Creating a Front-End Processor

Exploring Palm OS®

Written by Christopher Bey
Technical assistance from Vivek Magotra

Exploring Palm OS: Creating a Front-End Processor
Document Number 3118-001
November 9, 2004
For the latest version of this document, visit
http://www.palmos.com/dev/support/docs/.

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

Table of Contents

About This Document **vii**

Intended Audience vii
FEP Developers vii
Other Developers vii
Requirements viii
What this Book Contains. viii
The *Exploring Palm OS* Series ix
Additional Resources x

1 Basic Concepts **1**

What Is a Front-End Processor? 1
How Does a User Input Text?. 2
How Is Inline Input Processed? 2
How Does the FEP Handle Conversion? 2
What Is a FEP in the Palm OS? 3
For More Information 3

2 The FEP User Interface **5**

Input Area Buttons 5
Standard FEP Buttons. 5
The Change Mode Button 6
Interactions with Forms and Fields 7
The Sample FEP User Interface 7
A Simplified Chinese FEP User Interface. 9
A Japanese FEP User Interface 10
Edit Menu Items 13
The FEP Panel 14

3 Creating a FEP Shared Library **17**

The Sample FEP 17
Sample FEP File List 17
The TestSampleFep Application 19
FEP Code Structure. 20
Text Services Manager Server. 20

Event Flow in a FEP 21
Initialization Sequence 21
System Events 22
Field-Level Events 22
Notes About Event Handling 23
FEP Type and Creator ID 23
Modifying the Sample FEP 23
Changing the Locale 24
Handling Text Services Manager Button Events 24
Handling Other Events 24
Handling the Mode Indicator 25
Handling Auto-Yomi Events (Japanese only) 25
Auto-Extending the Maximum Size of a Field 25
Adding User Dictionary Functions 26
Debugging and Testing the FEP 26

4 Text Services Manager Reference 29

Text Services Manager Constants 29
Feature Constants 29
TsmFepModeType 29
Text Services Manager Functions and Macros 30
TsmGetFepMode 30
TsmSetFepMode 31

5 Text Services FEP Reference 33

FEP Events 33
tsmConfirmEvent 33
tsmFepButtonEvent 34
tsmFepModeEvent 35
tsmFepChangeEvent 35
tsmFepDisplayOptionsEvent 36
tsmFepSelectOptionEvent 37
Text Services FEP Structures and Types 37
FepPanelAddWordParamsType 37
TsmFepActionType 37
TsmFepEventType 39
TsmFepInfoType 41

TsmFepStatusType 42
Text Services FEP Constants 44
Button ID Constants 44
Error Codes 45
Miscellaneous Constants 46
Text Services FEP Launch Codes 46
sysAppLaunchCmdFepPanelAddWord 46
Text Services FEP Functions 47
TsmFepCommitAction 47
TsmFepHandleEvent 47
TsmFepMapEvent 48
TsmFepOptionsList 49
TsmFepReset 49
TsmFepTerminate 50
TsmGetCurrentFepCreator 50
TsmGetSystemFepCreator 51
TsmSetCurrentFepCreator 51
TsmSetSystemFepCreator 52
Text Services FEP Plugin Functions 52
TsmLibFepClose 52
TsmLibFepCommitAction 53
TsmLibFepDrawModeIndicator 54
TsmLibFepDrawOption 55
TsmLibFepHandleEvent 55
TsmLibFepMapEvent 57
TsmLibFepOpen 57
TsmLibFepReset 58
TsmLibFepTerminate 58
TsmLibGetFepInfo 59

A TextServicesFep.h **61**

Index **67**

About This Document

Intended Audience

This document intended for developers who want to create a language front-end processor (FEP) for a Palm Powered™ device. A front-end processor comprises an engine for converting text from one form to another (for example, from ASCII to Chinese Hanzi) as well as a user interface for entering characters and confirming the conversion. The information in this document is also useful for developers, such as those implementing their own text controls, who want to interface with the FEP.

Besides fluency in the language for which you are creating the front-end processor, you need knowledge of Palm OS® and C/C++ programming.

IMPORTANT: The *Exploring Palm OS* series is intended for developers creating native applications for Palm OS Cobalt. If you are interested in developing applications that work through PACE and that also run on earlier Palm OS releases, read the latest versions of the *Palm OS Programmer's API Reference* and *Palm OS Programmer's Companion* instead.

FEP Developers

The entire content of this document is relevant to developers who are creating FEPs.

Other Developers

Developers who aren't creating FEPs but instead are implementing their own text controls, and who thus want to interface with the FEP,

may find the entire document useful. However, certain portions of this document are particularly important:

- Chapter 1, "Basic Concepts."
- Chapter 2, "The FEP User Interface."
- Chapter 3, "Creating a FEP Shared Library," from "Event Flow in a FEP" on page 21 through "FEP Type and Creator ID" on page 23.
- Chapter 4, "Text Services Manager Reference."
- "FEP Events" on page 33, in Chapter 5, "Text Services FEP Reference."

Requirements

This document assumes you are using the following versions of the Palm OS development tools:

- Palm OS Developer Suite
- the most recent version of Palm OS Resource Editor
- the most recent version of Palm OS Simulator
- the appropriate ROM file for the Palm OS version and language you want to support

It also assumes that you have installed the latest Palm OS SDK and the appropriate language support.

IMPORTANT: FEPs based on this Sample FEP Kit are compatible only with Palm OS Cobalt.

What this Book Contains

The following topics are covered in this book:

Chapter 1, "Basic Concepts." This chapter tells you what a front-end processor (FEP) is in the Palm OS.

Chapter 2, "The FEP User Interface." This chapter provides some examples of FEP user interfaces used by the Sample FEP and by current Chinese and Japanese Palm Powered handhelds.

Chapter 3, "Creating a FEP Shared Library." This chapter describes the Sample FEP project and tells you how to modify it to create your own FEP.

Chapter 4, "Text Services Manager Reference." This chapter describes the Text Services Manager API, which serves as the connection between the front-end processor and the rest of the Palm OS.

Chapter 5, "Text Services FEP Reference." This chapter describes the front-end processor API. The front-end processor shared library that you design must conform to this API.

The *Exploring Palm OS* Series

This book is a part of the *Exploring Palm OS* series. Together, the books in this series document and explain how to use the APIs exposed to third-party developers by the fully ARM-native versions of Palm OS, beginning with Palm OS Cobalt. Each of the books in the *Exploring Palm OS* series explains one aspect of the Palm operating system and contains both conceptual and reference documentation for the pertinent technology.

As of this writing, the complete *Exploring Palm OS* series consists of the following titles:

- *Exploring Palm OS: Programming Basics*
- *Exploring Palm OS: Memory, Databases, and Files*
- *Exploring Palm OS: User Interface*
- *Exploring Palm OS: User Interface Guidelines* (coming soon)
- *Exploring Palm OS: System Management*
- *Exploring Palm OS: Text and Localization*
- *Exploring Palm OS: Input Services*
- *Exploring Palm OS: High-Level Communications*
- *Exploring Palm OS: Low-Level Communications*
- *Exploring Palm OS: Telephony and SMS*
- *Exploring Palm OS: Multimedia*
- *Exploring Palm OS: Security and Cryptography*

- *Exploring Palm OS: Creating a Front-End Processor*
- *Exploring Palm OS: Application Porting Guide*

Additional Resources

- Documentation

  PalmSource publishes its latest versions of this and other 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

- 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://kb.palmsource.com

1

Basic Concepts

This chapter describes several concepts that will help you understand what a front-end processor is and how it is used.

This chapter discusses the following topics:

- What Is a Front-End Processor?
- How Does a User Input Text?
- How Is Inline Input Processed?
- How Does the FEP Handle Conversion?
- What Is a FEP in the Palm OS?
- For More Information

What Is a Front-End Processor?

A **front-end processor** (FEP), also known as an **input method**, is a facility that automatically converts phonetic or syllabic characters into ideographic or complex characters. With a front-end processor, you can use the Latin characters found on a standard keyboard to input syllables that are then converted to characters in the target language, such as the thousands of characters used in languages like Japanese, Chinese, and Korean.

For example, text input in Japanese requires software for translating Romaji (phonetic Japanese that uses Latin characters) or Hiragana (syllabic Japanese) into Kanji (ideographic Chinese characters) or Katakana (syllabic characters used mainly for foreign words). One Hiragana sequence may correspond to more than one Kanji character. The front-end processor must grammatically parse sentences or clauses of Hiragana text and select the best combination of Kanji, Hiragana, and Katakana characters to represent that text.

How Does a User Input Text?

Most front-end processors perform the character conversion within the current line of text, a method known as **inline input**. The field code passes events to the FEP, which then returns information about the appropriate text to display. Special characters, such as space or linefeed, are often used to initiate or confirm conversion.

In the Palm OS®, a front-end processor is known as a **text service**. The **Text Services Manager** provides functions that let forms, fields, and text services communicate about what happens in the **active input field**—the location in which the user enters text and where the text service (such as the front-end processor) tells the field what text to display.

How Is Inline Input Processed?

The front-end processor processes the user input, called **raw text**, as it is entered. The field code then draws the text on the screen as entered. The front-end processor then *converts* the raw text, translating it from phonetic or syllabic to a more complex representation. Finally, it *confirms* the converted text upon user approval of the conversion. The front-end processor continually tells the field code to remove the confirmed text from the beginning of the active input area.

An exception to this process occurs when a user taps outside the active input area or otherwise causes the field to lose the focus. In this case, the user has implicitly requested confirmation of the existing text.

How Does the FEP Handle Conversion?

The field code works with the FEP to support inline conversion. This is the only method of FEP-related text entry supported by Palm OS. There is no floating window or bottom-line input, as is provided by some other operating systems. Text that is part of the inline conversion process (that is, text in the active input area) is underlined by the field code.

Because a sequence of characters rarely has a one-to-one mapping with a single word or character in the FEP's conversion dictionary, the FEP's user interface can be extended. For example, in the Sample FEP, when a user writes an acronym that has more than one possible meaning, the user must choose the appropriate meaning. In the Palm OS Sample FEP, the user interface presents these options in a pop-up list.

What Is a FEP in the Palm OS?

In the Palm OS, FEPs are text service components that are implemented as shared libraries. These libraries can be created by any third-party developer or Palm OS licensee. FEPs must be implemented according to the FEP Shared Library API, which is described in Chapter 5, "Text Services FEP Reference."

Applications can control the FEP through the Text Services Manager API, which is described in Chapter 4, "Text Services Manager Reference." This API provides functions to get and set the current FEP mode, which is useful for explicitly disabling the FEP when a field shouldn't allow in-line conversion.

For More Information

Palm provides a sample FEP shared library to help you get started.

- Chapter 2, "The FEP User Interface," describes the user interface of a sample FEP running on a Palm Powered™ device.
- Chapter 3, "Creating a FEP Shared Library," describes how to modify a sample FEP to create a new FEP shared library.
- Chapter 4, "Text Services Manager Reference," describes the Text Services Manager API.
- Chapter 5, "Text Services FEP Reference," describes the FEP Shared Library API. A FEP must implement the API described in this chapter.

2

The FEP User Interface

This chapter describes the user interface of the Sample FEP that is provided with the FEP Kit. This interface is similar to the interface that some real FEPs use, including the Palm OS® standard Simplified Chinese FEP, called the Pinyin FEP, and the standard Japanese FEP. This chapter also provides examples of input area designs for handhelds that rely on FEPs for entering characters.

This chapter covers the following concepts:

- Input Area Buttons
- Interactions with Forms and Fields

Input Area Buttons

Handheld devices that include FEPs typically provide either three or four extra input area buttons in addition to those provided on all Palm Powered™ handhelds. Like other input area buttons, tapping one of these buttons sends a `keyDownEvent` to the system. This event then affects what the user sees on the screen.

Standard FEP Buttons

Figure 2.1 shows an example of the three-button interface on a Simplified Chinese handheld.

Figure 2.1 Input area for a Simplified Chinese handheld

The three buttons correspond to "Convert," "Confirm," and "On/Off."

Convert triggers conversion of the text in the active field if there is no clause (converted text). If there is converted text, then it selects the next option. If the user has selected non-inline text and taps the Convert button, this text is used to “prime the pump.” This priming text gets passed to the FEP's conversion engine as if the user had entered it all with the FEP on (as if it were all raw inline text) and then tapped the Convert button.

Confirm accepts the currently entered text. If there is no **clause** (a grammatically meaningful group of characters), then all of the raw inline text is dumped into the field, and the inline text area becomes empty. If there is a clause, then only that clause is dumped into the field (that is, removed from the start of the inline text area).

On/Off turns the FEP on or off.

The **Keyboard** buttons shown in Figure 2.1 just bring up a keyboard dialog with the chosen character set.

The Change Mode Button

For languages such as Japanese, which have multiple character sets, the interface also contains a "Change Mode" button, shown in Figure 2.2.

Figure 2.2 Input area for a Japanese handheld

Change Mode toggles the FEP's input mode. For the Palm OS standard Japanese FEP, it toggles between Hiragana and Katakana. If there is no clause, but there is raw inline text, then it transliterates the text between Hiragana and Katakana. If there is a clause, then it converts the clause to Hiragana (if Kanji or Katakana) and Katakana (if Hiragana).

Interactions with Forms and Fields

In order to capture and convert text, the FEP must interact extensively with Palm OS and with an application's forms and fields. The following sections illustrate this process with examples.

The Sample FEP User Interface

In this section, we will use the Sample FEP to illustrate common FEP interface features. The Sample FEP converts acronyms into their full-length representations. Because the results of the sample FEP are all simple English phrases containing only low ASCII characters that are present on every language version of Palm OS, this sample project will run on any language version of Palm OS.

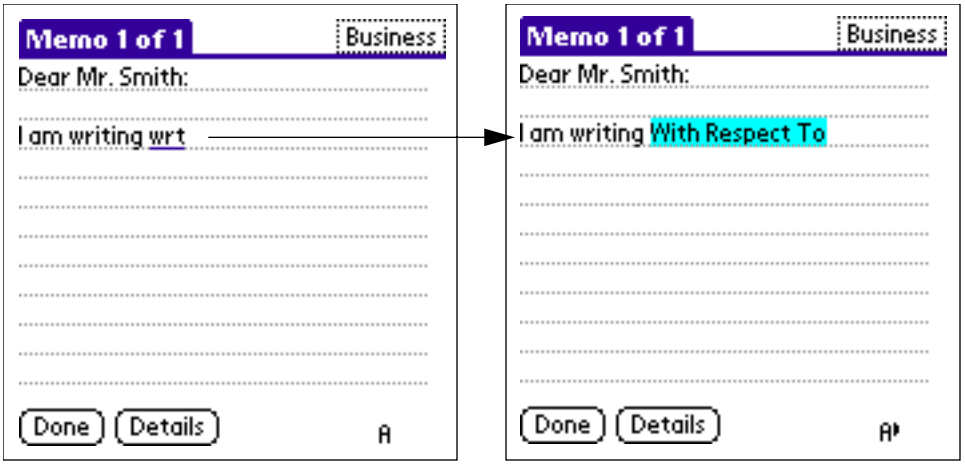
Inline Text Conversion

Suppose you want to begin a memo with the words, "With regard to your letter on the 28th,..." The Sample FEP lets you save time by writing the acronym, "wrt," instead of "With Regard To."

As the first step, create a new memo in Memo Pad and write the letters "wrt."

To convert the acronym to its long version, write the space character in the input area or tap the Convert button. This action displays the full-length version of the acronym (see Figure 2.3).

Figure 2.3 Sample FEP, converting “wrt”



Options Pop-up List

In the Sample FEP, if you try the conversion operation more than once, the options pop-up list appears. It contains a list of all possible matches for that acronym in the user dictionary (see Figure 2.4).

Figure 2.4 Sample FEP options list

Select the full-length version that is most appropriate, then tap the Confirm button. The final result contains the correct phrase.

A Simplified Chinese FEP User Interface

This section illustrates the Pinyin FEP.

Inline Text Conversion

Suppose you want to create a new Address Book entry, including the City/Province Name, “Beijing.” The Pinyin FEP lets you enter Simplified Chinese syllables using Latin characters.

To begin, create a new Address Book entry and place the cursor in the City/Province field. Write the letters “beijing” in the input area.

To convert the word to its Chinese version, write the “space” character in the input area or tap the Convert button. This action displays the Chinese version (see Figure 2.5).

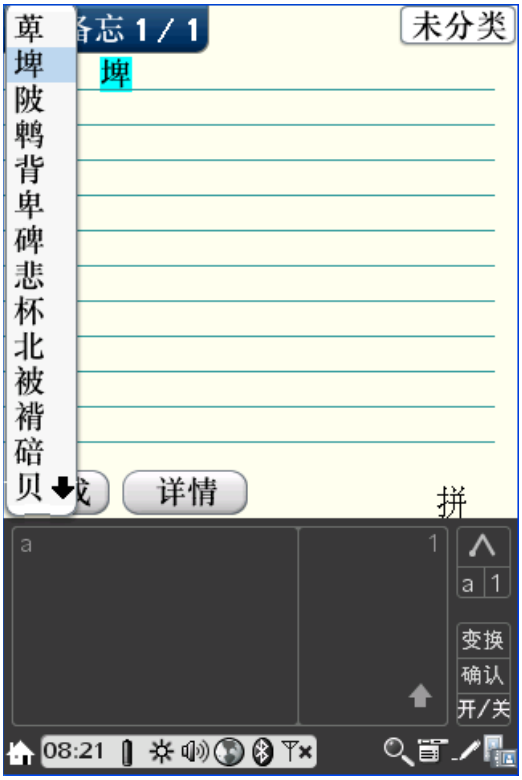
Figure 2.5 Completed conversion of “Beijing”

Tap the Confirm button to accept the characters and continue writing.

Options Pop-up List

In a FEP that supports an options pop-up list, if you write some inline text and then write the “space” character or tap the Convert button more than once, the pop-up list appears. For example, if you write the characters “bei,” and tap Convert twice, you get the pop-up list shown in Figure 2.6.

Figure 2.6 Pinyin FEP options pop-up list



Select the character or characters that are most appropriate. When finished, tap the Confirm button.

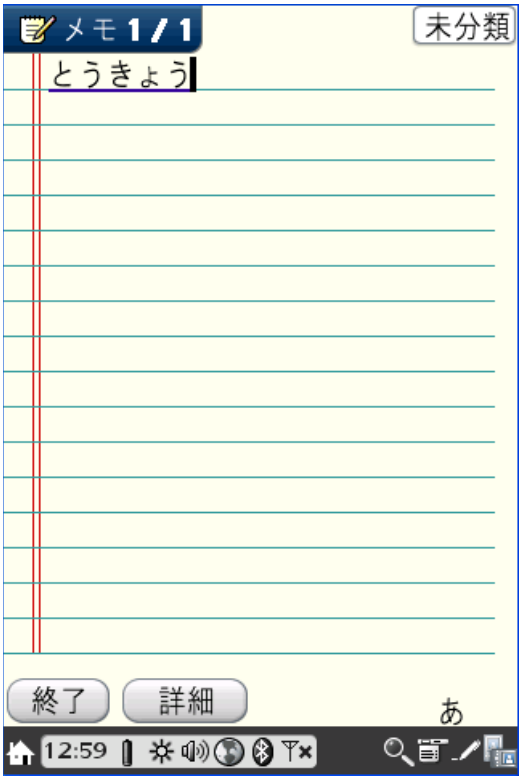
A Japanese FEP User Interface

This section illustrates the Japanese FEP, which shows how to use a four-button input area.

Inline Text Conversion

Suppose you want to enter the Kanji characters for “Tokyo” in the City Name field of an Address Book entry. Writing the two Romaji syllables, “tou” and “kyou,” results in the Hiragana characters shown in Figure 2.7.

Figure 2.7 “Tokyo,” pre-conversion

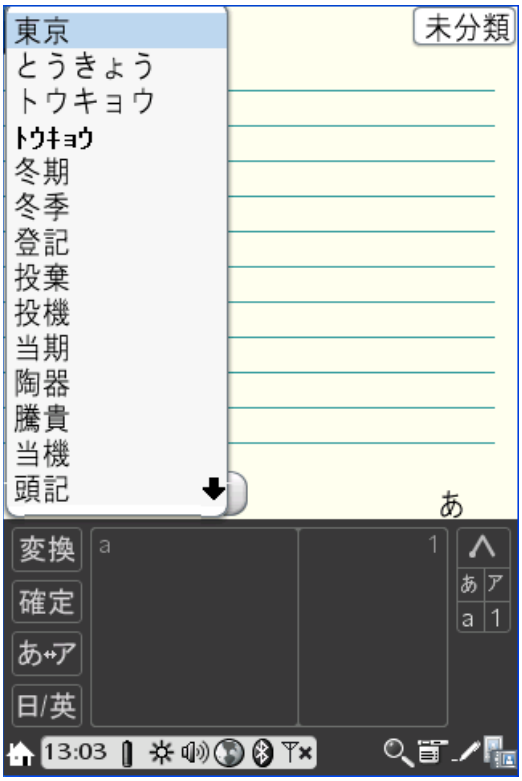


To get the correct Kanji characters, write the “space” character or tap the Convert button.

Options Pop-up List

When you enter the “space” character or tap the Convert input area button, the correct character may not appear immediately. If you enter “space” or tap the Convert button more than once, the options pop-up list appears. See Figure 2.8 for an example.

Figure 2.8 Japanese FEP options pop-up list



Select the character or characters that are most appropriate. When finished, tap the Confirm button.

The final result contains the correct combination of Kanji characters, as shown in Figure 2.9.

Figure 2.9 Completed Conversion of “Tokyo”

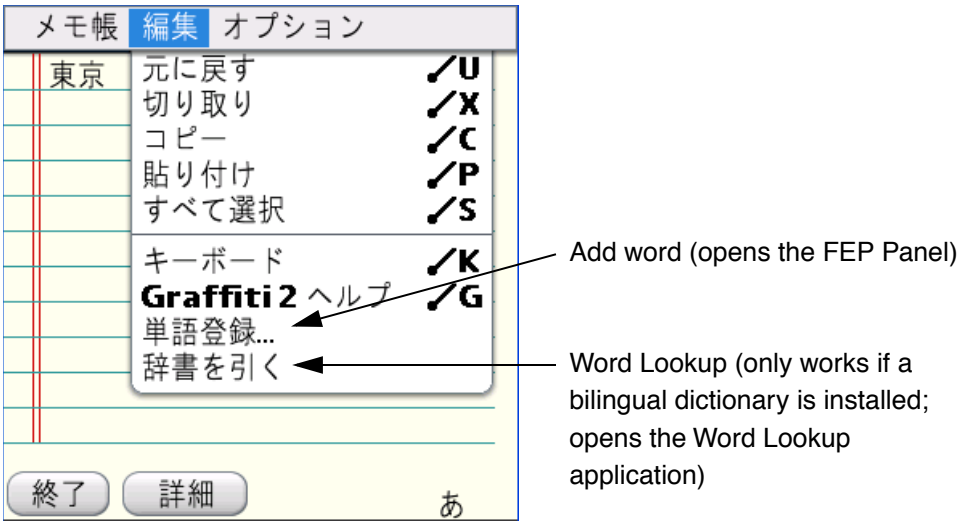
Edit Menu Items

A Palm OS FEP can automatically add a new menu item to any Edit menu that ends with the Graffiti® 2 Help item. This item calls a conversion dictionary that the user can edit.

Palm Powered handhelds may include a built-in bilingual dictionary for the user’s reference. This dictionary is not linked to the FEP, but to the Word Lookup application. Instead, the FEP can support a User Dictionary in addition to its own conversion dictionary. The user can add custom words and their converted forms to the User Dictionary.

Figure 2.10 illustrates the Edit menu items on a typical Japanese device.

Figure 2.10 Example of Japanese edit menu



The FEP Panel

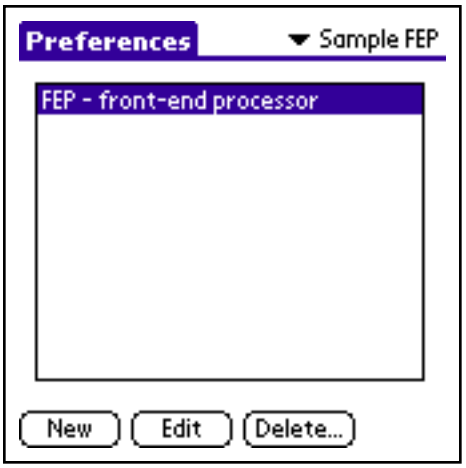
The FEP Panel lets users add words to the User Dictionary and maintain the list of added words. It can also let the user set FEP preferences.

You can access the FEP Panel, which contains the list of added words, from the Preferences application or from the **Edit > Add Word** menu item. Note that the Add Word item will automatically be included in any menu that contains the Graffiti 2 Help command (`sysEditMenuGraffitiCmd`), provided there also exists a panel (PRC with type `'panl'`) that has the same creator as the current FEP.

The FEP Panel is usually displayed on the pop-up list as “Sample FEP” or the corresponding name of the FEP used on the handheld. The following figures show the FEP Panel from a handheld running the Sample FEP. The user interface is in English by default.

Note that sample code for the FEP Panel is not provided in the FEP kit. Earlier versions of the kit for Palm OS Garnet do include FEP Panel sample code.

Figure 2.11 Sample FEP Panel



The New Dictionary Entry dialog appears when a user taps the New button or selects the Add Word option from the Edit menu in an application. The following figure shows the New Dictionary Entry dialog from the Sample FEP Panel.

Figure 2.12 Sample FEP New Dictionary Entry Dialog

3

Creating a FEP Shared Library

The Sample FEP Kit provides a Sample FEP shared library to help you get started with creating your own FEP. The Sample FEP Kit also includes a test application that will help you debug and test your FEP.

This chapter discusses the following topics:

- The Sample FEP
- Text Services Manager Server
- Event Flow in a FEP
- FEP Type and Creator ID
- Modifying the Sample FEP
- Debugging and Testing the FEP

The Sample FEP

The Sample FEP shipped with this document is a simple acronym converter: it takes an acronym as input and converts it to the spelled-out English version. PalmSource provides this sample code as a starting place for creating your own FEP. The following sections describe the basic structure of the Sample FEP project.

IMPORTANT: FEPs based on this Sample FEP Kit are compatible with Palm OS® Cobalt.

Sample FEP File List

The files that you will use with the Sample FEP project are described in the following table. All paths below (except the first one) are relative to the sample FEP directory: `\PDK\samples\SampleFep`

Table 3.1 Sample FEP File List

File Name	Description
Directory: PDK \headers	
TextServicesFep.h	Header file that contains most of the function declarations required to implement the FEP API. This header is available in the PDK \headers directory. For those developers without access to the PDK, see Appendix A for the contents of this header file.
Directory: \Dictionaries	
SampleFep-Med.pdb	Sample dictionary for use with the Sample FEP. It contains a list of acronyms and their spelled-out English equivalents. **Warning:** The Sample FEP will not run without this dictionary installed.
Directory: \Headers	
SampleFep.h	Public header file for the FEP. This file contains the custom defines and function declarations that are specific to a particular FEP. For example, this is where the FEP name is defined.
Directory: \rsc	
SampleFepLib.xrd	An XML resource description file that contains the sample FEP icon name and bitmap references. Bitmaps are stored in the `\SampleFep` subdirectory.
**Directory: **	
SampleFep.cpp	Source code for Sample FEP top-level functions.
SampleFepEngine.cpp	Source code for Sample FEP low-level conversion functions.
SampleFepEngine.h	Header file for the Sample FEP conversion engine.

Table 3.1 Sample FEP File List *(continued)*

File Name	Description
SampleFepEvents.cpp	Source code for Sample FEP event-handling functions.
SampleFepEvents.h	Header file for the Sample FEP event handling functions.
SampleFepGlobals.h	Header file that defines a structure typedef that is used to pass FEP state (global and session-specific) to all functions.
SampleFepOptions.cpp	Source code for Sample FEP options list functions.
SampleFepOptions.h	Header file for the sample FEP options list functions.
SampleFepPrv.h	Header file that contains the private FEP declarations which are used by multiple source files. For example, the FEP internal state structure uses some of these declarations.
SampleFepUtil.cpp	Source file for functions that are **not** top-level. These include functions called by top-level functions in `SampleFep.cpp`, `SampleFepEngine.cpp` (the dictionary engine), `SampleFepEvents.cpp` (event handling), and `SampleFepOptions.cpp` (options list handling).
SampleFepUtil.h	Header file for the Sample FEP utility functions.
Other files	The other files in this directory are Palm OS Developer Suite project files.

The TestSampleFep Application

The TestSampleFep application is also included in the Sample FEP Kit in SampleFep\Test. This application lets you switch between the default FEP and your new FEP for testing purposes. The application contains a text field and buttons that represent the input area buttons that you normally see on a handheld that uses a FEP: “Convert,” “Confirm,” “On/Off”, and “Activate/Deactivate”.

The TestSampleFep application requires that the FEP you are testing have the creator `'sfep'`.

The section, "Debugging and Testing the FEP" on page 26, describes this application in more detail.

FEP Code Structure

The FEP code is structured into three layers:

- `SampleFep.cpp` implements all of the top-level FEP functions described in Chapter 5, "Text Services FEP Reference."
- Below this level are `SampleFepEvents.cpp` and `SampleFepOptions.cpp`, which implement the FEP user interface.
- The user interface depends on the FEP engine, which is implemented by functions in the `SampleFepEngine.cpp` file.

Text Services Manager Server

The Text Services Manager has a server component that globally maintains the current FEP and current FEP mode across all processes and threads.

There can be only one current FEP in the system and one global FEP mode, even though there may be more than one user interface context that is using a FEP. In such a case, each would be using the same FEP, but the server maintains separate state objects for each client so it's always clear which is the active client.

There can be multiple clients running in different threads, so a FEP client must get its current mode from the server and not attempt to maintain its own mode. This is because the mode could be changed by a client in another thread. Use the function `TsmGetFepMode()` to get the FEP mode and `TsmSetFepMode()` to set the FEP mode. These functions interact with the server to get and set the mode.

When a FEP session starts or becomes the active session after having been inactive, the server calls the `TsmLibFepReset()` function in the FEP library. This gives the FEP a chance to start with a clean

slate; any internal globals it may have been maintaining for another session should get cleaned up here. It also alerts the FEP that the global FEP mode may have changed and it should check the mode and set it if necessary.

If you want to use globals in your FEP, ensure that they are thread safe. You can put them in the FEP instance record (`TsmFepStatusType`), or always reset them when the FEP receives the `TsmLibFepReset()` call.

Event Flow in a FEP

This section describes how Palm OS interacts with a FEP, including how events get passed to a FEP.

Initialization Sequence

The following sequence of calls is used to load the correct FEP when a new user interface context is created:

1. The Text Services Manager is initialized. It calls `TsmGetSystemFepCreator()` to determine the creator code of the system FEP. It then calls `TsmLibGetFepInfo()` for this FEP to find out whether or not the FEP should be loaded. If the FEP version number is invalid, the FEP will not be loaded. For more information about the FEP version number, read about the `TsmFepInfoType` data structure.
2. If the FEP should be loaded, then the Text Manager calls `TsmLibFepOpen()`. If that call succeeds, then the FEP is set as the current FEP.

The FEP should maintain some sort of reference counting in order to be aware if it is being called to be loaded more than once. In such a case all it needs to do is set up its globals and does not need to allocate a new set, since only once FEP session will be active at any time.

You can see how the Sample FEP uses the `gFepOpenCount` global to maintain reference counting.

System Events

The following event processing takes place above the level of field editing of text:

- Whenever `SysHandleEvent()` is called, it calls the Text Manager. If the event is a Text Services Manager virtual `keyDownEvent` (for example, one of the four Text Services Manager input area buttons, or a mode change virtual character), then the event gets re-posted as a `tsmFepButtonEvent` or `tsmFepModeEvent`. Otherwise, `TsmLibFepMapEvent()` is called. This lets the FEP remap certain events when appropriate. For example, the "space" `keyDownEvent` gets remapped to a Text Services Manager button "convert" event if there is an active inline session which contains text.
- Whenever `FrmHandleEvent()` is called with a `keyDownEvent`, `tsmFepButtonEvent`, or `tsmFepModeEvent`, and there is no active field, then it calls `TsmLibFepHandleEvent()`. This lets the user turn the FEP on and off, for example, even when the current form has no text field.

Field-Level Events

The following events occur when a user is editing text in a field:

Whenever `FldHandleEvent()` is called, it calls `TsmLibFepHandleEvent()`. If the FEP handles the event, then it returns `true`, and `FldHandleEvent()` skips its event processing. Based on the results returned in the status and action structures passed to `TsmLibFepHandleEvent()`, the field code will update text, redraw the field, and update the selection range or insertion point as appropriate.

For example, suppose the field gets a `keyDownEvent`. The FEP determines whether it is active ("on") or inactive ("off"). If the FEP handles the event, `TsmLibFepHandleEvent()` returns information that tells the field code what to do (for example, "move the insertion point one character to the right").

When `FldHandleEvent()` has finished processing the structures returned by `TsmLibFepHandleEvent()`, it calls `TsmLibFepCommitAction()`). This lets the FEP do things like

unlock/deallocate the buffers it uses to pass back inline text to the field code.

Whenever the field loses the focus (for example, the user tapped elsewhere on the screen or the form was closed), `TsmLibFepTerminate()` is called. This lets the FEP reset its state; it can also return information to the field about how things need to be updated (for example, if partially entered Hiragana characters need to be deleted).

At various times the field code will call `TsmLibFepReset()` to put the FEP into a known, safe state. The FEP should treat this like a call to `TsmLibFepTerminate()`, but without returning any action information.

Notes About Event Handling

When the user has written some text and confirmed it, the FEP has the option of posting a `tsmConfirmEvent` event. Eventually, this event reaches the top of the event queue. If the form is gone and another form is current, the new form receives the event, but that form was not the form that generated the event. To avoid this problem, the application must verify that the form ID contained in the event matches the current form ID.

FEP Type and Creator ID

The FEP creator ID must be unique, and should be assigned like any other creator ID by registering at http://dev.palmos.com/creatorid/. The FEP's type must be `sysFileTFep` (`'libt'`).

The name can be anything you like.

Modifying the Sample FEP

This section talks about how to handle various issues involved with writing FEPs.

- Changing the Locale
- Handling Text Services Manager Button Events
- Handling Other Events

- Handling the Mode Indicator
- Handling Auto-Yomi Events (Japanese only)
- Auto-Extending the Maximum Size of a Field
- Adding User Dictionary Functions

Changing the Locale

The Sample FEP uses a Latin character set. If you want to design a FEP for a different language, you must have access to a version of Palm OS that supports that language and its character set. You must select an appropriate prefix file for your target locale.

Handling Text Services Manager Button Events

As described in Chapter 2, "The FEP User Interface," a typical FEP receives input from three or four special Text Services Manager buttons in the input area. Tapping one of these buttons generates a `keyDownEvent` with one of the following characters: `vchrTsm1`, `vchrTsm2`, `vchrTsm3`, or `vchrTsm4`. When `SysHandleEvent()` passes these events to `TsmLibFepMapEvent()`, a corresponding Text Services Manager button event gets posted. This event eventually gets passed to the FEP through the form or field code.

One of the hardest user interface events to handle properly in a FEP is when the user selects text and then taps on the "Convert" button. For instance, if the FEP can't process all of the text at once, the FEP needs to be able to tell the caller that it took only part of the text.

Look at the `HandleButtonEvent` function in `SampleFepEvents.cpp` for an example, because this shifting of text can be difficult to get right. FEP buffer size is limited, and it can be difficult to communicate back to the field code when this limitation becomes an issue.

Handling Other Events

Some regular `keyDownEvents` (for example, space and linefeed) are given special meaning by the FEP. As with the FEP buttons, these get converted into Text Services Manager button events by `TsmLibFepMapEvent()` when appropriate (for example, if there is

no current conversion session, then the space and linefeed characters are given no special meaning by the FEP).

Handling the Mode Indicator

The FEP is given a chance to draw its own mode indicator in the space normally occupied by the handwriting recognition system's shift mode indicator. If the FEP is off, then the regular mode indicator is in effect.

Handling Auto-Yomi Events (Japanese only)

In the Japanese language, some words (especially names) may have pronunciations that differ from the usual pronunciation for that sequence of characters. The characters that represent the pronunciation are called *yomi*. The standard PalmSource Japanese Address Book contains corresponding yomi text fields for the Last Name, First Name, and Company Name fields.

Whenever the FEP dumps converted text, either as a result of a call to `TsmLibFepHandleEvent()` or `TsmLibFepTerminate()`, it also posts an "auto-yomi" event through `tsmConfirmEvent`. This event contains pointers to the yomi text and the resulting converted text. This event is used by the Japanese Address Book to automatically fill in the pronunciation field when the user is editing the Last Name, First Name, and Company Name fields.

In most cases, the auto-yomi event causes the yomi field to be filled in with the same characters used in the regular text field. The user can then change the yomi text manually if the pronunciation should be different.

If a specific name always has the same pronunciation, you can add an entry to the FEP User Dictionary through the FEP Panel. Then, when the user enters that name, the corresponding yomi text will always appear in the yomi text field.

Auto-Extending the Maximum Size of a Field

The Category Manager and the Keyboard application use features set up by the Text Services Manager with information provided by the FEP (by `TsmLibGetFepInfo()`). These features mainly

involve auto-extending the maximum size (in bytes) of a field so that the user can temporarily enter more text (for example, in a transitional character set such as Japanese Hiragana). This expanded text will later get converted into fewer bytes of the final character set (for example, Japanese Kanji).

Adding User Dictionary Functions

Data found in a front-end processor's user dictionary is typically used by the FEP engine during conversion and is viewed or edited in a FEP Panel, whose user interface must be supplied by the FEP. The sample FEP code shows an example of how to do this.

The format of the user dictionary is proprietary to each vendor; the Palm OS does not have a standard format. Currently there is no public API to get entries from the user dictionary or to add entries to the user dictionary.

When the user selects the "Add Word" command from the Edit menu, the `sysAppLaunchCmdFepPanelAddWord` command launches the FEP Panel that has the same creator ID as the FEP and the type `'panl'`.

The system passes the `FepPanelAddWordParamsType` structure to the FEP Panel with the `sysAppLaunchCmdFepPanelAddWord` launch code. This structure contains information about the new word to be added to the user dictionary.

Debugging and Testing the FEP

The TestSampleFep application makes it easy to test your FEP in the Palm OS Simulator. It lets you activate and deactivate your FEP. It also provides a text field so that you can enter and convert text.

The test application requires that your FEP have the creator `'sfep'`.

To debug the FEP:

1. Run Palm OS Simulator using a compatible ROM.
2. Install the FEP's conversion dictionary, if it is not embedded in the PRC file. (For the Sample FEP, install `SampleFep-Med.pdb`.)

3. Debug the application.
4. In Palm OS Simulator, tap the Activate button to make your FEP the current FEP. Until you deactivate the FEP or reset the handheld, all FEP calls will be handled by your FEP. The button should now say, "Deactivate."
5. To deactivate your FEP, launch the TestSampleFep application. Tap the Deactivate button to make the system FEP the current FEP. You can now safely delete or update your sample FEP.

4

Text Services Manager Reference

This chapter provides information about the public Text Services Manager (TSM) APIs.

Text Services Manager Constants 29
Text Services Manager Functions and Macros 30

The header file `TextServicesMgr.h` declares the API that this chapter describes.

Text Services Manager Constants

Feature Constants

Purpose Constants used with the `FtrGet()` function.

Declared In `TextServicesMgr.h`

Constants

`#define tsmFtrCreator sysFileCTextServices`
Creator ID of the Text Services Manager.

`#define tsmFtrNumFlags 0`
Selector passed to `FtrGet()` to get the Text Services Manager flags.

`#define tsmFtrFlagsHasFep 0x00000001L`
Indicates the bit set in the return value of `FtrGet()` if a FEP is installed.

TsmFepModeType Typedef

Purpose Specifies the input modes used by the functions `TsmGetFepMode()` and `TsmSetFepMode()`.

Declared In `TextServicesMgr.h`

Prototype
```
typedef uint16_t TsmFepModeType
```

Constants

`#define tsmFepModeDefault ((TsmFepModeType)0)`
The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.

`#define tsmFepModeOff ((TsmFepModeType)1)`
Indicates that there is no active FEP input mode (the FEP is off).

`#define tsmFepModeCustom ((TsmFepModeType)128)`
A custom FEP input mode. You can have more than one custom mode; the starting value is 128. Katakana is an example of a custom input mode for the Japanese FEP.

Text Services Manager Functions and Macros

TsmGetFepMode Function

Purpose Returns the current input mode for the active FEP.

Declared In `TextServicesMgr.h`

Prototype
```
TsmFepModeType TsmGetFepMode (void)
```

Parameters None.

Returns If there is an active FEP, returns the current mode for the active FEP. If there is no active FEP, returns `tsmFepModeOff`.

Comments The most common use for this function is to save the current FEP mode. You could then call `TsmSetFepMode()` to set the current mode to "off" and again to restore the saved mode once the application has finished using a special text field.

TsmSetFepMode Function

Purpose Sets the input mode for the active FEP.

Declared In `TextServicesMgr.h`

Prototype

```
TsmFepModeType TsmSetFepMode
    (TsmFepModeType inNewMode)
```

Parameters → *inNewMode*
The new FEP input mode.

Returns Returns the previous input mode. If there is no active FEP, returns `tsmFepModeOff`.

Comments The most common use for this function is to set the FEP mode to "off" while the application is using a special text field, and then to restore the previous mode. See `TsmGetFepMode()` for more information on saving and restoring the FEP mode.

One common reason for explicitly disabling the FEP in code is when a text field will only contain 7-bit ASCII (numeric fields automatically turn off the FEP). For example, if the application has a password field and the contents of that field will always be 7-bit ASCII, the application should turn off the FEP to help prevent the user from entering invalid characters into the field.

Another common case occurs when the application has a numeric field, but cannot just rely on the numeric field attribute. For example, if you want the user to be able to enter the minus ("-") sign, you cannot use a numeric field because the field code prevents the user from entering this character since it's not a digit or a period. In this case, you should make it a regular field and have the application screen the characters. The application should disable the FEP when such a pseudo-numeric field is active.

5

Text Services FEP Reference

This chapter provides information about the FEP API as declared in `TextServicesFep.h` (see Appendix A). For source code examples, see `SampleFep.cpp` in the Sample FEP.

This chapter also provides information about the system-level events specific to the Text Services Manager (TSM). These events and structures defined in the Palm OS® header files `EventCodes.h` and `Event.h`.

FEP Events 33
Text Services FEP Structures and Types 37
Text Services FEP Constants 44
Text Services FEP Launch Codes 46
Text Services FEP Functions 47
Text Services FEP Plugin Functions 52

Your FEP shared library must implement the API described in the section "Text Services FEP Plugin Functions" on page 52.

FEP Events

tsmConfirmEvent

Purpose Optionally sent by a FEP when converted text has been confirmed, either explicitly by the user or as a result of the field losing the focus.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct _TSMConfirmType {
   char *yomiText;
   uint16_t formID;
   uint16_t padding_1;
}
```

Fields

`yomiText`
A pointer to the raw text that the user entered during conversion, which corresponds to the converted text being confirmed.

`formID`
The ID of the form that was active when the converted text was confirmed. This is useful for proper event processing when `tsmConfirmEvent` is being generated while one form is being closed and other form is opened: the event won't be processed until after the new form has become active.

`padding_1`
Padding bytes.

Comments An application (such as the Address Book) can use the data contained in this event to automatically set the data in a pronunciation field, instead of forcing users to re-enter the same text that they just passed to the FEP.

tsmFepButtonEvent

Purpose Tapping on a Text Services Manager input area button posts a `keyDownEvent` with a virtual character code (`vchrTsm1` through `vchrTsm4`) and the command bit set in the event's modifier field. If the `keyDownEvent`'s character code matches one of the four that correspond to the input area buttons, `SysHandleEvent()` calls the Text Services Manager to remap the event to be a `tsmFepButtonEvent`.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct _TSMFepButtonType {
   uint16_t buttonID;
}
```

Fields `buttonID`

This field can have one of the values defined in "Button ID Constants" on page 44. Some FEPs may not use all of these values.

NOTE: The `tsmFepButtonShorten` and `tsmFepButtonLengthen` values don't correspond to any of the four input area buttons; typically these values are generated by a physical keyboard, and are used to indicate clause shortening and lengthening.

tsmFepModeEvent

Purpose Used to change the FEP mode. This includes turning the FEP off, and turning it on (in its default mode). FEP mode changes must be handled through events to ensure proper FEP/field code synchronization.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct _TSMFepModeEventType {
   uint16_t mode;
}
```

Fields `mode`

One of the constants described in "TsmFepModeType" on page 29.

tsmFepChangeEvent

Purpose Sent by the Text Services Manager when the FEP is changed, to make all threads aware of the change. This event is used only by the

Text Services Manager and a FEP does not need to pay attention to it.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct tsmFepChange {
   uint32_t creator;
} tsmFepChange
```

Fields

`creator`
: Creator ID of the new FEP.

tsmFepDisplayOptionsEvent

Purpose Sent by the Text Services Manager when the user has requested that the FEP display the options list. This event is used only by the Text Services Manager and a FEP does not need to pay attention to it.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct tsmFepDisplayOptions {
   uint16_t numOptions;
   uint16_t curOption;
   uint16_t maxOptionWidth;
} tsmFepDisplayOptions
```

Fields

`numOptions`
: Number of options in the list.

`curOption`
: The number of the currently selected option in the list. List items are numbered beginning with 0.

`maxOptionWidth`
: The maximum option width in pixels.

tsmFepSelectOptionEvent

Purpose Sent by the Text Services Manager to the FEP when an item is selected in the FEP options list.

For this event, the `EventType` `data` field contains the structure shown in the Prototype section, below.

Declared In `Event.h`

Prototype

```
struct tsmFepSelectOption {
   int16_t selection;
} tsmFepSelectOption
```

Fields

`selection`
The number of the selected option. List items are numbered beginning with 0.

Text Services FEP Structures and Types

FepPanelAddWordParamsType Struct

Purpose The parameter block for the launch command `sysAppLaunchCmdFepPanelAddWord`.

Declared In `TextServicesFep.h`

Prototype

```
typedef struct {
   const char *wordP;
   size_t wordLen;
} FepPanelAddWordParamsType
```

Fields

`wordP`
Pointer to the word to be added or looked up.

`wordLen`
Length of the word in `wordP`.

TsmFepActionType Struct

Purpose The FEP functions `TsmLibFepHandleEvent()` and `TsmLibFepTerminate()` use this structure to tell the caller what

needs to be done to update the text display to synchronize it with the FEP.

Declared In `TextServicesFep.h`

Prototype

```
typedef struct {
   size_t dumpLength;
   size_t primedLength;
   Boolean updateText;
   Boolean updateSelection;
   Boolean handledEvent;
} TsmFepActionType
```

Fields

`dumpLength`
: Tells the caller how many bytes of text in the `TsmFepStatusType.inlineText` data should be removed from the front of the inline area and made part of the regular (non-inline) text. A value of zero means that nothing is being dumped.

`primedLength`
: The FEP uses this field to tell the caller how much of the priming text (passed to it in `TsmFepEventType.primeText`) it was able to accept or use. The caller uses this value to trim unused bytes from the end of the inline area, because initially it assumes that all of the selected (priming) text became inline text.

`updateText`
: This field is set to `true` whenever the contents or length of the `TsmFepStatusType.inlineText` data has been changed.

`updateSelection`
: This field is set to `true` whenever the clause or highlighting offsets in the `TsmFepStatusType` structure have changed.

`handledEvent`
: This field is `true` when `TsmLibFepHandleEvent()` has completely handled the event, and thus the caller should not do any further processing of the event.

TsmFepEventType Struct

Purpose This structure is passed to the `TsmLibFepHandleEvent()` function. It contains extra information required by the FEP to correctly handle the event. This structure is filled in by the field object, and is read-only; the FEP does not need to update any of the fields in this structure.

Declared In `TextServicesFep.h`

Prototype

```
typedef struct {
   size_t penOffset;
   Boolean penLeading;
   Boolean formEvent;
   uint16_t padding;
   size_t maxInline;
   char *primeText;
   size_t primeOffset;
   size_t primeLen;
} TsmFepEventType
```

Fields

`penOffset`
: The offset (in bytes) from the beginning of the `TsmFepStatusType.inlineText` data to the offset of the character located at the event's `screenX` and `screenY` location. This field is only valid for `penDownEvents`.

  Note that this field will contain a negative number if the user taps on text before the start of the inline area, and it could also be past the end of the inline text area.

`penLeading`
: Indicates whether a `penDownEvent` occurred on the left side of the character following `penOffset`, or the right side of the character preceding `penOffset`. The value is `true` if the `penDown` Event was on the left (leading edge) side of the following character.

`formEvent`
: This field is `true` if `TsmLibFepHandleEvent()` is being called by the form code (when there is no active field), and thus the status record should not be modified.

`padding`
: Padding bytes.

`maxInline`
: The maximum number of bytes allowable in the inline text area. It is up to the FEP to constrain the results it passes back in the `TsmFepStatusType` record such that `convertedLen` + `pendingLen` is always less than or equal to this limit. This limit is calculated by the field object, based on the amount of text in the field, the current size of the inline area, and the maximum allowable text in the field.

`primeText`
: A pointer to text used to "prime" the conversion process. If there is no active inline area, its value is determined as follows: the user selects text in a field, and then the user turns the FEP on, taps the mode change button, or taps the convert button.

  The field code will set `primeText` to be the field's text pointer.

`primeOffset`
: The offset to the beginning of the selected text defined by the `primeText` pointer.

`primeLen`
: The length of the selected text defined by the `primeText` pointer.

NOTE: The value of `primeLen` might be greater than the maximum amount of text that the FEP can handle. In that case, the FEP should ignore text beyond what it can handle, and set up the `TsmFepActionType.primedLength` field with the amount of text it was able to use for the inline text.

TsmFepInfoType Struct

Purpose The `TsmFepInfoType` structure is returned by the `TsmLibGetFepInfo()` function, which is usually called before the FEP is actually opened.

Declared In `TextServicesFep.h`

Prototype

```
typedef struct {
   uint32_t apiVersion;
   uint32_t libVersion;
   uint32_t libMaker;
   CharEncodingType encoding;
   LmLanguageType language;
   size_t stackExtra;
   size_t fieldExtra;
} TsmFepInfoType
```

Fields

`apiVersion`

This field should always be set to `tsmFepAPIVersion`, which is a constant defined in `TextServicesFep.h` (see Appendix A). The `tsmAPIVersion` constant is a standard Palm OS version number of the format `x.y.z<release stage><release number>`

and encoded as a 32-bit value. For example, version 1.12b3 would be encoded as 0x01122003.

The value in this field is used by the Text Services Manager to decide if the FEP implements an appropriate version of the API. If the value returned by the FEP matches the current API version number in the major and minor fields, then the FEP can be used. Otherwise the FEP is ignored.

`libVersion`

The version number for any custom APIs implemented by the FEP library. This field is useful for code that calls any of the library's extended functions; for example, a function that accesses the user dictionary.

`libMaker`

A four character "FEP Maker" code. This should always be the same as the FEP creator ID. Otherwise, by convention this code should match the creator code used by any associated panel or application that is part of the FEP software package.

`encoding`
: A Text Manager character encoding value as defined in `PalmLocale.h`, for example `charEncodingPalmSJIS` for Japanese FEPs.

`language`
: A Locale Manager language code as defined in `LocaleMgrTypes.h`; for example, `lJapanese` for Japanese FEPs.

`stackExtra`
: The maximum amount of stack space in bytes that would be used by the FEP in response to the `TsmLibFepHandleEvent()` call.

`fieldExtra`
: The number of extra bytes needed to auto-expand "short" fields so that the user can enter enough pre-conversion text to correctly specify the post-conversion results.

  For example, some Japanese Kanji characters could require up to ten bytes of text entry in order to specify the Hiragana characters that will be converted into two double-byte Kanji characters. In this example, six extra bytes would be required to set the last two characters in a field to the converted Kanji. This value is primarily used by the Category code when the user is editing the names of categories.

TsmFepStatusType Struct

Purpose This structure is allocated by `TsmLibFepOpen()`, and returned to the caller. It is then passed to many of the FEP functions, until `TsmLibFepClose()` deallocates it. The FEP uses this structure to tell the field object code what to display, and how to display it.

Declared In `TextServicesFep.h`

Prototype

```
typedef struct {
   char *inlineText;
   size_t convertedLen;
   size_t pendingLen;
   size_t selectStart;
   size_t selectEnd;
   size_t clauseStart;
   size_t clauseEnd;
} TsmFepStatusType
```

Fields

`inlineText`
The reference number of the FEP shared library. This is filled in by the Text Services Manager after the call to `TsmLibFepOpen()` succeeds.

`convertedLen`
A pointer to the text that is controlled by the FEP. This text is often called the "active input area" text.

`pendingLen`
The amount of text (in bytes) in the `inlineText` data that has been entered but not yet converted. This text always follows the converted text.

`selectStart`
The offset (in bytes) from the beginning of the `inlineText` data to the beginning of the selected text.

`selectEnd`
The offset (in bytes) from the beginning of the `inlineText` data to the end of the selected text. If there is an insertion point, but no selection range, then this value will be the same as `selectStart`.

`clauseStart`
The offset (in bytes) from the beginning of the `inlineText` data to the beginning of the current clause text. Only converted text can contain clauses. If there is no converted text, or no clause, then this field should contain zero.

`clauseEnd`
The offset (in bytes) from the beginning of the `inlineText` data to the end of the current clause text. If there is no converted text, or no clause, then this field should contain zero.

NOTE: If the FEP is dumping text from the inline area into the field object, these offsets are still relative to the state of the inline text *before* any dumping has taken place. The `TsmLibFepCommitAction()` call should update the FEP's internal state to reflect the effect of dumping text.

NOTE: The FEP typically adds extra information to the end of this record, to maintain private information about the session.

Text Services FEP Constants

Button ID Constants

Purpose Possible values for the `buttonID` field in a `tsmFepButtonEvent` event.

Declared In `TextServicesFep.h`

Constants

`#define tsmFepButtonConvert 0`
The Convert button.

`#define tsmFepButtonConfirm 1`
The Confirm button.

`#define tsmFepButtonMode 2`
The Mode button.

`#define tsmFepButtonOnOff 3`
The On/Off button.

`#define tsmFepButtonShorten 4`
The Shorten button.

`#define tsmFepButtonLengthen 5`
The Lengthen button.

Comments The `tsmFepButtonShorten` and `tsmFepButtonLengthen` values don't correspond to any of the four input area buttons; typically these values are generated by a physical keyboard, and are used to indicate clause shortening and lengthening.

Error Codes

Purpose Error codes returned by FEP shared library functions.

Declared In `TextServicesFep.h`

Constants

`#define tsmErrFepCantCommit (tsmErrorClass | 2)`
The `TsmLibFepCommitAction()` function encountered an error.

`#define tsmErrFepCustom (tsmErrorClass | 128)`
FEPs can return custom error codes starting from here.

`#define tsmErrFepNeedCommit (tsmErrorClass | 1)`
The FEP is waiting for a `TsmLibFepCommitAction()` call.

`#define tsmErrFepNotOpen (tsmErrorClass | 3)`
The FEP is not open.

`#define tsmErrFepReentrancy (tsmErrorClass | 8)`
The FEP is currently running code in another thread and cannot process the call.

`#define tsmErrFepStillOpen (tsmErrorClass | 4)`
The FEP has additional contexts that are still active.

`#define tsmErrFepWrongAPI (tsmErrorClass | 5)`
The FEP library API version does not match the Text Services Manager API version.

`#define tsmErrFepWrongEncoding (tsmErrorClass | 6)`
The FEP has received data in the wrong encoding. Currently the OS doesn't do anything special when this error code is returned by the FEP.

`#define tsmErrFepWrongLanguage (tsmErrorClass | 7)`
The FEP has received data in the wrong language. Currently the OS doesn't do anything special when this error code is returned by the FEP.

`#define tsmErrUnimplemented (tsmErrorClass | 0)`
The FEP doesn't implement the function. Currently the OS doesn't do anything special when this error code is returned by the FEP.

Miscellaneous Constants

Purpose Miscellaneous constants.

Declared In `TextServicesFep.h`

Constants

```
#define tsmFepAPIVersion (sysMakeROMVersion(6, 0,
  0, sysROMStageRelease, 0))
```
Text Services Manager FEP API version information.

`#define tsmFtrNumFepStackExtra 128`
Selector used for `FtrGet()` to get the maximum number of extra stack bytes required by the FEP.

`#define tsmFtrNumFepFieldExtra 129`
Selector used for `FtrGet()` to get the maximum number of extra field bytes required by the FEP.

`#define tsmInvalidFepCreator 0`
Creator code used to indicate no FEP, for getting and setting the current and system FEP.

Text Services FEP Launch Codes

sysAppLaunchCmdFepPanelAddWord

Purpose Send this launch code to the FEP panel to add a word to the FEP user dictionary.

Declared In `CmnLaunchCodes.h`

Prototype `#define sysAppLaunchCmdFepPanelAddWord 87`

Parameters The launch code's parameter block pointer references a `FepPanelAddWordParamsType` structure that indicates the word to be added.

Text Services FEP Functions

TsmFepCommitAction Function

Purpose Unlocks or deallocates any buffers used to pass information back to the caller in the `TsmFepStatusType` record as a result of a `TsmLibFepHandleEvent()` or `TsmLibFepTerminate()` call.

Declared In `TextServicesFep.h`

Prototype
```
status_t TsmFepCommitAction (void)
```

Parameters None.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code in another thread and cannot process the call. A FEP can prevent this error by doing reference counting.

Comments This function also updates any status record or internal offsets that need adjusting because text is being dumped from the inline text area.

TsmFepHandleEvent Function

Purpose Tells the caller whether or not the FEP completely handled the event. Updates the `TsmFepStatusType` record as appropriate, and sets fields in the `TsmFepActionType` record to tell the caller what needs to be updated.

Declared In `TextServicesFep.h`

Prototype
```
status_t TsmFepHandleEvent
    (const EventType *inEventP,
    const TsmFepEventType *inTsmEventP,
    TsmFepStatusType **ioStatusP,
    TsmFepActionType *outActionP)
```

Parameters → *inEventP*
A pointer to a system event record, such as a typical `penDownEvent` or `keyDownEvent`.

→ *inTsmEventP*
A pointer to a `TsmFepEventType` structure, which contains extra information about the event.

↔ *ioStatusP*
Pointer to a status pointer for this context.

← *outActionP*
A pointer to a `TsmFepActionType` structure, which the FEP fills in with information that the caller needs to know to correctly update the display to reflect the current FEP state.

Returns Return one of the following:

`errNone`
The event was handled successfully (`outActionP.handledEvent` is `true`), or the event was not completely handled by this function (`outActionP.handledEvent` is `false`).

`tsmErrFepReentrancy`
The FEP is currently running code.

`tsmErrFepNeedCommit`
The FEP is waiting for a `TsmLibFepCommitAction()` call.

See Also FEP Events

TsmFepMapEvent Function

Purpose Determines whether or not an event should be remapped by the FEP. If it needs to be remapped, then it posts the remapped event to the event queue.

Declared In `TextServicesFep.h`

Prototype

```
Boolean TsmFepMapEvent
   (const EventType *inEventP)
```

Parameters → *inEventP*
A pointer to the event record.

Returns `true` if the event was remapped.

Comments This function is commonly used to remap FEP button shortcut characters (that is, space or linefeed) to their FEP button equivalents. For example, it maps the shift left and right arrow

`keyDownEvents` to shorten/lengthen clause events. Note that the remapping is conditional on the state of the FEP. For example, a space is only remapped to a convert event if the FEP has inline text, otherwise it gets treated like a regular space character (no remapping).

TsmFepOptionsList Function

Purpose Pops up the list of options for the FEP.

Declared In `TextServicesFep.h`

Prototype

```
void TsmFepOptionsList (uint16_t iNumOptions,
    uint16_t iCurOption, uint16_t iMaxOptionWidth)
```

Parameters

→ *iNumOptions*
Number of options in the list.

→ *iCurOption*
The currently selected item.

→ *iMaxOptionWidth*
Maximum width of the list in pixels.

Returns The index of the list item selected, or `noListSelection` if no item was selected.

Comments This function queues an event to pop up the FEP options list. When this event is handled by `TsmFepHandleEvent()`, the `TsmLibFepDrawOption()` function is called to draw each item in the list. This function is passed the item number and the bounds in which the option item is to be drawn.

TsmFepReset Function

Purpose Calls through to the current FEP's `TsmLibFepReset()` function, which resets the FEP by clearing all buffers and setting the state back to raw text.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmFepReset (void)
```

Parameters None.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code and cannot process the call.

Comments This function does not change the mode. It also ignores any pending commits (see `TsmLibFepCommitAction()`).

TsmFepTerminate Function

Purpose Calls through to the current FEP's `TsmLibFepTerminate()` function, which ends the conversion session, if active, updates the `TsmFepStatusType` record with the new status, and fills in the `TsmFepActionType` record to tell the caller what needs to be updated.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmFepTerminate
    (TsmFepStatusType **ioStatusP,
    TsmFepActionType *outActionP)
```

Parameters

↔ *ioStatusP*
Pointer to the FEP's status record.

← *outActionP*
Pointer to the FEP's action record.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code and cannot process the call.

TsmGetCurrentFepCreator Function

Purpose Gets the creator ID of the current FEP.

Declared In `TextServicesFep.h`

Prototype

```
Boolean TsmGetCurrentFepCreator
    (uint32_t *oFepCreatorP)
```

Parameters

← *oFepCreatorP*
Pointer to the current FEP creator ID. If there is no current FEP, then **oFepCreatorP* contains `tsmInvalidFepCreator`.

Returns `true` if there is a current FEP.

See Also `TsmSetCurrentFepCreator()`, `TsmGetSystemFepCreator()`

TsmGetSystemFepCreator Function

Purpose Gets the creator ID of the system FEP. The system FEP is the FEP that will be used to initialize the current FEP when you perform a soft-reset of the handheld.

Declared In `TextServicesFep.h`

Prototype

```
Boolean TsmGetSystemFepCreator
    (uint32_t *oFepCreatorP)
```

Parameters

← *oFepCreatorP*
Pointer to the returned FEP plugin library creator ID, or `tsmInvalidFepCreator` if there is no system FEP.

Returns `true` if there is a system FEP.

See Also `TsmSetSystemFepCreator()`, `TsmGetCurrentFepCreator()`

TsmSetCurrentFepCreator Function

Purpose Sets the current FEP to be the FEP with the specified creator ID, opens it and makes it usable.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmSetCurrentFepCreator
    (uint32_t iFepCreator)
```

Parameters

→ *iFepCreator*
FEP plugin library creator ID.

Returns `errNone` if the current FEP was changed to the FEP with the specified creator ID. Otherwise, it returns one of the following result codes:

`tsmErrFepWrongAPI`
The FEP library API version does not match the Text Services Manager API version.

`sysInvalidRefNum`
The FEP library could not be opened.

`tsmErrFepStillOpen`
The previous FEP is still open.

Comments All FEP plugin libraries are of type `sysFileTFep`.

See Also `TsmGetCurrentFepCreator()`, `TsmSetSystemFepCreator()`

TsmSetSystemFepCreator Function

Purpose Sets the creator ID of the system FEP.

Declared In `TextServicesFep.h`

Prototype
```
void TsmSetSystemFepCreator
    (uint32_t iFepCreator)
```

Parameters → *iFepCreator*
FEP plugin library creator ID.

Returns Nothing.

See Also `TsmGetSystemFepCreator()`, `TsmSetCurrentFepCreator()`

Text Services FEP Plugin Functions

Your FEP shared library must implement the functions described in this section.

TsmLibFepClose Function

Purpose Deallocates a `TsmFepStatusType` structure previously returned by `TsmLibFepOpen()`. If this closes the last active FEP session, then disposes of any shared information (for example, unlocks dictionary data).

Declared In `TextServicesFep.h`

Prototype
```
status_t TsmLibFepClose
    (TsmFepStatusType *ioStatusP)
```

Parameters ↔ *ioStatusP*
Status pointer for this context.

Returns Return one of the following:

`errNone`
No error; call succeeded.

`tsmErrFepNotOpen`
The FEP is not open.

`tsmErrFepReentrancy`
The FEP is currently running code in another thread.

`tsmErrFepStillOpen`
The FEP has additional contexts that are active.

Comments This function ignores any pending commits (see `TsmLibFepCommitAction()`).

See Also `TsmLibFepOpen()`

TsmLibFepCommitAction Function

Purpose Unlocks or deallocates any buffers used to pass information back to the caller in the `TsmFepStatusType` record as a result of a `TsmLibFepHandleEvent()` or `TsmLibFepTerminate()` call.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmLibFepCommitAction
    (TsmFepStatusType *ioStatusP)
```

Parameters ↔ *ioStatusP*
Status pointer for this context.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code in another thread and cannot process the call.

Comments This function also updates any status record or internal offsets that need adjusting because text is being dumped from the inline text area.

TsmLibFepDrawModeIndicator Function

Purpose If the FEP is active, then draws a mode indicator which corresponds to the FEP's current mode.

Declared In `TextServicesFep.h`

Prototype

```
Boolean TsmLibFepDrawModeIndicator
   (TsmFepModeType inFepMode, uint16_t gsiState,
   Coord x, Coord y)
```

Parameters

→ *inFepMode*
The FEP input mode; see "TsmFepModeType" on page 29. This is the current FEP mode as maintained by the server.

→ *gsiState*
Mode indicator state. It can have the values defined by `GsiShiftState` in `GraffitiShift.h`.

→ *x*
x coordinate of the location where the character is to be drawn (left bound).

→ *y*
y coordinate of the location where the character is to be drawn (top bound).

Returns `true` if the call drew the mode indicator.

Comments For Japanese, the recommended mode indicators are as follows:

FEP off, regular mode	lower-case, full-width Latin "a"
FEP off, shifted mode	upper-case, full-width Latin "a"
FEP on, default (Hiragana)	Hiragana "a"
FEP on, Katakana	Katakana "a"

For Chinese handhelds, which may have multiple FEPs, use a character that will identify the FEP that is currently active. For example, the Palm OS standard Pinyin FEP uses the "pin" character to indicate that the FEP is on.

NOTE: The mode indicator doesn't change based on the *state* of the FEP. For example, the FEP mode stays the same whether or not the inline session contains converted text.

TIP: You can use the constants `kMaxGsiWidth` and `kMaxGsiHeight` to limit the size of your mode indicator. These constants are defined in `GraffitiShift.h`. All of the pixels in the rectangle defined by these constants must be set (that is, erased or redrawn) if the FEP draws the indicator, to ensure proper updating.

TsmLibFepDrawOption Function

Purpose Draws an option in the FEP options list.

Declared In `TextServicesFep.h`

Prototype

```
void TsmLibFepDrawOption
    (const TsmFepStatusType *inStatusP,
    uint16_t iItemNumber,
    const RectangleType *iBounds)
```

Parameters

→ *`inStatusP`*
Pointer to the FEP status record.

→ *`iItemNumber`*
The item number of the option to draw.

→ *`iBounds`*
The bounds where the option is to be drawn.

Returns Nothing.

Comments This function is called by the list code, once for each item in the list. This function should draw the appropriate options item within the bounds indicated by *`iBounds`*.

TsmLibFepHandleEvent Function

Purpose Tells the caller whether or not the FEP completely handled the event. Updates the `TsmFepStatusType` record as appropriate, and

sets fields in the TsmFepActionType record to tell the caller what needs to be updated.

Declared In TextServicesFep.h

Prototype

```
status_t TsmLibFepHandleEvent
    (const EventType *inEventP,
    const TsmFepEventType *inTsmEventP,
    TsmFepStatusType *ioStatusP,
    TsmFepActionType *outActionP)
```

Parameters

→ *inEventP*
Pointer to a system event record, such as a typical penDownEvent or keyDownEvent.

→ *inTsmEventP*
Pointer to a TsmFepEventType structure, which contains extra information about the event.

↔ *ioStatusP*
Status pointer for this context.

← *outActionP*
Pointer to a TsmFepActionType structure, which the FEP fills in with information that the caller needs to know to correctly update the display to reflect the current FEP state.

Returns One of the following:

errNone
The event was handled successfully (*outActionP*.handledEvent is true), or the event was not completely handled by this function (*outActionP*.handledEvent is false).

tsmErrFepReentrancy
The FEP is currently running code.

tsmErrFepNeedCommit
The FEP is waiting for a TsmLibFepCommitAction() call.

TsmLibFepMapEvent Function

Purpose Determines whether or not an event should be remapped by the FEP. If it needs to be remapped, then this function posts the remapped event to the event queue.

Declared In `TextServicesFep.h`

Prototype

```
Boolean TsmLibFepMapEvent
    (const TsmFepStatusType *inStatusP,
    const EventType *inEventP)
```

Parameters

→ *inStatusP*
Pointer to the FEP status record.

→ *inEventP*
Pointer to the event record.

Returns `true` if the event was remapped.

Comments This function is commonly used to remap FEP button shortcut characters (that is, space or linefeed) to their FEP button equivalents. For example, it maps the shift left and right arrow `keyDownEvents` to shorten/lengthen clause events. Note that the remapping is conditional on the state of the FEP. For example, a space is only remapped to a convert event if the FEP has inline text, otherwise it gets treated like a regular space character (no remapping).

TsmLibFepOpen Function

Purpose Allocates and initializes a new instance of the `TsmFepStatusType` structure, and returns this structure to the caller. If this is the first `TsmLibFepOpen()` call, it should also set up any shared information, such as dictionary data.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmLibFepOpen
    (TsmFepStatusType **outStatusP)
```

Parameters

← *outStatusP*
Pointer to a pointer to the new instance (status) record.

Returns `errNone` if the call was successful; otherwise, returns a standard error code that indicates the nature of the problem, such as `memErrNotEnoughSpace` or `dmErrCantFind`.

Comments Note that typically a FEP will allocate extra space at the end of the `TsmFepStatusType` structure to hold extra information about the session.

See Also `TsmLibFepClose()`

TsmLibFepReset Function

Purpose Resets the FEP (input method) by clearing all buffers and setting the state back to raw text. However, does not change the mode.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmLibFepReset
    (TsmFepStatusType *ioStatusP)
```

Parameters

↔ *ioStatusP*
Status pointer for this context.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code and cannot process the call.

Comments This function ignores any pending commits (see `TsmLibFepCommitAction()`).

TsmLibFepTerminate Function

Purpose Ends the conversion session, if active. Updates the `TsmFepStatusType` record with the new status, and fills in the `TsmFepActionType` record to tell the caller what needs to be updated.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmLibFepTerminate
    (TsmFepStatusType *ioStatusP,
    TsmFepActionType *outActionP)
```

Parameters

↔ *ioStatusP*
Pointer to the FEP's status record

← *outActionP*
Pointer to the FEP's action record.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code and cannot process the call.

TsmLibGetFepInfo Function

Purpose Fills in the `TsmFepInfoType` structure with information about the FEP.

Declared In `TextServicesFep.h`

Prototype

```
status_t TsmLibGetFepInfo
    (TsmFepInfoType *outInfoP)
```

Parameters ← *outInfoP*
Pointer to the information record to be filled in.

Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code and cannot process the call.

Comments This function can and will get called before the FEP library has been opened, so potentially no globals have been set up. This function should just fill in the information record with the available information and return.

A

TextServicesFep.h

WARNING! This file is provided here for illustrative purposes only. It is normally considered “private” because its contents are subject to change. Subsequent releases of Palm OS® make no guarantee of compatibility; any code that depends on the contents of this file is not guaranteed to work with future releases of the OS.

```
/***********************************************************************
 *
 * Copyright (c) 1999-2003 PalmSource, Inc. All rights reserved.
 *
 * File: TextServicesFep.h
 *
 * Release: Palm OS 6.0
 *
 * Description:
 * Header file for calling the FEP or its associated user dictionary editor.
 *
 ***********************************************************************/

#ifndef _TEXTSERVICESFEP_H_
#define _TEXTSERVICESFEP_H_

#include <PalmTypes.h>
#include <Event.h>  // EventType
#include <SystemMgr.h>  // sysAppLaunchCmdCustomBase
#include <TextServicesMgr.h>  // TsmFepModeType
#include <TextMgr.h>  // CharEncodingType


/***********************************************************************
 * Public constants
 ***********************************************************************/

// Our uint32_t version number available in TsmFepInfoType.apiVersion
// 0xMMmfsbbb, where MM is major version, m is minor version
// f is bug fix, s is stage: 3-release,2-beta,1-alpha,0-development,
// bbb is build number for non-releases
```

```
// V1.12b3   would be: 0x01122003
// V2.00a2   would be: 0x02001002
// V1.01     would be: 0x01013000

#define tsmFepAPIVersion(sysMakeROMVersion(6, 0, 0, sysROMStageRelease, 0))

// Creator code used to indicate no FEP, for get/set of current/system FEP.
#define tsmInvalidFepCreator 0

// Possible values for the .buttonID field in a tsmFepButtonEvent event.
#define tsmFepButtonConvert 0
#define tsmFepButtonConfirm 1
#define tsmFepButtonMode 2  // Was tsmFepButtonKana
#define tsmFepButtonOnOff 3
#define tsmFepButtonShorten 4
#define tsmFepButtonLengthen 5

// Selector used with call to FtrGet(tsmFtrCreator, xxx) to get the
// max number of extra stack bytes required by the FEP.
#define tsmFtrNumFepStackExtra 128

// Selector used with call to FtrGet(tsmFtrCreator, xxx) to get the
// max number of extra field bytes required by the FEP.
#define tsmFtrNumFepFieldExtra129

// Errors specific to the Text Services Fep library.
#define tsmErrUnimplemented (tsmErrorClass | 0)
#define tsmErrFepNeedCommit (tsmErrorClass | 1)
#define tsmErrFepCantCommit (tsmErrorClass | 2)
#define tsmErrFepNotOpen (tsmErrorClass | 3)
#define tsmErrFepStillOpen (tsmErrorClass | 4)
#define tsmErrFepWrongAPI (tsmErrorClass | 5)
#define tsmErrFepWrongEncoding (tsmErrorClass | 6)
#define tsmErrFepWrongLanguage (tsmErrorClass | 7)
#define tsmErrFepReentrancy (tsmErrorClass | 8)
#define tsmErrFepCustom (tsmErrorClass | 128)

/***********************************************************************
 * Public types
 ***********************************************************************/

// Structure returned by TsmLibGetFepInfo routine.
typedef struct {
   uint32_tapiVersion;  // Tsm API implemented by library.
   uint32_tlibVersion;  // Custom API implemented by library.
   uint32_tlibMaker;  // Who made this input method (creator).

   CharEncodingType encoding;  // e.g. charEncodingPalmLatin
```

```
   LmLanguageTypelanguage;  // e.g. lJapanese

   size_t stackExtra;  // Extra stack space needed by FEP
   size_t fieldExtra;  // Extra field space needed by FEP.
} TsmFepInfoType;

// Structure returned by TsmFepHandleEvent/TsmFepTerminate routines
// Note that the updateText and updateSelection flags are for efficiency
// only - the field code can use these to reduce the amount of redrawing
// required.
typedef struct {
   size_t dumpLength;  // Length of text to dump (or zero)
   size_t primedLength;  // Length of priming text used by FEP

   Boolean updateText;  // True -> update inline text.
   Boolean updateSelection;  // True -> update selection range.
   Boolean handledEvent;  // True -> Fep handled event.
   Boolean reserved;
} TsmFepActionType;

// Structure passed to TsmFepHandleEvent routine.
typedef struct {
   size_t penOffset;  // Offset (relative to start of inline text)
                      // of event's screenX/screenY location.
   Boolean penLeading;  // True -> position is on leading edge of the
                        // character at penOffset.
   Boolean formEvent;  // True -> caller is form code, thus NO CHANGES
                       // to TsmStatusRec are allowed.
   uint16_t padding;
   size_t maxInline;  // Max allowable size of inline, in bytes.
   char *primeText;  // ptr to selected text (if inline not active)
   size_t primeOffset;  // Offset to selected text.
   size_t primeLen;  // Length of selected text.
} TsmFepEventType;

// Structure exchanged with many FEP routines. This is how
// the FEP tells the editing code what to display, and how
// to display it. Note that it's also the context record for the
// FEP, thus additional (private) conversion information will
// typically be appended by the FEP.
typedef struct {
   char *inlineText;  // ptr to inline text.

   size_t convertedLen;  // Length of converted text.
   size_t pendingLen;  // Length of unconverted (pending) text.

   size_t selectStart;  // Start of selection range.
   size_t selectEnd;  // End of selection range (can extend past
```

```
                     // end of inline text)

   size_t clauseStart;  // Start of converted clause highlighting
   size_t clauseEnd;  // End of converted clause highlighting
} TsmFepStatusType;

// Parameter block passed with the sysAppLaunchCmdFepPanelAddWord command,
// when the user selects "Add Word..." from the system edit menu.
typedef struct
{
   const char* wordP;  // Ptr to word to add to FEP's user dictionary.
   size_t wordLen;  // Length of word.
} FepPanelAddWordParamsType;

/***********************************************************************
 * Public functions
 ***********************************************************************/

#ifdef __cplusplus
extern "C" {
#endif

// Get the creator of the system FEP.
Boolean TsmGetSystemFepCreator(uint32_t *oFepCreatorP);

// Set the creator of the system FEP.
void TsmSetSystemFepCreator(uint32_t iFepCreator);

// Get the creator of the current FEP.
Boolean TsmGetCurrentFepCreator(uint32_t *oFepCreatorP);

// Set the creator of the current FEP, and make it active.
status_t TsmSetCurrentFepCreator(uint32_t iFepCreator);

status_t TsmFepHandleEvent(const EventType* inEventP,
                           const TsmFepEventType* inTsmEventP,
                           TsmFepStatusType **ioStatusP,
                           TsmFepActionType *outActionP);

Boolean TsmFepMapEvent(const EventType *inEventP);

status_t TsmFepTerminate(TsmFepStatusType **ioStatusP, TsmFepActionType
*outActionP);

status_t TsmFepReset(void);

status_t TsmFepCommitAction(void);
```

```
// Display a deferred options list. Used by native FEPs in 6.0 to display
// an options list at a later time, to avoid re-entrancy problems.
void TsmFepOptionsList(uint16_t iNumOptions,
                       uint16_t iCurOption,
                       uint16_t iMaxOptionWidth);


/***********************************************************************
 * FEP Shared Library routines. These are the declarations of the
 * functions inside of the current FEP library, which will be called by the
 * cover routines above.
 ***********************************************************************/

// Open up an instance of the Fep. The Fep is responsible for allocating
// the TsmFepStatusType structure (to which it might append additional
// context information) and returning back a pointer to it.
status_t TsmLibFepOpen(TsmFepStatusType** outStatusP);

// Close down an instance of the Fep. The Fep is responsible
// for disposing of the TsmFepStatusType which it allocated in TsmLibFepOpen().
status_t TsmLibFepClose(TsmFepStatusType* ioStatusP);

// Return information about the Fep in the TsmFepInfoType structure.
status_t TsmLibGetFepInfo(TsmFepInfoType* outInfoP);

// Handle an event passed in <inEventP>. Additional information about the event
// is passed in the TsmFepEventType structure. Update the inline text data in
// the TsmFepStatusType, and tell the caller what happened by setting up the
// TsmFepActionType structure (including whether the event was handled by the
// Fep).
status_t TsmLibFepHandleEvent(const EventType* inEventP,
                              const TsmFepEventType* inTsmEventP,
                              TsmFepStatusType* ioStatusP,
                              TsmFepActionType* outActionP);

// Decide if <inEvent> should be remapped to some other event. If so, return
// true. If we return true, then go ahead and perform the remapping by posting
// a new  event with the remapped info.
Boolean TsmLibFepMapEvent(const TsmFepStatusType* inStatusP,
                          const EventType* inEventP);

// Terminate an inline session. Typically this involves 'dumping' all of the
// converted text, and potentially deleting any untransliterated input text.
// As with TsmLibFepHandleEvent, update the inline text data in the
// TsmFepStatusType, and indicate what was done in the TsmFepActionType.
status_t TsmLibFepTerminate(TsmFepStatusType* ioStatusP,
                            TsmFepActionType*  outActionP);
```

```
// Reset an inline session. The state of the Fep is reset to empty, raw
// text, nothing to dump, etc. This call should only be made when the
// conversion results are not required, otherwise TsmTerminate should be used.
status_t TsmLibFepReset(TsmFepStatusType* ioStatusP);

// The caller has processed the action which was returned by either the
// TsmHandleEvent or TsmTerminate routine, so it is now safe to reset any
// temporary state information (e.g. dumped text) in <ioStatus>.
status_t TsmLibFepCommitAction(TsmFepStatusType* ioStatusP);

// Draw the Fep mode indicator at location <x,y>.
Boolean TsmLibFepDrawModeIndicator(TsmFepModeType inFepMode,
                                   uint16_t gsiState,
                                   Coord x,
                                   Coord y);

// Draw an option in the FEP options list.
void TsmLibFepDrawOption(const TsmFepStatusType *inStatusP,
                         uint16_t iItemNumber,
                         const RectangleType* iBounds);

#ifdef __cplusplus
}
#endif

#endif
```

Index

Symbols

_TSMConfirmType 34
_TSMFepButtonType 35
_TSMFepModeEventType 35

A

active input area 2
API version number 41
auto-yomi events 25, 34

B

button ID constants 44
buttons
    input area 5

C

clause 6
confirmation 2
conversion, of text 2, 7, 9, 11

E

Edit menu 13
error codes 45
event flow 21
event handling 23
Event.h 33
EventCodes.h 33
events 33
    auto-yomi 25, 34

F

FEP
    code structure 20
    definition 1
    event flow 21
    resetting 23
FEP events 33
FEP Panel
    adding entries 25
    creator ID 26
    interface 14
FepPanelAddWordParamsType 37
field
    extending size 25
    processing raw text 2
field-level events 22
FldHandleEvent 22
form 2, 34
front-end processor. *See* FEP 1

H

Hiragana 1

I

initialization sequence 21
inline input. 2
input area 35, 44
input area buttons 5, 35, 44
input method 1
input mode 7, 29

K

Kanji 1
Katakana 1

L

locale 24
Locale Manager 42

M

mode indicator 25, 54

O

options pop-up list 11, 19

P

penDownEvent 39
priming text 38

R

raw text 2
remap characters 48, 57
resetting the FEP 23
Romaji 1

S

Sample FEP
    code structure 20
    file list 17
    modifying 23
Sample FEP Kit 17
Sample FEP structure 17
SampleFep.cpp 33
shared libraries 3
shift indicator 25, 54
space and linefeed characters 24
sysAppLaunchCmdFepPanelAddWord 46
sysInvalidRefNum 51
system events 22

T

TestSampleFep 26
TestSampleFep project 19
text service 2
Text Services Manager 2
TextServicesFep.h 33
TextServicesMgr.h 29
TSM 2
tsmConfirmEvent 33
tsmErrFepCantCommit 45
tsmErrFepCustom 45
tsmErrFepNeedCommit 45, 48, 56
tsmErrFepNotOpen 45
tsmErrFepReentrancy 45, 47, 48, 50, 53, 56, 58, 59
tsmErrFepStillOpen 45, 52
tsmErrFepWrongAPI 45, 51
tsmErrFepWrongEncoding 45
tsmErrFepWrongLanguage 45
tsmErrUnimplemented 45
TsmFepActionType 37
tsmFepAPIVersion 46
tsmFepButtonConfirm 44
tsmFepButtonConvert 44
tsmFepButtonEvent 34
tsmFepButtonLengthen 44
tsmFepButtonMode 44
tsmFepButtonOnOff 44
tsmFepButtonShorten 44
tsmFepChange 36
tsmFepChangeEvent 35
TsmFepCommitAction() 47
tsmFepDisplayOptions 36
tsmFepDisplayOptionsEvent 36
TsmFepEventType 39
TsmFepHandleEvent() 47
TsmFepInfoType 41
TsmFepMapEvent() 48
tsmFepModeCustom 30
tsmFepModeDefault 30
tsmFepModeEvent 35
tsmFepModeOff 30
TsmFepModeType 29
TsmFepOptionsList() 49
TsmFepReset() 49
tsmFepSelectOption 37
tsmFepSelectOptionEvent 37
TsmFepStatusType 42
TsmFepTerminate() 50
tsmFtrCreator 29
tsmFtrFlagsHasFep 29
tsmFtrNumFepFieldExtra 46
tsmFtrNumFepStackExtra 46
tsmFtrNumFlags 29
TsmGetCurrentFepCreator() 50
TsmGetFepMode() 30
TsmGetSystemFepCreator() 51
TsmHandleEvent 22
tsmInvalidFepCreator 46
TsmLibFepClose() 52
TsmLibFepCommitAction 22
TsmLibFepCommitAction() 53
TsmLibFepDrawModeIndicator() 54
TsmLibFepDrawOption() 55
TsmLibFepHandleEvent 22, 23
TsmLibFepHandleEvent() 55
TsmLibFepMapEvent() 57
TsmLibFepOpen() 57
TsmLibFepReset() 58
TsmLibFepTerminate() 58
TsmLibGetFepInfo() 59
TsmSetCurrentFepCreator() 51

TsmSetFepMode() 31
TsmSetSystemFepCreator() 52

U

User Dictionary
    functions 26
    interface 13

V

virtual characters 24, 34

Y

yomi characters 25