



Applications and Dynamic Input Areas

Palm OS® 5 SDK (68K) R3

CONTRIBUTORS

Written by Jean Ostrem and Christopher Bey

Engineering contributions by Grant Glouser, Rachid Hasnou, Ezekiel Sanborn de Asis, Andy Stewart

Copyright © 2003, PalmSource, Inc. and its affiliates. All rights reserved. This documentation may be printed and copied solely for use in developing products for Palm OS® software. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation or adaptation) without express written consent from PalmSource, Inc.

PalmSource, Inc. reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of PalmSource, Inc. to provide notification of such revision or changes.

PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO REPRESENTATIONS OR WARRANTIES THAT THE DOCUMENTATION IS FREE OF ERRORS OR THAT THE DOCUMENTATION IS SUITABLE FOR YOUR USE. THE DOCUMENTATION IS PROVIDED ON AN "AS IS" BASIS. PALMSOURCE, INC. AND ITS SUPPLIERS MAKE NO WARRANTIES, TERMS OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING WARRANTIES, TERMS, OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND SATISFACTORY QUALITY. TO THE FULL EXTENT ALLOWED BY LAW, PALMSOURCE, INC. ALSO EXCLUDES FOR ITSELF AND ITS SUPPLIERS ANY LIABILITY, WHETHER BASED IN CONTRACT OR TORT (INCLUDING NEGLIGENCE), FOR DIRECT, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, OR PUNITIVE DAMAGES OF ANY KIND, OR FOR LOSS OF REVENUE OR PROFITS, LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, OR OTHER FINANCIAL LOSS ARISING OUT OF OR IN CONNECTION WITH THIS DOCUMENTATION, EVEN IF PALMSOURCE, INC. OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Palm OS, Palm Computing, HandFAX, HandSTAMP, HandWEB, Graffiti, HotSync, iMessenger, MultiMail, Palm.Net, PalmPak, PalmConnect, PalmGlove, PalmModem, PalmPoint, PalmPrint, and PalmSource are registered trademarks of PalmSource, Inc. or its affiliates. Palm, the Palm logo, MyPalm, PalmGear, PalmPix, PalmPower, AnyDay, EventClub, HandMAIL, the HotSync logo, PalmGlove, Palm Powered, the Palm trade dress, Smartcode, Simply Palm, ThinAir, WeSync, and Wireless Refresh are trademarks of PalmSource, Inc. or its affiliates. All other product and brand names may be trademarks or registered trademarks of their respective owners.

IF THIS DOCUMENTATION IS PROVIDED ON A COMPACT DISC, THE OTHER SOFTWARE AND DOCUMENTATION ON THE COMPACT DISC ARE SUBJECT TO THE LICENSE AGREEMENT ACCOMPANYING THE COMPACT DISC.

Applications and Dynamic Input Areas
Document Number 3106-002
August 28, 2003

PalmSource, Inc.
1240 Crossman
Sunnyvale, CA 94089
USA
www.palmsource.com

Table of Contents

About This Document	v
What This Book Containsv
Additional Resourcesv
1 Applications and the Dynamic Input Area	1
The Dynamic Input Area Feature2
Size Constraints3
Input Area Policy5
Setting the Input Area Policy.5
Enabling the Input Trigger.6
Setting an Input Area State6
Resizing a Form7
Hiding and Showing the Control Bar	10
Pen Input Manager Compatibility.	10
New sysFtrNumInputAreaFlags Support	11
Additional winDisplayChangedEvent	11
Restoration of Input Trigger State.	12
New pinInputAreaUser Input Area State	12
New Stat... Functions	13
New Support for Changing Display Orientation	13
2 Pen Input Manager Reference	15
Pen Input Manager Constants	15
sysNotifyDisplayResizedEvent.	15
winDisplayChangedEvent.	16
Input Area States.	16
Input Trigger States.	17
Form Dynamic Input Area Policies	17
Orientation States	18
Orientation Trigger States	19
Pen Input Manager Functions	20
PINGetInputAreaState	20
PINGetInputTriggerState	20
PINSetInputAreaState	21

PINSetInputTriggerState	22
Other Functions	22
FrmGetDIAPolicyAttr	23
FrmSetDIAPolicyAttr	23
StatGetAttribute	24
StatHide	25
StatShow	25
SysGetOrientation	26
SysSetOrientation	27
SysGetOrientationTriggerState	27
SysSetOrientationTriggerState	28
WinSetConstraintsSize	29

Index

31

About This Document

This book describes how to write an application that works with a dynamic input area. A **dynamic input area** is a software implementation of the input area that is traditionally silkscreened onto the device. Implementing the area as software allows the user to expand and collapse the area at will, giving more space to the display of application data when it is needed.

What This Book Contains

This book contains the following information:

- [Chapter 1, “Applications and the Dynamic Input Area,”](#) on page 1 describes how to modify your application so that it responds to the dynamic input area appropriately.
- [Chapter 2, “Pen Input Manager Reference,”](#) on page 15 provides reference material for the new APIs.

Additional Resources

- Documentation
PalmSource publishes its latest versions of documents for Palm OS developers at
<http://www.palmos.com/dev/support/docs/>
- Training
PalmSource and its partners host training classes for Palm OS developers. For topics and schedules, check
<http://www.palmos.com/dev/training>

About This Document

Additional Resources

- Knowledge Base

The Knowledge Base is a fast, web-based database of technical information. Search for frequently asked questions (FAQs), sample code, white papers, and the development documentation at

<http://www.palmos.com/dev/support/kb/>

Applications and the Dynamic Input Area

On most existing Palm Powered™ handhelds, Palm OS® applications are drawn in a fixed space that is 160 X 160 pixels or 320 X 320 pixels in size, depending on the device's screen density. The input area, where the user enters characters, is silkscreened onto such devices. Some Palm Powered handhelds have **dynamic input areas**, as shown in [Figure 1.1](#), that users can close when they do not need to enter characters. (Tapping the arrow in the lower right closes the input area, leaving a smaller control bar visible.)

On these devices, the application area is not fixed—it is the traditional square size while the input area is opened, and it is a rectangular size when the input area is closed, giving more space to the application.

Figure 1.1 Closable Dynamic Input Area



A control bar at the bottom of the screen, shown in [Figure 1.2](#), contains various buttons, including one called the trigger (the arrow at the right in this example), that opens the input area when it is closed. A trigger also typically appears in the input area to provide a way for the user to close the input area.

Figure 1.2 Control Bar



Applications and the Dynamic Input Area

The Dynamic Input Area Feature

The control bar is typically closed when the input area is open, except on double density screens, when it is always shown; however, licensees can change this behavior.

To take advantage of the additional screen space, your application must respond when the input area is opened or closed and redraw the current form in the available application space. This document describes how to do so.

To make your application work with a dynamic input area, do the following:

1. Check the feature to make sure that the API described in this document is available. (See [“The Dynamic Input Area Feature”](#) on page 2.)
2. Call [WinSetConstraintsSize](#) to set the size constraints for each form in response to the `frmLoadEvent` or `frmOpenEvent`. (See [“Size Constraints”](#) on page 3.)
3. Set up each form in your application to work with the Pen Input Manager in response to the `frmLoadEvent` or `frmOpenEvent`, and the `winEnterEvent`. This involves calling [FrmSetDIAPolicyAttr](#), [PINSetInputTriggerState](#), and [PINSetInputAreaState](#). (See [“Input Area Policy”](#) on page 5.)
4. Respond to the `sysNotifyDisplayResizedEvent` notification by posting a `winDisplayChangedEvent` and then in this event handler, redraw the form in the available space. (See [“Resizing a Form”](#) on page 7.)

NOTE: Note that if an application does not call `FrmSetDIAPolicyAttr` before a form is drawn, the system assumes it is a legacy form that doesn’t support a closable input area. In this case, the input area is drawn without a trigger.

The Dynamic Input Area Feature

Before you can use any of the API described in this document, you must make sure that the dynamic input area API is available. Test the `pinFtrAPIVersion` feature as shown in [Listing 1.1](#)

Listing 1.1 Checking the dynamic input area feature

```
err = FtrGet(pinCreator, pinFtrAPIVersion, &version);  
if (!err && version) {  
    //PINS exists  
}
```

If this feature is defined, a new manager called the **Pen Input Manager** controls the input area and notifies the application of any changes in the input area state.

NOTE: Version 1.1 (`pinAPIVersion1_1`) of the Pen Input Manager is slightly different from version 1.0 (`pinAPIVersion1_0`). It doesn't matter which version you find; you should write your application to be compatible with both, as described in the following sections.

Size Constraints

After you've determined that the Pen Input Manager is available, you need to let it know the **size constraints** for each form in your application. You do so by calling the function [`WinSetConstraintsSize`](#) when the form is loaded; that is, in response to the `frmLoadEvent`. (Alternately, you can do this on a `frmOpenEvent`.)

The `WinSetConstraintsSize` function takes seven parameters: the handle to the form's window, the minimum, preferred, and maximum heights for the form, and the minimum, preferred, and maximum widths for the form. The Pen Input Manager uses this information to determine whether the form supports a closed input form.

If you don't specify size constraints, the Pen Input Manager assumes that the minimum, maximum, and preferred height and width are all 160 standard coordinates. In this way, legacy applications are always sized appropriately when the input area is open.

Applications and the Dynamic Input Area

Size Constraints

[Listing 1.2](#) shows how you might call `WinSetConstraintsSize` for an application with three forms: a main form, and edit form, and a password dialog. `WinSetConstraintsSize` must be called in response to the `frmLoadEvent` or `frmOpenEvent`. You typically handle the `frmLoadEvent` in the your application's `AppHandleEvent`, so you would place this code in your application's `AppHandleEvent` function.

Listing 1.2 Setting the size constraints

```
if (eventP->eType == frmLoadEvent) {
    // Load the form resource.
    formId = eventP->data.frmLoad.formID;
    frmP = FrmInitForm(formId);
    FrmSetActiveForm(frmP);

    // Set the same policy for each form in application.
    err = FrmSetDIAPolicyAttr(frmP, frmDIAPolicyCustom);

    // Enable the input trigger
    err = PINSetInputTriggerState(pinInputTriggerEnabled);

    formWinH = FrmGetWindowHandle (frmP);
    // Set the event handler for the form, and set each form's
    // size requirements.
    switch (formId) {
        case MainForm:
            FrmSetEventHandler(frmP, MainFormHandleEvent);
            WinSetConstraintsSize(formWinH, 80, 160, 225,
                                  160, 160, 160);
            break;

        case EditForm:
            FrmSetEventHandle(frmP, EditFormHandleEvent);
            WinSetConstraintsSize(formWinH, 100, 160, 225,
                                  160, 160, 160);
            break;

        case PasswordDialog:
            FrmSetEventHandler(frmP, PasswordDialogHandleEvent);
            FrmSetDIAPolicyAttr(frmPolicyStayOpen);
            break;

        default:
            break;
    }
}
```

```
        return true;  
    }
```

In [Listing 1.2](#), `WinSetConstraintsSize` is called once for each form in the application. There is currently nothing that should resize a form's width, so each of these calls uses 160 for all of the width parameters. The full-screen forms prefer to be 160 pixels high because the application uses these same forms when running on older devices that use static input areas; however, it's acceptable for the main form to be as short as 80 pixels and the Edit form to be as short as 100 pixels if the input area needs more space. Finally, the input area should not be closable for the password dialog because it is needed to enter the password.

Input Area Policy

To set up your application's forms to work with the Pen Input Manager, you must do the following for each form:

- Set an **input area policy** using the function [FrmSetDIAPolicyAttr](#) to let the system know that the form will resize itself when the input area is opened and closed. Do so in response to the `frmLoadEvent` or `frmOpenEvent`.
- Enable the **input trigger**, which is what the user uses to open and close the input area, using the function [PINSetInputTriggerState](#). An example of the trigger is the lower right arrow shown in [Figure 1.1](#) and [Figure 1.2](#) on page 1.

Setting the Input Area Policy

The input area policy that you set with [FrmSetDIAPolicyAttr](#) specifies whether your form supports a dynamic input area. [Listing 1.2](#) shows how you might call `FrmSetDIAPolicyAttr` in your application's `AppHandleEvent`.

Most forms should use a policy of `frmDIAPolicyCustom`. If not, the system treats your form as it does all forms in a legacy application. It opens the input area when the form is opened so that the form has the normal square area in which to draw, and it

Applications and the Dynamic Input Area

Input Area Policy

disables the input trigger so that the user cannot close the input area while your form is being displayed.

NOTE: Future Palm OS versions might not disable the input trigger for legacy applications or forms.

The input area policy is set on a form-by-form basis rather than an application-by-application basis because each form might have different requirements for the input area. A form that relies heavily on user input might need to ensure that the input area is opened while the form is active. However, a form that has no editable fields, like an address list, may want to keep the input area closed so that more information is visible.

Enabling the Input Trigger

For each form in your application that uses the policy `frmDIAPolicyCustom`, you can enable the input area's input trigger. Use the function [PINSetInputTriggerState](#) to enable the trigger (see [Listing 1.2](#)).

You must enable the input trigger for each form because a system dialog might disable it. System dialogs appear on top of your application's forms. When the dialog is dismissed and control returns to your application, the input trigger may be disabled. You won't receive a `frmLoadEvent` or `frmOpenEvent` because your form is already loaded and opened. Instead, you get the `winEnterEvent`. Therefore, to make sure your users can open and close the input area while your application is active, even after a system dialog is displayed, you must enable the trigger in response to the `winEnterEvent`.

Note that the key code associated with the input trigger button is `vchrInputAreaControl`, so a `keyDownEvent` with this character is enqueued whenever the trigger is tapped.

Setting an Input Area State

Most forms should not set a state for the input area. They should let the user decide whether the input area is open or closed.

In rare cases, it may be beneficial for the form to decide whether the input area is opened or closed. For example, you might have a tall form that closes the input area so that the user can see the entire form without scrolling.

You can set the input area state by using the function [`PINSetInputAreaState`](#). Specify the value `pinInputAreaUser` to set the state to the last user-selected state.

Be careful not to set the input area state too much. If the input area is opened and closed automatically in too many instances, the result may be a jumpy user interface that produces a jarring user experience. It is probably best to let your users decide what they want to do.

Resizing a Form

When the Pen Input Manager opens or closes the input area or control bar or changes the display orientation, it broadcasts the [`sysNotifyDisplayResizedEvent`](#) notification to all applications that have registered for it.

When the application receives the `sysNotifyDisplayResizedEvent`, it must post a `winDisplayChangedEvent` (using `EvtAddUniqueEventToQueue`), which forms can handle to resize themselves.

Listing 1.3 Posting a `winDisplayChangedEvent`

```
case sysAppLaunchCmdNotify:
    if (((SysNotifyParamType*) cmdPBP)->notifyType ==
        sysNotifyDisplayResizedEvent)
    {
        EventType resizedEvent;
        MemSet(&resizedEvent, sizeof(EventType), 0);
        //add winDisplayChangedEvent to the event queue
        resizedEvent.eType = winDisplayChangedEvent;
        EvtAddUniqueEventToQueue(&resizedEvent, 0, true);
    }
    break;
```

Applications and the Dynamic Input Area

Resizing a Form

It's possible for the input area to change state while a menu is open. To handle this case, an application should also enqueue a `winDisplayChangedEvent` (using code similar to [Listing 1.3](#)) when it receives a `winEnterEvent`.

The notification contains a rectangle specifying the new bounds for the current form. In general, forms should respond to the closing of the input area by moving any command buttons to the bottom of the new rectangle and by resizing the data area of the display.

[Listing 1.4](#) shows a simple example of handling the `winDisplayChangedEvent`.

Listing 1.4 Handling `winDisplayChangedEvent`

```
// Input area was opened or was closed. Must resize form.
case winDisplayChangedEvent:
    // get the current bounds for the form
    frmP = FrmGetActiveForm();
    WinGetBounds (FrmGetWindowHandle(frmP), &curBounds);

    // get the new display window bounds
    WinGetBounds(WinGetDisplayWindow(), &displayBounds);

    EditFormResizeForm(frmP, &curBounds, &displayBounds);
    FrmDrawForm(frmP);
    handled = true;
    break;
```

This example is for a form containing one multi-line text field, a scroll bar, and several command buttons at the bottom of the form. It responds to the `winDisplayChangedEvent` by passing the form's current bounds and new bounds to a function named `EditFormResizeForm`, which is shown in [Listing 1.5](#). This function determines the difference between the current height and width and the new height and width and then applies that difference to the objects in the form. For the command buttons, it changes their position so that they always appear at the bottom of the form. For the text field and scroll bar, it resizes them so that the more text lines are displayed on the screen.

Note that after resizing a form with a text field, it is a good idea to call `FldRecalculateField` to update the word-wrapping for the field's new size.

Listing 1.5 Resizing a form

```
void EditFormResizeForm(FormType *frmP,
RectangleType* fromBoundsP, RectangleType* toBoundsP)
{
    Int16 heightDelta, widthDelta;
    UInt16 numObjects, i;
    Coord x, y;
    RectangleType objBounds;

    heightDelta = widthDelta = 0;
    numObjects = 0;
    x = y = 0;
    FieldType* fldP;

    // Determine the amount of the change
    heightDelta=(toBoundsP->extent.y - toBoundsP->topLeft.y) -
        (fromBoundsP->extent.y - fromBoundsP->topLeft.y);
    widthDelta=(toBoundsP->extent.x - toBoundsP->topLeft.x) -
        (fromBoundsP->extent.x - fromBoundsP->topLeft.x);

    // Iterate through objects and re-position them.
    // This form consists of a big text field and
    // command buttons. We move the command buttons to the
    // bottom and resize the text field to display more data.
    numObjects = FrmGetNumberOfObjects(frmP);
    for (i = 0; i < numObjects; i++) {
        switch (FrmGetObjectTypes(frmP, i)) {
            case frmControlObj:
                FrmGetObjectPosition(frmP, i, &x, &y);
                FrmSetObjectPosition(frmP, i, x + widthDelta, y +
                    heightDelta);
                break;
            case frmFieldObj:
            case frmScrollBarObj:
                FrmGetObjectBounds(frmP, i, &objBounds);
                objBounds.extent.x += widthDelta;
                objBounds.extent.y += heightDelta;
                FrmSetObjectBounds(frmP, i, &objBounds);
                fldP = (FieldType*) FrmGetObjectPtr(frmP, i);
                FldRecalculateField(fldP, false);
                break;
        }
    }
}
```

Hiding and Showing the Control Bar

There are two functions that applications can use to hide and show the control bar: [StatHide](#) and [StatShow](#).

It's best not to manually hide or show the control bar, but there may be some situations in which an application needs to draw to the entire display area and thus must hide the control bar. However, if the control bar is hidden, this prevents users from exiting to the Launcher or opening the input area. If the control bar is hidden, you must provide a mechanism for the user to exit the application or to show the control bar.

To determine if the control bar is hidden or showing, you can call [StatGetAttribute](#) with the `statAttrBarVisible` selector. You can obtain the bounds of the control bar by using the `statAttrDimension` selector.

Note that the `Stat...` functions are available only in Pen Input Manager version 1.1.

Pen Input Manager Compatibility

Pen Input Manager version 1.1 was introduced with Palm OS version 5.3SC. There are some differences between Pen Input Manager version 1.1 and version 1.0.

The version of the Pen Input Manager is returned in the `version` parameter of the following `FtrGet` call:

```
err = FtrGet(pinCreator, pinFtrAPIVersion, &version);
```

We recommend writing your application to be compatible with all versions of the Pen Input Manager. It will be compatible if you follow the guidelines in this book.

This section documents the differences in version 1.1 so that you know the details. They include:

- [New sysFtrNumInputAreaFlags Support](#)
- [Additional winDisplayChangedEvent](#)
- [Restoration of Input Trigger State](#)

- [New pinInputAreaUser Input Area State](#)
- [New Stat... Functions](#)
- [New Support for Changing Display Orientation](#)

New sysFtrNumInputAreaFlags Support

The presence of the Pen Input Manager in version 1.1 does not indicate the capabilities of the device. In version 1.0, the device supports all of the following features, and the flags are not implemented.

In version 1.1, you must test another feature, `sysFtrNumInputAreaFlags`, to determine if the device supports a dynamic input area, live ink, and a closable dynamic input area.

```
err = FtrGet(sysFtrCreator, sysFtrNumInputAreaFlags, &flags)
```

A selector is available to determine if the OS supports the dynamic input area. If the `grfFtrInputAreaFlagDynamic` flag is set to 0, or `FtrGet` returns an error, then the dynamic input area is not supported. Likewise, if the `grfFtrInputAreaFlagCollapsible` flag is set to 0, or if `FtrGet` returns an error, then a closable dynamic input area is not supported.

The `flags` argument is initialized using bits defined in `Graffiti.h`:

```
#define grfFtrInputAreaFlagDynamic 0x00000001  
#define grfFtrInputAreaFlagLiveInk 0x00000002  
#define grfFtrInputAreaFlagCollapsible 0x00000004
```

Additional winDisplayChangedEvent

Version 1.1 of the Pen Input Manager uses an additional mechanism for notifying applications that the input area or control bar has opened or closed. Version 1.0 sends the `sysNotifyDisplayResizedEvent` notification, while version 1.1 sends this notification and also the `winDisplayChangedEvent`.

To be compatible with both versions, an application should post a `winDisplayChangedEvent` (using

`EvtAddUniqueEventToQueue`) when the `sysNotifyDisplayResizedEvent` notification is received.

[Listing 1.3](#) shows how to do this.

Restoration of Input Trigger State

With Pen Input Manager version 1.1, if you set an input area policy of `frmDIAPolicyCustom` but don't call `PINSetInputTriggerState` to enable or disable the trigger or call `PINSetInputAreaState` to open or close the input area, then the system automatically restores the last user-selected input area state and enables the trigger (1.0 doesn't do this). If `PINSetInputAreaState` and/or `PINSetInputTriggerState` is called by the application, however, then the form's resulting state is restored when the form is updated due to a call to `FrmDrawForm` or a `WinEnterEvent`.

With Pen Input Manager version 1.0, you must enable the input trigger for each form because a system dialog might disable it. System dialogs appear on top of your application's forms. System dialogs use the same input area policy as legacy applications do: the input area is opened and the user is not allowed to close it. When the dialog is dismissed and control returns to your application, the input trigger will still be disabled. You won't receive a `frmLoadEvent` or `frmOpenEvent` because your form is already loaded and opened. Instead, you get the `winEnterEvent`. Therefore, to make sure your users can open and close the input area while your application is active, even after a system dialog is displayed, you must enable the trigger in response to the `winEnterEvent`.

New `pinInputAreaUser` Input Area State

Pen Input Manager version 1.1 implements a new input area state: `pinInputAreaUser`. However, you can use this state in applications designed to run in both 1.0 and 1.1 environments, because it will simply be ignored in 1.0.

New Stat... Functions

The following new functions are implemented in Pen Input Manager version 1.1, but not in version 1.0:

[StatGetAttribute](#), [StatHide](#), and [StatShow](#)

It's best not to use these functions in order to be compatible with devices running Pen Input Manager version 1.0.

New Support for Changing Display Orientation

Pen Input Manager version 1.1 implements support for changing the display orientation between portrait, landscape, and the reverse of each. This allows the display to be rotated to any of the four possible directions.

The following functions support the display orientation feature:

[SysGetOrientation](#), [SysSetOrientation](#),
[SysGetOrientationTriggerState](#),
[SysSetOrientationTriggerState](#).

Not all devices support changing the display orientation. For devices that don't support changing the display orientation, the only valid orientation is portrait.

NOTE: Orientation support is implemented only in Pen Input Manager version 1.1 in Palm OS version 5.3. Pen Input Manager version 1.1 is available on earlier OS versions, but depending on licensee support, may or may not include this feature. To check if this function is implemented in Pen Input Manager 1.1 in a Palm OS version earlier than 5.3, you must use `SysGlueTrapExists`.

Applications and the Dynamic Input Area

Pen Input Manager Compatibility

Pen Input Manager Reference

This chapter provides reference material for the Pen Input Manager API as declared in the header file `PenInputMgr.h`. It discusses the following topics:

- [Pen Input Manager Constants](#)
- [Pen Input Manager Functions](#)
- [Other Functions](#)

Pen Input Manager Constants

sysNotifyDisplayResizedEvent

The `sysNotifyDisplayResizedEvent` notification is broadcast by [PINSetInputAreaState](#) after the dynamic input area or control bar has been opened or closed. Normally, the user opens and closes the dynamic input area and control bar, but applications may also do so, though this is not encouraged.

Applications may respond to the notification by redrawing the active form in the available space.

```
#define sysNotifyDisplayResizedEvent 'scrs'
```

sysNotifyDisplayResizedEvent Specific Data

`notifyDetailsP` points to a `SysNotifyDisplayResizedDetailsType` structure.

Pen Input Manager Reference

Pen Input Manager Constants

Prototype `typedef struct SysNotifyDisplayResizedDetailsTag
{
 RectangleType newBounds;
} SysNotifyDisplayResizedDetailsType;`

Fields `newBounds` The new bounds of the application area after the input area or control bar has been opened or closed. The application should draw the current form within these bounds.

winDisplayChangedEvent

In Pen Input Manager version 1.1, the event `winDisplayChangedEvent` is posted by [PINSetInputAreaState](#) after the dynamic input area has been opened or closed. Normally, the user opens and closes the dynamic input area, but applications may also do so.

Applications may respond to the event by redrawing the active form in the available space.

By the time the application receives this event, the OS has already changed the bounds of the display window as appropriate to the state of the input area. The application must resize its active form's window and relay layout the form accordingly.

Input Area States

[Table 2.1](#) lists constants that define the states that the input area can have. An application can obtain the input area's current state with [PINGetInputAreaState](#) and set it with [PINSetInputAreaState](#).

Table 2.1 Input area states

Constant	Value	Description
<code>pinInputAreaOpen</code>	0	The dynamic input area is being displayed.
<code>pinInputAreaClosed</code>	1	The dynamic input area is not being displayed.

Table 2.1 Input area states (continued)

Constant	Value	Description
		The dynamic input area is in this state after the user taps the input trigger to close it. An application also might request that the dynamic input area be closed by calling PINSetInputAreaState with this state.
<code>pinInputAreaNone</code>	2	The input area is not dynamic, or there is no input area. Do not pass this value to <code>PINSetInputAreaState</code> .
<code>pinInputAreaUser</code>	5	Pass this value to <code>PINSetInputAreaState</code> to tell the Pen Input Manager to activate the last user-selected input area state.

Input Trigger States

[Table 2.2](#) lists constants that specify the state of the input area icon in the status bar. An application can obtain this state with [PINGetInputTriggerState](#) and set it with [PINSetInputTriggerState](#).

Table 2.2 Input trigger states

Constant	Value	Description
<code>pinInputTriggerEnabled</code>	0	The input trigger is enabled, meaning that the user is allowed to open and close the dynamic input area.
<code>pinInputTriggerDisabled</code>	1	The input trigger is disabled, meaning that the user is not allowed to close the dynamic input area.
<code>pinInputTriggerNone</code>	2	There is no dynamic input area.

Form Dynamic Input Area Policies

A dynamic input area policy specifies how the dynamic input area should be handled while a form is active. These values are used for

Pen Input Manager Reference

Pen Input Manager Constants

the `diaPolicy` attribute in a form's attribute structure. You can set the value with [FrmSetDIAPolicyAttr](#) and retrieve it with [FrmGetDIAPolicyAttr](#).

Table 2.3 Form dynamic input area policy constants

Constant	Value	Description
<code>frmDIAPolicyStayOpen</code>	0	Forces the dynamic input area to stay open while the form is active. The input trigger is disabled.
<code>frmDIAPolicyCustom</code>	1	The user and the application control whether the input area is active.

Orientation States

[Table 2.4](#) lists constants that specify the display orientation. An application can obtain this state with [SysGetOrientation](#) and set it with [SysSetOrientation](#).

Table 2.4 Orientation state constants

Constant	Value	Description
<code>sysOrientationUser</code>	0	Pass this value to <code>SysSetOrientation</code> to tell the system to activate the last user-selected orientation.
<code>sysOrientationPortrait</code>	1	The display is in portrait orientation.
<code>sysOrientationLandscape</code>	2	The display is in landscape orientation.

Table 2.4 Orientation state constants

Constant	Value	Description
<code>sysOrientationReversePortrait</code>	3	The display is in reverse portrait orientation (upside-down from the normal portrait orientation).
<code>sysOrientationReverseLandscape</code>	4	The display is in reverse landscape orientation (upside-down from the normal landscape orientation).

Orientation Trigger States

[Table 2.5](#) lists constants that specify the state of the orientation icon in the status bar (the icon that allows the user to change the display orientation). An application can obtain this state with [SysGetOrientationTriggerState](#) and set it with [SysSetOrientationTriggerState](#).

Table 2.5 Orientation trigger state constants

Constant	Value	Description
<code>sysOrientationTriggerDisabled</code>	0	The orientation trigger is disabled, meaning that the user is not allowed to change the display orientation.
<code>sysOrientationTriggerEnabled</code>	1	The orientation trigger is enabled, meaning that the user is allowed to change the display orientation.

Pen Input Manager Functions

PINGetInputAreaState

Purpose Returns the current state of the dynamic input area.

Prototype `UInt16 PINGetInputAreaState (void)`

Parameters None.

Result One of the constants defined in the section “[Input Area States](#)” on page 16.

See Also [PINSetInputAreaState](#), [PINGetInputTriggerState](#)

PINGetInputTriggerState

Purpose Returns the status of the input area icon in the status bar.

Prototype `UInt16 PINGetInputTriggerState (void)`

Parameters None.

Result One of the constants defined in the section “[Input Trigger States](#)” on page 17.

See Also [PINGetInputAreaState](#)

PINSetInputAreaState

Purpose	Sets the state of the input area.								
Prototype	<code>Err PINSetInputAreaState (UInt16 state)</code>								
Parameters	<table><tr><td>-> state</td><td>The state to which the input area should be set. See "Input Area States" on page 16 for a list of possible values.</td></tr></table>	-> state	The state to which the input area should be set. See " Input Area States " on page 16 for a list of possible values.						
-> state	The state to which the input area should be set. See " Input Area States " on page 16 for a list of possible values.								
Result	<table><tr><td colspan="2">Returns one of the following error codes:</td></tr><tr><td>errNone</td><td>Success.</td></tr><tr><td>pinErrNoSoftInputArea</td><td>There is no dynamic input area on this device.</td></tr><tr><td>pinErrInvalidParam</td><td>You have entered an invalid state parameter.</td></tr></table>	Returns one of the following error codes:		errNone	Success.	pinErrNoSoftInputArea	There is no dynamic input area on this device.	pinErrInvalidParam	You have entered an invalid state parameter.
Returns one of the following error codes:									
errNone	Success.								
pinErrNoSoftInputArea	There is no dynamic input area on this device.								
pinErrInvalidParam	You have entered an invalid state parameter.								
Comments	After opening or closing the input area, this function broadcasts the notification sysNotifyDisplayResizedEvent (and in Pen Input Manager version 1.1, posts the event winDisplayChangedEvent to the event queue). Applications register for this notification or respond to the event if they wish to resize themselves.								
See Also	PINGetInputAreaState , PINSetInputTriggerState , " Setting an Input Area State " on page 6								

PINSetInputTriggerState

- Purpose** Sets the state of the input area icon in the status bar.
- Prototype** `Err PINSetInputTriggerState (UInt16 state)`
- Parameters** `-> state` The state to which the input trigger should be set. See “[Input Trigger States](#)” on page 17 for a list of possible values.
- Result** Returns one of the following error codes:
- | | |
|------------------------------------|--|
| <code>errNone</code> | Success. |
| <code>pinErrNoSoftInputArea</code> | There is no dynamic input area on this device. |
| <code>pinErrInvalidParam</code> | You have specified an invalid state parameter. |
- Comments** Applications or Palm OS call this function to enable the input area icon in the status bar. Normally, this trigger is enabled and should remain enabled, allowing the user the choice of displaying the input area or not. Legacy applications might disable the trigger on some devices.
- See Also** [PINGetInputTriggerState](#), [PINSetInputAreaState](#), “[Enabling the Input Trigger](#)” on page 6

Other Functions

This section lists other functions that are implemented as part of the Pen Input Manager in Palm OS 5.

FrmGetDIAPolicyAttr

Purpose	Returns a form's dynamic input area policy.		
Prototype	<code>UInt16 FrmGetDIAPolicyAttr (FormPtr formP)</code>		
Parameters	<table><tr><td>-> formP</td><td>A pointer to a FormType structure.</td></tr></table>	-> formP	A pointer to a FormType structure.
-> formP	A pointer to a FormType structure.		
Result	Returns one of the constants listed in " Form Dynamic Input Area Policies " on page 17.		
See Also	FrmSetDIAPolicyAttr		

FrmSetDIAPolicyAttr

Purpose	Sets a form's dynamic input area policy.				
Prototype	<code>Err FrmSetDIAPolicyAttr (FormPtr formP, UInt16 diaPolicy)</code>				
Parameters	<table><tr><td>-> formP</td><td>A pointer to a FormType structure.</td></tr><tr><td>-> diaPolicy</td><td>One of the constants listed in "Form Dynamic Input Area Policies" on page 17.</td></tr></table>	-> formP	A pointer to a FormType structure.	-> diaPolicy	One of the constants listed in " Form Dynamic Input Area Policies " on page 17.
-> formP	A pointer to a FormType structure.				
-> diaPolicy	One of the constants listed in " Form Dynamic Input Area Policies " on page 17.				
Result	Returns <code>errNone</code> if no error or <code>pinErrInvalidParam</code> if the <code>diaPolicy</code> parameter is out of range.				
Comments	Applications call this function in response to the <code>frmLoadEvent</code> or <code>frmOpenEvent</code> , to set the policy that the form should use for opening and closing the dynamic input area. Note that if an application does not call this function, the default is <code>frmDIAPolicyStayOpen</code> . This allows legacy application to always be sized appropriately because the input area is always open, with the trigger disabled, while the legacy application is running.				
See Also	PINSetInputAreaState , FrmGetDIAPolicyAttr , " Setting the Input Area Policy " on page 5				

StatGetAttribute

Purpose Returns the control bar state.

Prototype `Err StatGetAttribute (UInt16 selector,
UInt32* dataP)`

Parameters

-> selector	Attribute selector, as described in the Comments section below.
-> dataP	Pointer to the returned data, as described in the Comments section below.

Result Returns one of the following error codes:

`errNone` Success.

`sysErrParamErr` You have specified an invalid selector parameter.

Comments The following values are supported for the selector parameter:

`statAttrBarVisible`

Checks if the control bar is visible. The return data is set to 0 if the control bar is hidden, or is set to 1 if the control bar is visible or the input area is open.

`statAttrDimension`

Gets the control bar bounds. The return data is two `UInt16` values, where the first is the width of the control bar and the second is the height. The dimensions use the active coordinate system.

Compatibility Implemented only in Pen Input Manager version 1.1.

See Also [StatHide](#)

StatHide

Purpose	Hides the control bar.						
Prototype	<code>Err StatHide (void)</code>						
Result	Returns one of the following error codes: <table><tr><td><code>errNone</code></td><td>Success.</td></tr><tr><td><code>sysErrNotAllowed</code></td><td>The device does not support a dynamic input area.</td></tr><tr><td><code>sysErrInputWindowOpen</code></td><td>The input area is open (so the control bar is not currently visible).</td></tr></table>	<code>errNone</code>	Success.	<code>sysErrNotAllowed</code>	The device does not support a dynamic input area.	<code>sysErrInputWindowOpen</code>	The input area is open (so the control bar is not currently visible).
<code>errNone</code>	Success.						
<code>sysErrNotAllowed</code>	The device does not support a dynamic input area.						
<code>sysErrInputWindowOpen</code>	The input area is open (so the control bar is not currently visible).						
Comments	<p>The input area must be closed before you call this function.</p> <p>This function can be called by applications that want to draw to the entire display area and thus need to hide the control bar. However, hiding the control bar is discouraged since it prevents users from exiting to the Launcher or opening the input area via buttons that appear on the control bar. If the control bar is hidden, you must provide a mechanism for the user to exit the application.</p>						
Compatibility	Implemented only in Pen Input Manager version 1.1.						
See Also	StatGetAttribute , StatShow						

StatShow

Purpose	Shows the control bar.		
Prototype	<code>Err StatShow (void)</code>		
Result	Returns one of the following error codes: <table><tr><td><code>errNone</code></td><td>Success.</td></tr></table>	<code>errNone</code>	Success.
<code>errNone</code>	Success.		

Pen Input Manager Reference

Other Functions

`sysErrNotAllowed`

The device does not support a dynamic input area.

Comments If the input area is open when this function is called, it has no effect and `errNone` is returned.

Compatibility Implemented only in Pen Input Manager version 1.1.

See Also [StatGetAttribute](#), [StatHide](#)

SysGetOrientation

Purpose Returns the display orientation.

Prototype `UInt16 SysGetOrientation (void)`

Result Returns one of the constants listed in “[Orientation States](#)” on page 18.

Comments Not all devices support changing the display orientation. For devices that don’t support changing the display orientation, this function always returns `sysOrientationPortrait`.

Compatibility Implemented only in Pen Input Manager version 1.1 in Palm OS version 5.3. Some licensees may have implemented this function in Pen Input Manager version 1.1 in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if (SysGlueTrapExists(pinSysGetOrientation)) {  
    // SysGetOrientation exists  
}
```

See Also [SysSetOrientation](#)

SysSetOrientation

Purpose	Sets the display orientation.
Prototype	<code>Err SysSetOrientation (UInt16 orientation)</code>
Parameters	<code>-> orientation</code> The orientation to which the display should be set. See “ Orientation States ” on page 18 for a list of possible values.
Result	Returns one of the following error codes: <code>errNone</code> Success. <code>sysErrNotAllowed</code> Setting the display orientation is not supported on the device.
Comments	Not all devices support changing the display orientation.
Compatibility	Implemented only in Pen Input Manager version 1.1 in Palm OS version 5.3. Some licensees may have implemented this function in Pen Input Manager version 1.1 in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test: <pre>if (SysGlueTrapExists(pinSysSetOrientation)) { // SysSetOrientation exists }</pre>
See Also	SysGetOrientation

SysGetOrientationTriggerState

Purpose	Returns the display orientation trigger state.
Prototype	<code>UInt16 SysGetOrientationTriggerState (void)</code>
Result	Returns one of the constants listed in “ Orientation Trigger States ” on page 19.

Pen Input Manager Reference

Other Functions

Comments Not all devices support changing the display orientation. For devices that don't support changing the display orientation, this function always returns `sysOrientationTriggerDisabled`.

Compatibility Implemented only in Pen Input Manager version 1.1 in Palm OS version 5.3. Some licensees may have implemented this function in Pen Input Manager version 1.1 in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if
(SysGlueTrapExists (pinSysGetOrientationTriggerState)) {
// SysGetOrientationTriggerState exists
}
```

See Also [SysSetOrientationTriggerState](#)

SysSetOrientationTriggerState

Purpose Sets the display orientation trigger state.

Prototype `Err SysSetOrientationTriggerState (UInt16 triggerState)`

Parameters `-> triggerState` One of the constants listed in "[Orientation Trigger States](#)" on page 19.

Result Returns one of the following error codes:

<code>errNone</code>	Success.
<code>sysErrNotAllowed</code>	Setting the display orientation is not supported on the device.

Comments Not all devices support changing the display orientation.

Compatibility Implemented only in Pen Input Manager version 1.1 in Palm OS version 5.3. Some licensees may have implemented this function in

Pen Input Manager version 1.1 in an earlier OS version. To check if this function is implemented in an earlier OS version, use this test:

```
if
(SysGlueTrapExists(pinSysSetOrientationTriggerState)) {
// SysSetOrientationTriggerState exists
}
```

See Also [SysGetOrientationTriggerState](#)

WinSetConstraintsSize

Purpose Sets the maximum, preferred, and minimum size constraints for a window.

Prototype `Err WinSetConstraintsSize (WinHandle winHandle, Coord minH, Coord prefH, Coord maxH, Coord minW, Coord prefW, Coord maxW)`

Parameters	-> winHandle	A handle to a window.
	-> minH	The minimum height to which this window can be sized in standard coordinates. This value must be less than or equal to 160 pixels.
	-> prefH	The preferred height for this window in standard coordinates.
	-> maxH	The maximum height for this window in standard coordinates.
	-> minW	The minimum width for the window in standard coordinates.
	-> prefW	The preferred width for the window in standard coordinates.
	-> maxW	The maximum width for the window in standard coordinates.

Result Returns one of the following error codes:

errNone	Success.
---------	----------

Pen Input Manager Reference

Other Functions

`pinErrNoSoftInputArea`

There is no dynamic input area on this device.

`pinErrInvalidParam`

You have specified an invalid parameter.

Comments Applications must call this function in response to a `frmLoadEvent` or `frmOpenEvent` to set the size constraints for the window. If the application does not call this function, it is assumed that both the minimum and maximum values for the window are 160 pixels by 160 pixels.

See Also [“Size Constraints”](#) on page 3

Index

A

AppHandleEvent 4

D

dynamic input area v

F

frmDIAPolicyCustom 5, 6, 18
frmDIAPolicyStayOpen 18, 23
FrmGetDIAPolicyAttr 18, 23
frmLoadEvent 4, 5, 6, 12, 23, 30
frmOpenEvent 3, 4, 5, 6, 12, 23, 30
FrmSetDIAPolicyAttr 5, 18, 23

I

input area state 16, 20, 21, 22

P

Pen Input Manager 15
PenInputMgr.h 15
pinErrInvalidParam 22, 23, 30
pinErrNoSoftInputArea 22, 30
pinFtrAPIVersion 2
PINGetInputAreaState 16, 20
PINGetInputTriggerState 17, 20
pinInputAreaClosed 16
pinInputAreaNone 17
pinInputAreaOpen 16
pinInputAreaUser 17
pinInputTriggerDisabled 17
pinInputTriggerEnabled 17
pinInputTriggerNone 17
PINSetInputAreaState 7, 15, 16, 17, 21
PINSetInputTriggerState 5, 6, 17, 22

S

StatGetAttribute 24
StatHide 25
StatShow 25
SysGetOrientation 18, 26
SysGetOrientationTriggerState 19, 27
sysNotifyDisplayResizedEvent 7, 15, 21

sysOrientationLandscape 18
sysOrientationPortrait 18
sysOrientationReverseLandscape 19
sysOrientationReversePortrait 19
sysOrientationTriggerDisabled 19
sysOrientationTriggerEnabled 19
sysOrientationUser 18
SysSetOrientation 18, 27
SysSetOrientationTriggerState 19, 28

W

winDisplayChangedEvent 7, 8, 16, 21
winEnterEvent 6, 12
WinSetConstraintsSize 3, 4, 29

