfo(targetFileName);
StreamWriter w = f.CreateText();
w.WriteLine(System.Convert.ToBase64String(_imageData, 0, _imageData.Length));
w.Close();
```

The image in the Base64 format will be stored in the targetFileName.

Please refer to the Test Program Image2Base64 Button to see how to generate Base64 format image data.