# HandEra 330

# Programmer's Reference Manual

Revision 1.05
Revision Date: Sept 2001

**CONTACT INFORMATION**

| | |
|---|---|
| **HandEra, Inc.** | 2851 104th St., Suite H<br>Des Moines, IA 50322<br>U.S.A.<br>515-252-7525<br>www.handera.com |
| Palm Computing World Wide Web | www.palm.com |
| Metrowerks World Wide Web | www.metrowerks.com |

# Table of Contents

# Chapter 1: About This Document

The *HandEra 330 Programmer's Reference Manual* is part of the HandEra Software Development Kit (SDK). This document is intended as a guide for developers interested in creating applications for the HandEra 330 handheld computer.

This development kit does not contain information regarding the design and implementation of standard Palm OS applications. For this information, please refer to the Palm OS SDK documentation provided by Palm at http://www.palmos.com. Relevant documents include:

- *Palm OS SDK Reference*
- *Palm OS Programmer's Companion*
- *Constructor for Palm OS*

This document assumes the reader is familiar with basic Palm OS concepts detailed in *the Palm OS Programmer's Companion*. This document also uses the basic typographical conventions found in Palm documentation.

- Code elements, such as functions and structures, will use a fixed width font.
- **Bold type will be used for emphasis.**
- Document names, such as *Palm OS Programmer's Companion*, are italicized.

# Chapter 2: VGA API

This chapter provides a reference to the VGA Extension API procedures.  This chapter is directed toward Palm OS application developers who wish to write applications for the higher-resolution HandEra VGA screens.  It is assumed that the reader is familiar with the C programming language, in particular within the context of the Palm OS.

♦ Section 1 of this chapter provides background details on the screen.

♦ Section 2 of this chapter details the type declarations used by functions in the VGA API.

♦ Section 3 describes each of the VGA API calls, describing their function, parameters and return value.

♦ Section 4 lists the possible error codes and their interpretation.

## Section 1 – VGA screen overview

The VGA Extension provides an interface to the larger screen resolution on the HandEra 330.  The interface provides calls to determine the user interface area of the screen and rotation of the drawing.  VGA Extensions also provides backward compatibility modes to existing Palm OS applications written for a lower-resolution screen.

The use of this Software Development Kit (SDK) allows the developer to utilize the extra screen area on the device.

### VGA Extension 1.0 Feature Set

Before making a VGA Extension API call, an application needs to ensure that the VGA Extension itself is present and is compatible with the API call.  Attempting to make VGA Extension calls on a non-HandEra device will crash the application – this is an inherent limitation of any Palm OS extension.  The application should make a `FtrGet` function call to determine if the extension is present and what its version number is.

```
UInt32 version;
if FtrGet(TRGSysFtrID, TRGVgaFtrNum, &version) == 0)
{
   if (sysGetROMVerMajor(version) >= 1)
   {
      the VGA Extension 1.0 is present
   }
}
```

The VGA Extension calls may be grouped into four categories: screen management, font management, form management, and legacy application management.  The calls, grouped by category, are listed in the following tables, with brief descriptions of each function.

## Table 2-1.  Screen Management

| VgaGetScreenMode | Get the current screen mode. |
|---|---|
| VgaSetScreenMode | Set new screen mode. |
| VgaGetScreenState | Get the current screen state |
| VgaRestoreScreenState | Restore screen state |
| VgaRotateSelect | Screen rotation common dialog. |

## Table 2-2.  Font Management

| VgaBaseToVgaFont | Convert standard base font ID to the larger VGA font ID. |
|---|---|
| VgaVgaToBaseFont | Convert larger VGA font ID to standard Palm font ID |
| VgaFontSelect | Display UI form to select a font. |
| VgaIsVgaFont | Returns true if the font is one of the VGA fonts. |
| VgaTableUseBaseFont | Allows table items to be drawn with either the base font or larger VGA font. |

## Table 2-3.  Form Management

| VgaGetFrmTitleHeight | Get the form title height. |
|---|---|
| VgaFormModify | Modify form to a new form size. |

## Table 2-4.  Legacy Application Management

| VgaBitmapExpandedExtent | Returns the extent of a bitmap to be scaled |
|---|---|
| VgaWinDrawBitmapExpanded | Draws and scales a bitmap image |

# Section 2 – VGA API Type Reference

This section details the enumerated data types and structures used by the API functions.

## *VgaScreenModeType*

| | |
|---|---|
| screenModeScaleToFit | Application expands 1.5 times. |
| screenMode1To1 | Application displays as is. |

## *VgaRotateModeType*

| | |
|---|---|
| rotateModeNone | No rotation. |
| rotateMode90 | Screen rotates 90 degrees. |
| rotateMode180 | Screen rotates 180 degrees. |
| rotateMode270 | Screen rotates 270 degrees. |

## VgaOffsetModeType

| | |
|---|---|
| offsetModeTopLeft | Display at top left. |
| offsetModeTopCenter | Display at top center. |
| offsetModeTopRight | Display at top right. |
| offsetModeCenterLeft | Display at center left. |
| offsetModeCenterCenter | Display at center center. |
| offsetModeCenterRight | Display at center right. |
| offsetModeBottomLeft | Display at bottom left. |
| offsetModeBottomCenter | Display at bottom center. |
| offsetModeBottomRight | Display at bottom right. |

## VgaFontSelectType

| | |
|---|---|
| vgaFontSelectBase | Palm OS FontSelect form. |
| vgaFontSelectVgaText | FontSelect form with text fonts 4 Palm and 4 VGA. |

## VgaFormModifyType

| | |
|---|---|
| vgaFormModify160To240 | Converts a 160x160 form to a 240x240 form. |

# Section 3 – VGA API Function Reference

### VgaFormModify

**Purpose**          Scale form based on parameter type.

**Prototype**        `Err VgaFormModify(FormType *frmP, VgaFormModifyType type);`

**Parameters**       `-> frmP`             Pointer to form.

                  `-> type`             Modify types:

                                     VgaFormModify160To240 – Scale a 160x160 form to a 240x240 form.

**Result**           `Err`

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**         This routine resizes a form from one size to another and moves all items on the form to a specified location and font size.

                  **VgaForm160To240**

                  Resizes a 160x160 to 240x240 and adjust the objects location, size and font.

                  Calling VgaFormModify() should only be called in 1To1 Mode.

# VgaFrmGetTitleHeight

**Purpose**    Get the form title height.

**Prototype**    `UInt16 VgaFrmGetTitleHeight(void);`

**Parameters**    None

**Returns**    Height of forms title.

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**    Returns the height of the form titlebar in pixels.

## VgaGetScreenMode

**Purpose**    Get the current screen mode.

**Prototype**    `void VgaGetScreenMode(VgaScreenModeType *mode,`
                                `VgaRotateModeType *rotation);`

**Parameters**    `<- mode`        One of the screen modes.

                 `<-`             One of the rotation modes.
                 `rotation`

**Result**    None

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**    This function will suffice for the majority of applications.  However, if the application is sublaunched by a 3<sup>rd</sup> party application, then the function VgaGetScreenState() should be used.

## VgaGetScreenState

**Purpose**        Get the current screen state, such as its state and orientation.

**Prototype**      `void VgaGetScreenState(VgaScreenStateType *state);`

**Parameters**    `<- state`      Current screen state

**Result**        None

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**      This function should be used instead of VgaGetScreenMode() for applications that may be sublaunched by 3rd party applications, such as phone-lookup functions.  This should be used in conjunction with VgaSetScreenState().

## VgaIsVgaFont

**Purpose**      Determine whether a font ID is a VGA font.

**Prototype**    `Boolean VgaIsVgaFont(FontID font);`

**Parameters**   `-> font`     Font ID.

**Result**       True if font is a VGA font, false otherwise.

**Compatibility** Implemented only if VGA Extension 1.0 is present.

**Comments**     Use to determine whether the font ID is one of the larger VGA fonts.  (Each of the base fonts has a corresponding VGA font.)

## VgaBaseToVgaFont

**Purpose**      Get the VGA font ID that corresponds to the base font.

**Prototype**      `FontID VgaBaseToVgaFont(FontID font);`

**Parameters**      `->font`       Standard base font ID.

**Result**      Return the corresponding VGA font.

**Compatibility**      Implemented only if VGA Extension 1.0 is present.

**Comments**      Each of the base fonts has a corresponding VGA font.  VGA fonts are 1.5 times larger.

---

### VgaVgaToBaseFont

**Purpose**      Get the standard base font ID that corresponds to the larger VGA font.

**Prototype**    `FontID VgaVgaToBaseFont(FontID font);`

**Parameters**   `->font`      Standard base font ID.

**Result**       Return the corresponding base font.

**Compatibility**  Implemented only if VGA Extension 1.0 is present.

**Comments**     Each of base fonts has a corresponding VGA font.  VGA fonts are 1.5 times larger.

## VgaSetScreenMode

**Purpose**      Set the screen mode and rotation of a form within an application.

**Prototype**    ```
Err VgaSetScreenMode(VgaScreenModeType mode,
                     VgaRotateModeType rotation);
```

**Parameters**   `-> mode`                      Screen mode.

                 `-> rotation`                  Rotation mode.

**Result**       `errNone`                      Success

                 VgaErrModeUnsupported          Invalid mode and rotation combination.

**Compatibility**   Implemented only if VGA Extension 1.0 is present.

**Comments**

---

## VgaRestoreScreenState

**Purpose**    Restore the screen state, which includes mode, rotation, and offset of a form within an application.

**Prototype**    `Err VgaRestoreScreenState(VgaScreenStateype state);`

**Parameters**    `-> state`                Pointer to a structure specifying a saved screen state.  (Retrieved via a call to VgaGetScreenState())

**Result**    `errNone`                Success

`VgaErrModeUnsupported`    Invalid mode and rotation combination.

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**    This function should be used in conjunction with VgaGetScreenState().

## VgaRotateSelect

**Purpose**        Displays a dialog box in which the user can choose from one of four screen
                   rotations.

**Prototype**      `VgaRotateModeType VgaRotateSelect(VgaRotateModeType mode);`

**Parameters**     `-> mode`                           Default highlighted rotation.

**Result**         One of the following values:
                   `rotateModeNone`
                   `rotateMode90`
                   `rotateMode180`
                   `rotateMode270`

**Compatibility**  Implemented only if VGA Extension 1.0 is present.

**Comments**

---

## VgaTableUseBaseFont

**Purpose**     To change the fonts used with a table for a specific form.  By default, the table will be drawn using the VGA version of the fonts.  This function call allows the font version (either base or VGA) to be specified.

**Prototype**
```
void VgaTableUseBaseFont(TablePtr tableP,
                         Boolean  useBaseVersion);
```

**Parameters**     `-> tableP`                    Pointer to the table.

`-> useBaseVersion`            True if the base font is to be used.
                               False if the VGA font is to be used.

**Result**     `None`

**Compatibility**     Implemented only if VGA Extension 1.0 is present.

**Comments**     **Note:**  this function does not affect any customTableItems, which have their own font properties.

## VgaBitmapExpandedExtent

**Purpose**      Used to determine the extent of a bitmap if scaled by 1.5.  Not intended for use by general applications.

**Prototype**    `void VgaBitmapExpandedExtent(BitmapPtr bmPtr,`
                                      `Coord *  extentX,`
                                      `Coord *  extentY);`

**Parameters**   `<- bmPtr`        Pointer to non-expanded a bitmap

                 `-> extentX`      Scaled X dimension

                 `-> extentY`      Scaled Y dimension

**Result**       `None`

**Compatibility**   Implemented only if VGA Extension 1.0 is present.

**Comments**     This function is useful for 3$^{rd}$ party launchers that need to scale icons, otherwise, the function is primarily for system use.

## VgaFontSelect

**Purpose**    Displays a dialog box that allow a user to select a new font.

**Prototype**
```
FontID VgaFontSelect(VgaFontSelectType selectFormType,
                     FontID fontID);
```

**Parameters**    `-> selectFormType`    vgaFontSelectBase = Display Palm OS FontSelect.

vgaFontSelectVgaText = Display form to select one of four standard base fonts or one of four VGA fonts.

`-> fontID`    Initial font ID to highlight.

**Result**    `FontID`    New selected font ID.

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**    Use this routine to force either the Standard Palm OS Font Select dialog or the HandEra version.

# VgaWinDrawBitmapExpanded

**Purpose**        Used to draw and scale a bitmap by 1.5.

**Prototype**
```
void VgaWinDrawBitmapExpanded(BitmapPtr bitmapP,
                              Coord    X,
                              Coord    Y);
```

**Parameters**    `<- bitmapP`    Pointer to a bitmap

                `X`                 X location

                `Y`                 Y location

**Result**        `None`

**Compatibility**    Implemented only if VGA Extension 1.0 is present.

**Comments**      This function is useful for 3[rd] party launchers that need to scale icons, otherwise, the function is primarily for system use.

# Section 4 – VGA Extension Error Codes

When an error occurs during a VGA API call, an indication of the error is returned by the function to the caller.   The error may be one of the codes defined in the Palm OS header files.  The most common error return codes are as follows:

| | |
|---|---|
| sysErrParamErr | Invalid parameter used with internal system function. |
| sysErrNoFreeResource | There is not enough memory to complete the function. |

The VGA Extension also defines new error codes.  These constant values are defined in `vga.h`.

| | |
|---|---|
| VgaErrUnimplemented | Function not implemented. |
| VgaErrBadParam | Invalid Parameter. |
| VgaErrModeUnsupported | Screen mode not supported. |
| VgaErrScreenLocked | Unable to lock the OS screen. |

# Chapter 3: Silk API

This chapter is intended to introduce the use of, and provide a reference to, the Silk Extension API procedures. This chapter is directed toward Palm OS application developers who wish to customize or access the Silkscreened Area from within their applications. It is assumed that the reader is familiar with the C programming language, in particular within the context of the Palm OS.

♦   Section 1 of this chapter gives background detail.

♦   Section 2 details the shared data structures used by functions in the Silk API and describes each of the API calls, describing their function, parameters, and return value.

♦   Section 3 lists the possible error codes and their interpretation.

## Section 1 – Overview

The term **Silk** refers to the silkscreened area shown in the figure below. The Silk Extension manages this virtualized area.



**Figure 3-1.  Silkscreen**

When Graffiti is enabled, the Silk Extension follows the user's pen and draws within these areas to aid in Graffiti entry. The developer may utilize the Silk Extension API to move these areas within the silkscreened area with the restriction that the numeric entry must be to the immediate right of the alpha entry area and the same height.

The Silk Extension also supports any number of buttons in the silkscreened area. The Silk API allows the developer to define the list of buttons, and provide a template to draw these buttons and their inverted state (when pushed).

The use of this Software Development Kit (SDK) allows the developer to modify the location and number of these controls in the silk area, draw the new controls, and specify their location.

### Silk Extension 1.0 Feature Set

Before making a Silk Extension API call, an application needs to ensure that the Silk Extension itself is present and is compatible with the API call. Attempting to make Silk Extension calls on a non-HandEra device will crash the application – this is an inherent limitation of any Palm OS extension. The application should make a `FtrGet` function call to determine if the extension is present and what its version number is.

```
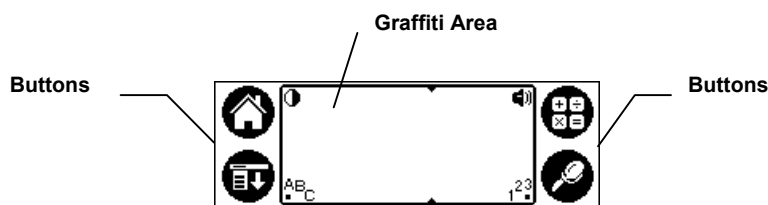UInt32 version;
if FtrGet(TRGSysFtrID, TRGSilkFtrNum, &version) == 0)
{
```

```
      if (sysGetROMVerMajor(version) >= 1)
      {
         // the Silk Extension 1.0 or higher is present
      }
   }
```

The Silk API calls may be grouped into two categories: silk window management and silk area and button management. The calls, grouped by category, are listed in the following tables, with brief descriptions of each call's function. An alphabetical listing with a detailed specification of each call is given in Section 2 – Silk API Function Reference on page 3-3.

### Table 3-1.  Silk Window Management

| | |
|---|---|
| SilkMaximizeWindow | Draw and activate the silkscreen window. |
| SilkMinimizeWindow | Erase and disable the silkscreen window. |
| SilkWindowMaximized | Returns state of the silkscreen window. |
| SilkGetWindow | Get the silkscreen window handle so it can be drawn on. |
| SilkGetTemplateBitmaps | Get bitmaps used by the template window |
| SilkSetTemplateBitmaps | Set bitmaps for the template windows to use. |
| SilkRestoreDefaultTemplates | Restore the original silkscreen window template and areas. |
| SilkGetPenEnabled | Returns the state of pen input in the silkscreen |
| SilkSetPenEnabled | Enable/disable pen input in the silkscreen |

### Table 3-2.  Silk Area and Button Management

| | |
|---|---|
| SilkGetButtonListSize | Returns the button list size. |
| SilkGetButtonList | Returns a pointer to the buttons. |
| SilkSetButtonList | Sets new buttons. |
| SilkGetAreas | Returns a point the graffiti areas. |
| SilkSetAreas | Sets the new graffiti areas. |
| SilkGetGraffitiPersistence | Returns current setting for graffiti persistence. |
| SilkSetGraffitiPersistence | Sets a new value for graffiti persistence. |

# Section 2 – Silk API Function Reference

This section contains an alphabetical listing of the functions available in the silk API, along with a brief description of each.

## Functions

### SilkMaximizeWindow

**Purpose**        Maximize the silkscreen.

**Prototype**      `void SilkMaximizeWindow(void);`

**Parameters**     `None`

**Result**         `None`

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**       If the silkscreen is already maximized, the function simply redraws the window.

## SilkMinimizeWindow

**Purpose**        Minimize the silkscreen

**Prototype**      `void SilkMinimizeWindow(void);`

**Parameters**     `None`

**Result**         `None`

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**       If the Silkscreen is already minimized, the function simply redraws the window.

## SilkWindowMaxmized

**Purpose**          Used to determine the current state of the silk window.

**Prototype**        `Boolean SilkWindowMaximized(void);`

**Parameters**       `None`

**Result**           Returns true if the window is maximized, false if the window is minimized

**Compatibility**    Implemented only if Silk Extension 1.0 is present.

**Comments**

## SilkGetWindow

**Purpose**      Return a WinHandle to the silkscreen window to allow drawing directly to onscreen window.

**Prototype**      `WinHandle SilkGetWindow(void);`

**Parameters**      None

**Result**      Handle to onscreen window for the silkscreen.

**Compatibility**      Implemented only if Silk Extension 1.0 is present.

**Comments**      The silkscreen window is redrawn from the template window to erase Graffiti within it or handle button presses.  Drawing directly to this window should only be used for animation.  In addition, applications running in Scale-To-Fit mode will need to temporarily disable the VGA extension while drawing to this window.

### SilkGetTemplateBitmaps

**Purpose**  Returns pointers to the bitmaps used for the silkscreen template.

**Prototype**
```
Err SilkGetTemplateBitmaps(
                BitmapPtr * maxSilkTemplate,
                BitmapPtr * selectedMaxSilkTemplate,
                BitmapPtr * minSilkTemplate,
                BitmapPtr * selectedMinSilkTemplate);
```

**Parameters**  `-> maxSilkTemplate`  A pointer to a maximized silkscreen bitmap ptr used to be used in the silk area.  Pass NULL for this parameter if you don't want to retrieve it.

`-> selectedMaxSilkTemplate`  A pointer to an inverted bitmap ptr used to draw a pushed button in the silk area. Pass NULL for this parameter if you don't want to retrieve it.

`-> minSilkTemplate`  A pointer to a bitmap ptr used in the minimized silk area. Pass NULL for this parameter if you don't want to retrieve it.

`-> selectedMinSilkTemplate`  A pointer to a bitmap ptr used to draw a pushed minimized button in the silk area. Pass NULL for this parameter if you don't want to retrieve it.

**Returns**  `Err`

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**

## SilkSetTemplateBitmaps

**Purpose**      Provide bitmaps for the silkscreen template and redraw the silkscreen window with the new bitmaps.

**Prototype**    ```
Err SilksetTemplateBitmaps(
                    BitmapPtr maxSilkTemplate,
                    BitmapPtr selectedMaxSilkTemplate,
                    BitmapPtr minSilkTemplate,
                    BitmapPtr selectedMinSilkTemplate);
```

**Parameters**   `-> maxSilkTemplate`          A pointer to a user bitmap that will be used in the silk area or NULL.

             `-> selectedMaxSilkTemplate`  A pointer to a user bitmap that will be used to draw a pushed button in the silk area or NULL

             `-> minSilkTemplate`          A pointer to a user bitmap that will be used in the minimized silk area or NULL

             `-> selectedMinSilkTemplate`  A pointer to a user bitmap that will be used to draw a pushed minimized button in the silk area or NULL.

**Returns**      `Err`

**Compatibility** Implemented only if Silk Extension 1.0 is present.

**Comments**

# SilkRestoreDefaultTemplate

**Purpose**       Restore the silkscreen template to its default.

**Prototype**      `Err SilkRestoreDefaultTemplate(void);`

**Parameters**    None

**Result**        `Err`

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**     This routine restores the default templates and draws the active bitmap to the screen.  It also restores the default areas and buttons.

SilkRestoreDefaultTemplate() is called on reset.  It is up to the developer to reload their templates at reset.

## SilkGetPenEnabled

**Purpose**       Returns the current state of all pen input into the Silkscreen area.

**Prototype**     `Err SilkGetPenEnabled(Boolean enabled);`

**Parameters**    enabled          Set to false to disable all pen input into the silkscreen.

**Result**        `Err`

**Compatibility** Implemented only if Silk Extension 1.0 is present.

**Comments**      System use only.

## SilkSetPenEnabled

**Purpose**       Disable or enable pen input into the Silkscreen area.

**Prototype**     `Err SilkSetPenEnabled(Boolean enabled);`

**Parameters**    enabled          Set to false to disable all pen input into the silkscreen.

**Result**        `Err`

**Compatibility** Implemented only if Silk Extension 1.0 is present.

**Comments**      System use only.

## SilkGetButtonListSize

**Purpose**    Get the silkscreen button list size.  Caller should call this function prior to calling SilkGetButtonList() to ensure a buffer large enough to receive the data.

**Prototype**    `UInt16 SilkGetButtonListSize(Boolean maximized);`

**Parameters**    `maximized`    Set  to true to get the maximized silkscreen button list size,

Set to false to get the minimized silkscreen button list size

**Result**    Size of the structure required for SilkGetButtonList().

**Compatibility**    Implemented only if Silk Extension 1.0 is present.

**Comments**

## SilkGetButtonList

**Purpose**         Get the silkscreen button list.

**Prototype**       `Err SilkGetButtonList(PenBtnListType *buttonList,`
                                   `Boolean        maximized);`

**Parameters**      `<- buttonList`   Pointer to the button list

                `maximized`       Set to true to get the maximized silkscreen button list
                                 Set to false to get the minimized silkscreen button list

**Result**          `Err`

**Compatibility**   Implemented only if Silk Extension 1.0 is present.

**Comments**        PenBtnListType is defined in the PalmOS 3.5 SDK header file "SysEvtMgr.h".

                      Make sure to call SilkGetButtonListSize() first to determine how much memory
is needed for buttonList.

## SilkSetButtonList

**Purpose**     Set the silkscreen button list.

**Prototype**   `Err SilkSetButtonList(PenBtnListType *buttonList,`
                              `Boolean        maximized);`

**Parameters**  `-> buttonList`    Pointer to PenBtnListType structure containing the new
                                   button information for the silk area.

                `maximized`        Set to true to set the maximized silkscreen button list
                                   Set to false to set the minimized silkscreen button list

**Result**      `Err`

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**    PenBtnListType is defined in the PalmOS 3.5 SDK header file "SysEvtMgr.h".

                This function copies the contents pointed to by buttonList.  Caller is
                responsible for freeing local copy.

## SilkGetAreas

**Purpose**        Get the silkscreen alpha and numeric areas.

**Prototype**      `Err SilkGetAreas(RectangleType *alphaEntry, RectangleType *numericEntry);`

**Parameters**    `<- alphaEntry`     Pointer to application allocated RectangleType structure that will be filled with the current information for the alphabetic character entry rectangle. Pass NULL for this parameter if you don't want to retrieve it.

                `<- numericEntry`   Pointer to application allocated RectangleType structure that will be filled with the current information for the numeric character entry rectangle. Pass NULL for this parameter if you don't want to retrieve it.

**Result**        Err

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**

---

## SilkSetAreas

**Purpose**      Set the silkscreen alpha and numeric areas.

**Prototype**    `Err SilkSetAreas(RectangleType *alphaEntry, RectangleType *numericEntry);`

**Parameters**   `-> alphaEntry`     Pointer to rectangle within the silk area where alphabetical characters will be recognized or NULL

                 `-> numericEntry`   Pointer to rectangle within the silk area where numeric characters will be recognized or NULL

**Result**       `Err`

**Compatibility**   Implemented only if Silk Extension 1.0 is present.

**Comments**     The numericEntry rectangle must be to the immediate right of the alphaEntry rectangle and the same height.

## SilkGetGraffitiPersistence

**Purpose**        Get number of timer ticks Graffiti remains on the silkscreen.

**Prototype**      `UInt32 SilkGetGraffitiPersistence(void);`

**Parameters**     `None`

**Result**         Number of timer ticks Graffiti remains on the silkscreen.

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Comments**       Use SysTicksPerSecond() to find the number of ticks per second.

# SilkSetGraffitiPersistence

**Purpose**        Set number of timer ticks Graffiti remains on the silkscreen.

**Prototype**      `void SilkSetGraffitiPersistence(Uint32 ticks);`

**Parameters**     `-> ticks`        Number of timer ticks Graffiti should remain on the silkscreen.

**Compatibility**  Implemented only if Silk Extension 1.0 is present.

**Result**         `None`

## Section 3 – Silk API Error Codes

When an error occurs during a Silk API call, an indication of the error is returned by the function to the caller.   The error may be one of the codes defined in the Palm OS header files.  The most common error return codes are as follows:

| | |
|---|---|
| sysErrParamErr | Invalid parameter used with internal system function. |
| sysErrNoFreeResource | There is not enough memory to complete the function. |
| dmErrCantOpen | Resource database cannot be opened |

The Silk Extension also defines a new error code. Its value is defined in `silk.h`.

| | |
|---|---|
| SilkErrBadParam | A parameter passed as an argument to one of the Silk API functions is invalid. |

# Chapter 4: Audio API

This chapter is intended to introduce the use of, and provide a reference to, the Audio Extension API procedures.  This chapter is directed toward Palm OS application developers who wish to customize or access the audio capabilities of the HandEra 330 from within their applications.  It is assumed that the reader is familiar with the C programming language, in particular within the context of the Palm OS.

♦   Section 1 of this chapter gives background detail.

♦   Section 2 details the shared data structures used by functions in the Audio API and describes each of the API calls, describing their function, parameters, and return value.

♦   Section 3 lists the possible error codes and their interpretation.

## Section 1 – Overview

### Audio Extension 1.0 Feature Set

Before making an Audio Extension API call, an application needs to ensure that the Audio Extension itself is present and is compatible with the API call.  Attempting to make Audio Extension calls on a non-HandEra device will crash the application – this is an inherent limitation of any Palm OS extension.  The application should make a `FtrGet` function call to determine if the extension is present and what its version number is.

```
UInt32 version;
if FtrGet(TRGSysFtrID, TRGAudioFtrNum, &version) == 0)
{
   if (sysGetROMVerMajor(version) >= 1)
   {
      // Audio Extension 1.0 or higher is present
   }
}
```

The Audio API calls may be grouped into four categories: general, volume control, record and playback, and DTMF.  The calls, grouped by category, are listed in the following tables, with brief descriptions of each function.

**Table 9-1.  General**

| AudGetSupportedFeatures | Find out what features are supported on this device. |
|---|---|

**Table 9-2.  Volume Control**

| AudioVolumeDlg | Display the volume dialog. |
|---|---|
| AudGetMasterVolume | Get the current volume setting. |
| AudSetMasterVolume | Set the volume. |
| AudGetMute | Get the mute status. |

| AudSetMute | Set the mute status. |
|---|---|

**Table 9-3.  Record and Playback**

| AudioCreateWave | Prepare for recording wave formatted data |
|---|---|
| AudioOpenWave | Prepare for playing wave formatted data. |
| AudioCloseWave | Close wave formatted data. |
| AudioPlayData | Start playing. |
| AudioRecordData | Start recording. |
| AudioPause | Stop playing or recording. |
| AudioSeek | Move to a new location in the data based on time. |
| AudioSeekPercent | Move to a new location in the data based on percent. |
| AudioTell | Get the current location in the data based on time. |
| AudioTellPercent | Get the current location in the data based on percent. |
| AudioOpenRawData | Prepare for playing or recording custom formatted data. |
| AudioCloseRawData | Close custom formatted data. |

**Table 9-3.  DTMF**

| AudPlayDTMFChar | Play a DTMF tone. |
|---|---|
| AudPlayDTMFStr | Play a sequence of DTMF tones. |

## Audio Extension 1.1 Feature Set

The application should make a `FtrGet` function call to determine if the extension is present and what its version number is.

```
UInt32 version;
if FtrGet(TRGSysFtrID, TRGAudioFtrNum, &version) == 0)
{
   if ((sysGetROMVerMajor(version) >= 1) &&
       (sysGetROMVerMinor(version) >= 1))
   {
      // Audio Extension 1.1 or higher is present
   }
}
```

**Table 9-4.  Amplifier Control**

| AudioAmplifierOn | Turns the amplifier circuitry on. |
|---|---|
| AudioAmplifierOff | Turns the amplifier circuitry off. |

# Section 2 – Audio API Data Reference

This section details the enumerated data types and structures used by the API functions.


## *AudioModeType*

| | |
|---|---|
| audioPlayMode | Setup for playback. |
| audioRecordMode | Setup for recording. |

## *AudioProgressType*

| | |
|---|---|
| UInt16 | Percent completed |
| UInt16 | Minutes |
| UInt16 | Seconds |
| UInt16 | 1/100$^{th}$ Seconds |


## *audioProgressEvent*

`audioProgressEvents` are generated while the `AudioPlayData` and `AudioRecordData` functions are in progress. The returned event includes the `AudioProgressType` as its data.

While the `AudioPlayData` function is in progress, `audioProgressEvents` are generated at the rate of 4/second.

While the `AudioRecordData` function is in progress, `audioProgressEvents` are generated at the rate of 2/second.

# Section 3 – Audio API 1.0 Function Reference

This section contains an alphabetical listing of the functions available in the audio API, along with a brief description of each.

## Functions

### AudGetSupportedFeatures

**Purpose**  Returns a bitmap of features supported by this implementation.

**Prototype**  Err AudGetSupportedFeatures(UInt32 *features);

**Parameters**
```
features - bitmap:
0x0001 audioFtrPlayWave
0x0002 audioFtrAjdVolume
0x0004 audioFtrDTMF
0x0008 audioFtrRecordWave
```

**Result**  `Err`

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

# AudioVolumeDlg

**Purpose**        Display the volume dialog and allow the user to change the volume.

**Prototype**      Err AudioVolumeDlg(void);

**Parameters**     None

**Result**         Err

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudGetMasterVolume

**Purpose**     Get the current volume setting.

**Prototype**    Err AudGetMasterVolume(UInt8 *volume);

**Parameters**   `->volume`          volume setting, range is 0 (no sound) to 255 (max volume).

**Result**       `Err`

**Compatibility** Implemented only if Audio Extension 1.0 is present.

**Comments**

# AudSetMasterVolume

**Purpose**        Set the volume.

**Prototype**      Err AudSetMasterVolume(UInt8 volume);

**Parameters**     `volume`            volume setting, range is 0 (no sound) to 255 (max volume).

**Result**         `Err`

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

### AudGetMute

**Purpose**      Get the mute status, which overrides the volume setting.

**Prototype**    Err AudGetMute(Boolean *mute);

**Parameters**   `->mute`          Returned mute status

**Result**       `Err`

**Compatibility** Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudSetMute

**Purpose**   Set the mute status, which overrides the volume setting.

**Prototype**   Err AudSetMute(Boolean mute);

**Parameters**   `mute`            New mute setting

**Result**   `Err`

**Compatibility**   Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioCreateWave

**Purpose**      Create wave format data and parse the headers.

**Prototype**    Err AudioCreateWave(AudioFormatType *dataFormat,
                          AudWriteProcPtr   writeData,
                          void *                userData);

**Parameters**   dataFormat      Returned structure describing the format of the data.

                 writeData       Callback function to write out recorded data.

                 userData        Pointer to optional user data passed on to the callback
                                 functions.

**Result**       Err

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioOpenWave

**Purpose**        Open wave format data for playback and parse the headers.

**Prototype**      Err AudioOpenWave( AudioFormatType *dataFormat,
                                AudReadProcPtr   getData,
                                void *                  userData);

**Parameters**     dataFormat        Returned structure describing the format of the data.

                   getData           Callback function to get more data to play

                   userData          Pointer to optional user data passed on to the callback
                                     functions.

**Result**         Err

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioCloseWave

**Purpose**      Close an open wave.  Must be called for each AudioOpenWave after done
                 playing or recording.

**Prototype**    Err AudioCloseWave(void);

**Parameters**   None

**Result**       Err

**Compatibility**   Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioPlayData

**Purpose**      Play data opened with AudioOpenWave or AudioOpenRawData in play mode.

**Prototype**      Err AudioPlayData(void);

**Parameters**      `None`

**Result**      `Err`

**Compatibility**      Implemented only if Audio Extension 1.0 is present.

**Comments**      `audioProgressEvents` are generated while the `AudioPlayData` function is in progress.  The returned event includes the `AudioProgressType` as its data.  While the `AudioPlayData` function is in progress, `audioProgressEvents` are generated at the rate of 4/second

## AudioRecordData

**Purpose**     Record data opened with AudioOpenWave or AudioOpenRawData in record mode.

**Prototype**     Err AudioRecordData(void);

**Parameters**     None

**Result**     `Err`

**Compatibility**     Implemented only if Audio Extension 1.0 is present.

**Comments**     `audioProgressEvents` are generated while the `AudioRecordData` function is in progress.  The returned event includes the `AudioProgressType` as its data.  While the `AudioRecordData` function is in progress, `audioProgressEvents` are generated at the rate of 2/second.

## AudioPause

**Purpose**     Pause/Stop recording or playing.

**Prototype**     Err AudioPause(void);

**Parameters**     `None`

**Result**     `Err`

**Compatibility**     Implemented only if Audio Extension 1.0 is present.

**Comments**

### AudioSeek

**Purpose**      Seek to a new location in the data.  Currently only valid in play mode.

**Prototype**    Err AudioSeek(UInt32 tenthsOfSeconds);

**Parameters**   tenthsOfSeconds      Time to seek to in 1/10ths of a second.

**Result**       Err

**Compatibility** Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioSeekPercent

**Purpose**       Seek to a new location in the data.  Only valid in play mode.

**Prototype**     Err AudioSeekPercent(UInt16 percent);

**Parameters**    percent              Location to seek to 0=begining, 100 = end.

**Result**        Err

**Compatibility** Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioTell

**Purpose**        Report current position in data.

**Prototype**      Err AudioTell(UInt32 *tenthsOfSeconds);

**Parameters**    ->tenthsOfSeconds    Returned current time position in data in 1/10ths of a second.

**Result**          `Err`

**Compatibility**   Implemented only if Audio Extension 1.0 is present.

**Comments**

# AudioTellPercent

**Purpose**        Report current position in the data.

**Prototype**      Err AudioTellPercent(UInt16 *percent);

**Parameters**    ->percent        Returned percent position in data.

**Result**          `Err`

**Compatibility**   Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioOpenRawData

**Purpose**  Open raw PWM data.  It is up to the caller to parse or write out the headers for the format being used.

**Prototype**  Err AudioOpenRawData(AudioModeType    mode,
                             AudioFormatType *dataFormat,
                             AudReadProcPtr   getData,
                             AudWriteProcPtr   writeData,
                             void *                  userData);

**Parameters**  mode              Play or record mode

               dataFormat        Passed in structure describing the format of the data.

               getData           Callback function to get more data to play.

               writeData         Callback function to write out recorded data.

               userData          Pointer to optional user data passed on to the callback functions.

**Result**  Err

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudioCloseRawData

**Purpose**       Close raw data.  Must be called for each AudioOpenRawData after done
                  playing or recording.

**Prototype**     Err AudioCloseRawData(void);

**Parameters**    None

**Result**        Err

**Compatibility** Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudPlayDTMFChar

**Purpose**        Play a DTMF tone.

**Prototype**      Err AudPlayDTMFChar(char ascChar, Int16 toneLength);

**Parameters**     `ascChar`        DTMF character

                   `toneLength`     Length of the tone (in 1/16$^{th}$ of a second increments)

**Result**         `Err`

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Comments**

## AudPlayDTMFStr

**Purpose**        Play a sequence of DTMF tones.

**Prototype**      Err AudPlayDTMFStr(char *ascStr, Int16 toneLength, Int16 toneGap);

**Parameters**     `ascStr`         Null-terminated string of DTMF tones

                     `toneLength`     Length of the tone (in $1/16^{th}$ of a second increments)

                     `toneGap`        Length of the gap between tones (in $1/16^{th}$ of a second increments)

**Compatibility**  Implemented only if Audio Extension 1.0 is present.

**Result**         `Err`

**Comments**

# Section 4 – Audio API 1.1 Function Reference

This section contains an alphabetical listing of the functions available in the audio API, along with a brief description of each.

## Functions

### AudioAmplifierOn

| | |
|---|---|
| **Purpose** | Turns the amplifier circuitry on. |
| **Prototype** | Err AudioAmplifierOn() |
| **Parameters** | None |
| **Result** | `Err` |
| **Compatibility** | Implemented only if Audio Extension 1.1 is present. |
| **Comments** | This should be called prior to issuing a series of AudioPlay() calls. It prevents the amplifier circuitry from intermittently powering down the amplifier. |

# AudioAmplifierOff

**Purpose**      Turns the amplifier circuitry off.

**Prototype**    Err AudioAmplifierOff(void);

**Parameters**   `None`

**Result**       `Err`

**Compatibility**  Implemented only if Audio Extension 1.1 is present.

**Comments**     This should be called immediately after AudioAmplifierOn() has been called, followed by a series of AudioPlay() calls. It forces the amplifier circuitry to power down.  Failure to call this function after issuing an AudioAmplifierOn() will result in battery drain.

# Section 5 – Audio API Error Codes

When an error occurs during an Audio API call, some indication of the error is returned by the function to the caller.  The error may be one of the codes defined in the Palm OS header files.  The most common error return codes are as follows:

| | |
|---|---|
| sysErrParamErr | Invalid parameter used with internal system function. |
| memErrNotEnoughSpace | There is not enough memory to complete the function. |

The Audio Extension also defines new error codes.  These constant values are defined in `audio.h`.

| | |
|---|---|
| audioErrUnimplemented | Function not implemented (on this hardware). |
| audioErrBadParam | Invalid Parameter. |
| audioErrInvalidData | Bad wave data. |
| audioErrUnsupportedFormat | Unsupported play/record format. |

# Index