Palm, Inc.

Using Five Way Navigator

SDK Documentation

Table of Contents

I. OVERVIEW

A. Description

Palm handhelds will use a new 5-way navigator to replace the current up and down hard buttons. The 5-way navigator can perform 5 different actions, conceptually left, right, up, down, and select. It provides flexible and powerful navigation control similar to that of regular mice and joysticks. It will make Palm handhelds more convenient for single hand operation. It will also make Palm handhelds more attractive for new applications, such as games and other GUI rich applications. Existing PalmOS applications can be enhanced to take advantage of the functionality to improve end user experience and provide new features.

B. Design Objectives

1. Compatibility

The new 5-way navigator should generate the same events for up and down operations, as the current up and down hard buttons would do. All existing applications that use these events will behave exactly the same way when they are used with the up and down buttons of the 5-way navigator, and they should not respond to the left/right/select buttons.

2. Consistency

Since some existing applications respond to the up/down hard buttons in different ways, the inconsistency has already become an issue. The 5-way navigator will not try to solve the existing problems. What we try to do is to make sure that all new applications handle the 5-way navigator in an intuitive and consistent way. For example, the up/down/left/right buttons should be consistently used to navigate through records, items or fields on a form, the select button should issue a command based on the current selection. In general, the 5-way navigator should be used to control navigation cross fields and records, but not on the input caret inside fields.

II. SPECIFICATIONS

A. 5-Way Description

The 5-Way navigator is an input device that can generate 5 different actions, conceptually left, right, up, down, and select. When any of the actions is performed – press or release of any 5-way buttons – the system will post a key down event into the system event queue. The event will carry the state and transition information regarding the 5-way navigator. The chr field of keyDown member of the event contains the virtual character for this event, and the corresponding keyCode field indicates the state and transition of buttons on the 5-way navigator.

The chr field is set to vchrPageUp if the up button is pressed, or vchrPageDown if the down button is pressed. Otherwise, it is set to vchrNavChange. In this way, the 5way navigator can be fully compatible with existing applications, since they only check the chr field. For applications that are aware of the 5way navigator, they must check the keyCode field for additional information (see below).

The keyCode field is 10-bit mask, including 5 state bits and 5 transition bits. The 5 state bits indicate the state of the 5-way navigator buttons at the time event happens. They can be tested against constants navBitUp, navBitDown, navBitLeft, navBitRight, and navBitSelect. If a button is pressed, its corresponding state bit is set. Otherwise, the bit is clear. The 5 transition bits indicate whether the 5-way navigator buttons have been pressed or released since the last 5-way navigator event. They can be tested against constants navChangeUp,

navChangeDown, navChangeLeft, navChangeRight, and navChangeSelect. If a button has been either pressed or released since last navigator event, its corresponding transition bit is set. Otherwise, it is clear.

The buttons on the 5-way navigator may or may not be independent. The independent 5-way navigator has 5 independent buttons or keys, which can be pressed and released in any combination. Non-independent 5-way navigator has physically connected or constrained button(s), which are not allowed to be pressed or released in some combination, much like the up/down hard button on old Palm handheld devices. Developers should not assume the independence of buttons on the 5-way navigator.

The 5way navigator also introduces 3 key bits, which are keyBitNavLeft, keyBitNavRight, keyBitNavSelect. They can be used with the Key Manager API, such as KeyCurrentState(), and KeySetMask(). Applications can use Key Manager APIs to control the 5way navigator buttons just like other hard buttons.

The following are some examples. Since all 5-way events are PalmOS key down events, only the keyDown member of the event is shown here. The event information is given as {chr, keyCode, modifiers} tuple, which are field of keyDown member of PalmOS event structure.

For simple click of the select key, application will see the following two events:

```
{vchrNavChange, navBitSelect | navChangeSelect, commandKeyMask}
{vchrNavChange, navChangeSelect, commandKeyMask}
```

For simple click of the up key, application will see the following two events:

```
{vchrPageUp, navBitUp | navChangeUp, commandKeyMask}
{vchrNavChange, navChangeUp, commandKeyMask}
```

For press, hold, and release of the select key, application will see the following events:

```
{vchrNavChange, navBitSelect | navChangeSelect, commandKeyMask}
{vchrNavChange, navBitSelect, commandKeyMask | autoRepeatKeyMask}
…
{vchrNavChange, navChangeSelect, commandKeyMask}
```

For rotating the 5-way controller starting from left, application will see the following events:

```
{vchrNavChange, navBitLeft | navChangeLeft,
 commandKeyMask}
{vchrPageUp, navBitUp | navChangeLeft | navChangeUp,
 commandKeyMask | autoRepeatKeyMask}
{vchrNavChange, navBitRight | navChangeUp | navChangeRight,
 commandKeyMask | autoRepeatKeyMask }
{vchrPageDown, navBitDown | navChangeRight | navChangeDown,
 commandKeyMask | autoRepeatKeyMask }
{vchrNavChange, navBitLeft | navChangeDown | navChangeLeft,
 commandKeyMask | autoRepeatKeyMask }
…
{vchrNavChange, navChangeLeft, commandKeyMask}
```

B. 5-Way Feature Set

Since the 5-way navigator is an optional input device, the Palm applications must be able to query the existence of the 5-way navigator. We plan to use the feature manager to provide this information.

Applications can use the following API call to determine the existence of the 5-way navigator and the version of its software implementation.

```
UInt32 version;
Err err = FtrGet(navFtrCreator, navFtrVersion, &version);
```

The existence of feature indicates the existence of the 5-way navigator and corresponding API implementation. The version number is used for future compatibility. For Rev 1.xx of this document, the version number is 0x00010000.

C. Using Macros

Some macros are defined and should the one used by the application to handle Left/Up/Select/Right/Down.

```
// A macro to use for apps that support navigation using the 5-way.
// By using this macro, we have consistent behavior in all our apps
// when it comes to how it handles multiple simultaneous button presses.
// You can use this macro even if the device does not have a 5-way controller.
//
// Usage:      if (NavKeyPressed(eventP, Select))
//                     if (NavKeyPressed(eventP, Left))
//                     if (NavKeyPressed(eventP, Right))
//                     if (NavKeyPressed(eventP, Up))               - also works without 5-way
//                     if (NavKeyPressed(eventP, Down))             - also works without 5-way
//
```

Select must be handled by the application on Release (to enable Long press on Select – see below).
Directions are handled on Press.

Using the macros will get the right behavior.

D. Default behavior

Palm handheld have some predefined actions that applications must usually not prevent. Some specific behavior might require it though

1. Long press on select

A long press on Select with Start the launcher. To prevent this, an application can handle the first keyPress of select in its form handler and return true.

Example to override the default behavior:

```
Boolean MyFormHandleEvent(EventType* evtP)
{
   if ((evtP->eType == keyDownEvent) &&
      (EvtKeydownIsVirtual(evtP)) &&
      (evtP->data.keyDown.chr == vchrNavChange) &&
      ((evtP->data.keyDown.keyCode & navChangeBitsAll) == navChangeSelect) &&
      ((evtP->data.keyDown.keyCode & navBitsAll) == navBitSelect))
   {
      // The patch won't get this event and long press is now disabled
      return true;
   }
   return false;
}
```

2. Navigation between dialog

If an application does not handle Select navigation, the system will look for Buttons within modals dialog (including Alert).

- if one button, the button is pressed and the dialog exited
- if more than one it look for affirmative text (OK, Done) or negative (No, Cancel) if Select was pressed.

Applications can override this behavior using the same macro and returning true in their form handler.

Example to override the default behavior:

```
Boolean MyFormHandleEvent(EventType* evtP)
{
   if ((evtP->eType == keyDownEvent) &&
      (EvtKeydownIsVirtual(evtP)) &&
      (NavKeyPressed(evtP, Select))
   {
      // The patch won't get this event and automatic handling in modal dialg
      // is disabled
      return true;
   }
   return false;
}
```