AlphaSmart, Inc.

Dana API Reference

Version 1.2
Date: 4/19/2002

CONTACT INFORMATION

AlphaSmart, Inc.

973 University Ave.
Los Gatos, CA 95032
U.S.A.
408-355-1000
www.alphasmart.com

Palm Computing World Wide Web www.palm.com

Metrowerks World Wide Web www.metrowerks.com

Table of Contents

About This Document

The *Dana API Reference* is part of the Dana Software Development Kit (SDK). AlphaSmart intends this document as a guide for developers interested in creating applications for Dana.

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 www.palmos.com. Relevant documents include:

- *Palm OS Reference*
- *Palm OS Companion*
- *Constructor for Palm OS*

This document assumes the reader is familiar with basic Palm OS concepts detailed in the *Palm OS 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 Companion*, are italicized.

Parts of the Dana feature set are based on technology licensed from HandEra, Inc. The APIs in the Dana SDK are similar to those in the HandEra SDK, but some modifications have been made in order to support the Dana hardware platform - in particular Dana's 160 x 560 pixel screen. Developers who have already modified their applications for the HandEra platform should find modifying their application for the Dana platform even simpler, since they are already familiar with many of the APIs and supporting Dana's larger screen requires fewer modifications

Chapter 1: Screen Extension

The Screen Extension provides an interface to the larger screen size on the Dana. The interface provides calls to determine the user interface area of the screen and rotation of the drawing. The Screen Extension also provides backward compatibility modes to existing Palm OS applications written for the standard Palm sized screens of 160 x 160. The use of this Software Development Kit (SDK) allows the developer to utilize the extra screen area on the device.

Screen Extension Feature Set

Before making a Screen Extension API call, an application needs to ensure that the Screen Extension itself is present and is compatible with the API call. Attempting to make Screen Extension calls on a non-Dana 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(AlphaSmartSysFtrID, ScrnFtrNum, &version) == 0)
{
   if (sysGetROMVerMajor(version) >= 1)
   {
      the Screen Extension 1.0 is present
   }
}
```

Another method is to use the macro provided in the Screen.h file.

```
UInt32 version;
if (_ScreenFeaturePresent (&version) == true)
{
    // use Screen API here

}
```

The Screen Extension calls are mainly focused on screen management. There is one call that assists a user in changing the default table font used by the Palm OS.

and font management. The calls, grouped by category, are listed in the following tables, with brief descriptions of each function.

Screen Management Functions

ScrnGetRotateMode	Get the current rotation mode.
ScrnSetRotateMode	Set new rotation mode.
ScrnTableSetFontID	Set the table font used by the Palm OS
ScrnGetSystemState	Get the current screen state

Screen API Types

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

ScrnScreenModeType

screenModeWideTall	Application is being displayed in the full screen.
screenModeCentered	Application is being displayed as a legacy application.

ScrnRotateModeType

RotateScrnMode0	No rotation.
RotateScrnMode90	Screen rotates 90 degrees.
RotateScrnMode270	Screen rotates 270 degrees.

Screen API Functions

ScrnGetRotateMode

Purpose Get the current rotation mode.

Prototype

```
void ScrnGetRotateMode (ScrnRotateModeType *rotation);
```

Parameters

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

Result None

Compatibility Implemented only if Screen Extension is present.

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

ScrnSetRotateMode

Purpose Set the rotation mode of the system.

Prototype

```
Err ScrnSetRotationMode(ScrnRotateModeType rotation);
```

Parameters	`-> rotation`	Rotation mode.
Result	`errNone`	Success
	ScrnErrModeUnsupported	Invalid mode and rotation combination.

Compatibility Implemented only if Screen Extension is present.

Comments

ScrnTableSetFontID

Purpose Change the font that the Palm OS uses for a table.

Prototype

```
void ScrnTableSetFontID(TablePrt table, FontID fontID);
```

Parameters

`-> table`	Palm OS table pointer.
`-> fontID`	The font that the Palm OS should use in handling the table.

Result `None`

Compatibility Implemented only if Screen Extension is present.

Comments

ScrnGetSystemState

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

Prototype
```
void ScrnGetSystemState(ScrnSystemStateType *state);
```

Parameters `<- state` Current screen state

Result None

Compatibility Implemented only if Screen Extension is present.

Comments This function should be used instead of ScrnGetScreenMode() for applications that may be sublaunched by 3rd party applications, such as phone-lookup functions.

Screen Extension Error Codes

When an error occurs during a Screen 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 Screen Extension also defines new error codes. These constant values are defined in Screen.h.

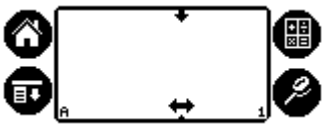
scrnErrUnimplemented	Function not implemented.
scrnErrBadParam	Invalid Parameter.
scrnErrModeUnsupported	Screen mode not supported.
scrnErrScreenLocked	Unable to lock the OS screen.

Chapter 2: WritePad Extension

This chapter is intended to introduce the use of, and provide a reference to, the WritePad Extension APIs. This chapter is directed toward Palm OS application developers who wish to customize or access the WritePad 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.

Overview

The term **WritePad** refers to the area of the screen shown below. The WritePad Extension manages this virtualized area for stylus input.



WritePad area

The WritePad area is not always available to the Dana user. If the application takes advantage of the full screen in 0 degree rotation then the WritePad area is not available to the user.

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

The WritePad Extension also supports any number of buttons in the WritePad area. The WritePad 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 WritePad area, draw the new controls, and specify their location.

WritePad Extension Feature Set

Before making a WritePad Extension API call, an application needs to ensure that the WritePad Extension itself is present and is compatible with the API call. Attempting to make WritePad Extension calls on a non-Dana 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(AlphaSmartSysFtrID, WritePadFtrNum, &version) == 0)
{
   if (sysGetROMVerMajor(version) >= 1)
   {
      // the WritePad Extension 1.0 or higher is present
   }
}
```

Another method is to use the macro provided in the WritePad.h file.

```
UInt32 version;
if (_WritePadFeaturePresent(&version) == true)
{
    // use WritePad API here

}
```

The WritePad API calls may be grouped into two categories: WritePad window management and WritePad area and button management. The calls, grouped by category, are listed in the following tables, with brief descriptions of each call's function. A listing with a detailed specification of each call is given in next section.

WritePad Window Management

WrtpMaximizeWindow	Draw and activate the WritePad window.
WrtpMinimizeWindow	Erase and disable the WritePad window.
WrtpWindowMaximized	Returns state of the WritePad window.
WrtpWindowRight	Returns whether WritePad is on the right or left in legacy mode
WrtpGetWindow	Get the WritePad window handle so it can be drawn on.
WrtpGetTemplateBitmaps	Get bitmaps used by the template window
WrtpSetTemplateBitmaps	Set bitmaps for the template windows to use.
WrtpRestoreDefaultTemplates	Restore the original WritePad window template and areas.
WrtpGetPenEnabled	Returns the state of pen input in the WritePad
WrtpSetPenEnabled	Enable/disable pen input in the WritePad

WritePad Area and Button Management

WrtpGetButtonListSize	Returns the button list size.
WrtpGetButtonList	Returns a pointer to the buttons.
WrtpSetButtonList	Sets new buttons.
WrtpGetAreas	Returns a point the graffiti areas.
WrtpSetAreas	Sets the new graffiti areas.
WrtpGetGraffitiPersistence	Returns current setting for graffiti persistence.
WrtpSetGraffitiPersistence	Sets a new value for graffiti persistence.

WritePad API Functions

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

WrtpMaximizeWindow

Purpose Maximize the WritePad.

Prototype

```
void WrtpMaximizeWindow(void);
```

Parameters None

Result None

Compatibility Implemented only if WritePad Extension is present.

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

WrtpMinimizeWindow

Purpose Minimize the WritePad

Prototype
```
void WrtpMinimizeWindow(void);
```

Parameters None

Result None

Compatibility Implemented only if WritePad Extension is present.

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

WrtpWindowMaxmized

Purpose Used to determine the current state of the WritePad window.

Prototype

```
Boolean WrtpWindowMaximized(void);
```

Parameters None

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

Compatibility Implemented only if WritePad Extension is present.

Comments

WrtpWindowRight

Purpose Used to determine side of the legacy form that the WritePad is on.

Prototype

```
Boolean WrtpWindowRight(void);
```

Parameters None

Result Returns true if the window is on the right, false if the window is on the left.

Compatibility Implemented only if WritePad Extension is present.

Comments

WrtpGetWindow

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

Prototype

```
WinHandle WrtpGetWindow(void);
```

Parameters None

Result Handle to onscreen window for the WritePad.

Compatibility Implemented only if WritePad Extension is present.

Comments The WritePad 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 will need to temporarily disable the Screen extension while drawing to this window.

WrtpGetTemplateBitmaps

Purpose Returns pointers to the bitmaps used for the WritePad template.

Prototype

```
Err WrtpGetTemplateBitmaps(
                      BitmapPtr *maxWritePadTemplate,
                      BitmapPtr
*selectedMaxWritePadTemplate,
                      BitmapPtr *minWritePadTemplate,
                      BitmapPtr
*selectedMinWritePadTemplate);
```

Parameters

`-> maxWritePadTemplate`	A pointer to a maximized WritePad bitmap ptr used to be used in the WritePad area. Pass NULL for this parameter if you don't want to retrieve it.
`-> selectedMaxWritePadTemplate`	A pointer to an inverted bitmap ptr used to draw a pushed button in the WritePad area. Pass NULL for this parameter if you don't want to retrieve it.
`-> minWritePadTemplate`	A pointer to a bitmap ptr used in the minimized WritePad area. Pass NULL for this parameter if you don't want to retrieve it.
`-> selectedMinWritePadTemplate`	A pointer to a bitmap ptr used to draw a pushed minimized button in the WritePad area. Pass NULL for this parameter if you don't want to retrieve it.

Returns `Err`

Compatibility Implemented only if WritePad Extension is present.

Comments

WrtpSetTemplateBitmaps

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

Prototype

```
Err WrtpSetTemplateBitmaps(
                        BitmapPtr maxWritePadTemplate,
                        BitmapPtr
selectedMaxWritePadTemplate,
                        BitmapPtr minWritePadTemplate,
                        BitmapPtr
selectedMinWritePadTemplate);
```

Parameters

`-> maxWritePadTemplate`	A pointer to a user bitmap that will be used in the WritePad area or NULL.
`-> selectedMaxWritePadTemplate`	A pointer to a user bitmap that will be used to draw a pushed button in the WritePad area or NULL
`-> minWritePadTemplate`	A pointer to a user bitmap that will be used in the minimized WritePad area or NULL
`-> selectedMinWritePadTemplate`	A pointer to a user bitmap that will be used to draw a pushed minimized button in the WritePad area or NULL.

Returns `Err`

Compatibility Implemented only if WritePad Extension is present.

Comments

WrtpRestoreDefaultTemplate

Purpose Restore the WritePad template to its default.

Prototype
```
Err WrtpRestoreDefaultTemplate(void);
```

Parameters None

Result
```
Err
```

Compatibility Implemented only if WritePad Extension 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.

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

WrtpGetPenEnabled

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

Prototype

```
Err WrtpGetPenEnabled(Boolean enabled);
```

Parameters

enabled	Set to false to disable all pen input into the WritePad.

Result

```
Err
```

Compatibility Implemented only if WritePad Extension is present.

Comments System use only.

WrtpSetPenEnabled

Purpose Disable or enable pen input into the WritePad area.

Prototype

```
Err WrtpSetPenEnabled(Boolean enabled);
```

Parameters

enabled	Set to false to disable all pen input into the WritePad.

Result

```
Err
```

Compatibility Implemented only if WritePad Extension is present.

Comments System use only.

WrtpGetButtonListSize

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

Prototype

```
UInt16 WrtpGetButtonListSize(Boolean maximized);
```

Parameters

`maximized`	Set to true to get the maximized WritePad button list size, Set to false to get the minimized WritePad button list size

Result Size of the structure required for WritePadGetButtonList().

Compatibility Implemented only if WritePad Extension is present.

Comments

WrtpGetButtonList

Purpose Get the WritePad button list.

Prototype

```
Err WrtpGetButtonList(PenBtnListType *buttonList,
                      Boolean         maximized);
```

Parameters

`<- buttonList`	Pointer to the button list
`maximized`	Set to true to get the maximized WritePad button list Set to false to get the minimized WritePad button list

Result `Err`

Compatibility Implemented only if WritePad Extension is present.

Comments PenBtnListType is defined in the PalmOS 3.5 SDK header file “SysEvtMgr.h”.

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

WrtpSetButtonList

Purpose Set the WritePad button list.

Prototype

```
Err WrtpSetButtonList(PenBtnListType *buttonList,
                      Boolean         maximized);
```

Parameters

`-> buttonList`	Pointer to PenBtnListType structure containing the new button information for the WritePad area.
`maximized`	Set to true to set the maximized WritePad button list Set to false to set the minimized WritePad button list

Result `Err`

Compatibility Implemented only if WritePad Extension 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.

WrtpGetAreas

Purpose Get the WritePad alpha and numeric areas.

Prototype

```
Err WrtpGetAreas(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 WritePad Extension is present.

Comments

WrtpSetAreas

Purpose Set the WritePad alpha and numeric areas.

Prototype

```
Err WrtpSetAreas(RectangleType *alphaEntry, RectangleType
*numericEntry);
```

Parameters

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

Result `Err`

Compatibility Implemented only if WritePad Extension is present.

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

WrtpGetGraffitiPersistence

Purpose Get number of timer ticks Graffiti remains on the WritePad.

Prototype
```
UInt32 WrtpGetGraffitiPersistence(void);
```

Parameters None

Result Number of timer ticks Graffiti remains on the WritePad.

Compatibility Implemented only if WritePad Extension is present.

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

WrtpSetGraffitiPersistence

Purpose Set number of timer ticks Graffiti remains on the WritePad.

Prototype

```
void WrtpSetGraffitiPersistence(Uint32 ticks);
```

Parameters

`-> ticks`	Number of timer ticks Graffiti should remain on the WritePad.

Compatibility Implemented only if WritePad Extension is present.

Result `None`

WritePad Extension Error Codes

When an error occurs during a WritePad 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 WritePad Extension also defines a new error code. Its value is defined in WritePad.h.

wrtpErrBadParam	A parameter passed as an argument to one of the WritePad API functions is invalid.

Chapter 3: Keyboard Extension

This chapter is intended to introduce the use of Keyboard Extension API procedures. This chapter is directed toward Palm OS application developers who wish to access the current state of the Keyboard. It is assumed that the reader is familiar with the C programming language, in particular within the context of the Palm OS.

Keyboard Extension Feature Set

Before making a Keyboard Extension API call, an application needs to ensure that the Keyboard Extension itself is present and is compatible with the API call. Attempting to make Keyboard Extension calls on a non-Dana 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(AlphaSmartSysFtrID, WTAPKeyboardAccessFtrNum, &version) ==
0)
{
   if (sysGetROMVerMajor(version) >= 1)
   {
      the Keyboard Extension 1.0 is present
   }
}
```

Another method is to use the macro provided in the KeyboardAccess.h file.

```
UInt32 version;
if (_KybdFeaturePresent (&version) == true)
{
    // use Keyboard API here

}
```

Keyboard API Types

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

KeyboardLayoutType

kybdLayoutQwerty	QWERTY keyboard layout.
kybdLayoutDvorak	Dvorak keyboard layout.
kybdLayoutLeft	Left handed keyboard layout.
kybdLayoutRight	Right handed keyboard layout.

KeyboardEvent

MS byte	Modifier key bits.
LS byte	Icode (AlphaSmart keycode)

Keyboard API Functions

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

KybdGetLayout

Purpose Get the current layout of the keyboard.

Prototype

```
KeyboardLayoutType KybdGetLayout(void);
```

Parameters None

Result

`KeyboardLayoutType`	contains the current layout of the keyboard: QWERTY, Dvorak, Right Handed, or LeftHanded.

Compatibility Implemented only if Keyboard Extension is present.

Comments On non US versions, the function will always return QWERTY. The user can change the layout of the keyboard using the Keyboard application in the layout form.

KybdGetModifiers

Purpose Get the current state of the modifier keys on the keyboard.

Prototype

```
KeyboardEvent KybdGetModifiers(void);
```

Parameters None

Result

`KeyboardEvent`	contains the current state of the modifier keys. A key bit = 1 indicates that the key is down. A key bit = 0 indicates that the key is up. This state is the logical state of the key and not necessarily the physical state of the key. The modifier key states are affected by Sticky Keys.

Compatibility Implemented only if Keyboard Extension is present.

Comments The bit definitions for all the modifier keys can be found in WideTallAppChars.h.

KybdGetKeyboardEmulation

Purpose Get the current state of the keyboard emulation when connected to a computer..

Prototype

```
Boolean KybdGetKeyboardEmulation (void);
```

Parameters None

Result

true	Connected to a USB computer.
false	Not connected to a USB computer.

Compatibility Implemented only if Keyboard Extension is present.

Comments System use only.

KybdSetKeyboardEmulation

Purpose Set the state of the keyboard emulation.

Prototype
```
Boolean KybdSetKeyboardEmulation (Boolean value);
```

Parameters

`<- value`	Determines the state of the keyboard emulation.

Result `None`

Compatibility Implemented only if Keyboard Extension is present.

Comments System use only.