palmsource™

Front-End Processor Developer’s Guide

CONTRIBUTORS

Written by Susan Schneider
Engineering contributions by Ken Krugler, Vivek Magotra.

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

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

Front-End Processor Developer's Guide
Document Number 3029-002-HW
June 19, 2003
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
What this Guide Contains . . . viii
Additional Resources . . . ix

1 Basic Concepts 1
What is a Front-End Processor? . . . 2
How Does a User Input Text? . . . 2
How is Inline Input Processed? . . . 2
How Does the FEP Handle Conversion? . . . 3
What is an FEP in the Palm OS? . . . 3
Where do I go from here? . . . 4

2 The FEP User Interface 5
Silkscreen Buttons . . . 6
Interactions with Forms and Fields . . . 7
Inline Text Conversion . . . 7
Options Pop-Up List . . . 8
The Edit Menu items . . . 9
User Dictionary Preferences Panel . . . 10

3 Creating an FEP Shared Library 13
The Sample FEP . . . 14
Sample FEP File List . . . 14
The TestSampleFep Project . . . 17
Sample FEP Targets . . . 17
FEP Code Structure . . . 18
Event Flow in an FEP . . . 18
Boot Sequence . . . 18
System Events . . . 19
Field-Level Events . . . 19
Notes About Event Handling . . . 20

FEP Type and Creator ID 20
Modifying the Sample FEP. 21
Creating Multi-Segment FEPs 21
Handling Global Data. 22
Changing the Locale 22
Handling Text Services Manager Button Events 22
Handling Other Events 23
Handling the Shift Indicator 23
Handling Auto-Yomi Events (Japanese only) 23
Auto-Extending the Maximum Size of a Field 24
Adding User Dictionary Functions 24
Debugging the FEP 25
Debugging the FEP on a Macintosh 25
Debugging the FEP on a PC 26
Testing the FEP. 27

4 Creating a User Dictionary Panel 29

The Sample User Dictionary Panel 30
Sample User Dictionary Panel File List 30
Sample User Dictionary Panel Targets. 31
Modifying the Sample User Dictionary Panel. 32
Assigning Creator and Database Types 32
Modifying the Panel Resource Files 32
Debugging the User Dictionary Panel 33
Testing the User Dictionary Panel. 34

5 The Text Services Manager API 37

Public Text Services Manager Data Structures 38
TsmFepModeType 38
Public Text Services Manager Functions 38
TsmGetFepMode. 38
TsmSetFepMode 39
Private Text Services Manager Functions. 41
TsmDrawMode 41
TsmFepCommitAction 41
TsmFepHandleEvent 42

TsmFepMapEvent 43
TsmFepOptionsList. 44
TsmFepReset 45
TsmFepTerminate 45
TsmGetCurrentFep 46
TsmGetCurrentFepCreator 47
TsmGetSystemFep 47
TsmGetSystemFepCreator 48
TsmHandleEvent. 49
TsmInit 49
TsmSetCurrentFep 50
TsmSetCurrentFepCreator. 50
TsmSetSystemFep 51
TsmSetSystemFepCreator 52

6 The Front-End Processor (FEP) API 53
Related Palm OS Data Structures 54
eventsEnum 54
SysEventType 54
FEP Events 55
tsmConfirmEvent 55
tsmFepButtonEvent 56
tsmFepModeEvent 56
FEP Data Structures. 57
TsmFepInfoType 57
TsmFepStatusType 59
TsmFepEventType 61
TsmFepActionType 62
FEP Functions 64
TsmLibFepOpen 64
TsmLibFepClose 64
TsmLibFepSleep 65
TsmLibFepWake 66
TsmLibFepCommitAction 66
TsmLibFepDrawModeIndicator 67
TsmLibFepHandleEvent. 68

TsmLibFepMapEvent 69
TsmLibFepReset 70
TsmLibFepTerminate 70
TsmLibGetFepInfo 71
TsmLibGetFepMode 71

A TextServicesPrv.h 73

Index 81

About This Document

Intended Audience

This document was originally intended only for developers who wanted to create a language front-end processor for a Palm Powered™ handheld. (A front-end processor comprises an engine for converting ASCII characters to the characters of another language (such as Japanese) as well as a user interface for entering characters and confirming the conversion.) However, 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 the Palm OS® and C/C++ programming.

This document assumes you are using standard Palm OS development tools such as Metrowerks CodeWarrior for the Palm OS, Constructor for the Palm OS, and Palm OS Emulator. It also assumes that you have installed the Palm OS SDK (Palm OS 4.0 or greater) and the appropriate language support.

FEP Developers

The entire contents 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 an FEP Shared Library," from "Event Flow in an FEP" on page 18 through "FEP Type and Creator ID" on page 20.
- Chapter 5, "The Text Services Manager API."
- Chapter 6, "The Front-End Processor (FEP) API."

What this Guide Contains

The following topics are covered in this guide:

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 describes the hardware user interface used by current Japanese Palm Powered devices and the software user interface used by a FEP.

Chapter 3, "Creating an FEP Shared Library." This chapter describes the Sample FEP project and tells you how to modify it to create your own FEP. (FEP developers can request the Sample FEP by contacting PalmSource at devinfo@corp.palm.com.)

Chapter 4, "Creating a User Dictionary Panel." This chapter describes Sample User Dictionary Panel project (this dictionary is part of the Sample FEP project) and tells you how to modify it to create your own interface to the user dictionary.

Chapter 5, "The Text Services Manager API." 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 6, "The Front-End Processor (FEP) API." This chapter describes the front-end processor API. The front-end processor shared library that you design must conform to this API.

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://www.palmos.com/dev/support/kb/

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 an FEP in the Palm OS?
- Where do I go from here?

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 a standard keyboard to generate the thousands of characters needed by languages such as 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 input method 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 area*—the location in a field 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, on a Japanese handheld, when Hiragana text is converted to Kanji, the user has the option of changing any individual phrase: lengthening it, shortening it, or selecting different Kanji options. In the Palm OS Japanese FEP, the user interface presents these options to the user through an *options pop-up list*.

What is an FEP in the Palm OS?

In the Palm OS, FEPs are text services 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 6, "The Front-End Processor (FEP) API."

The Palm OS and applications can control the FEP through the Text Services Manager API, which is described in Chapter 5, "The Text Services Manager API."

Where do I go from here?

- Chapter 2, “The FEP User Interface,” describes the user interface of a typical Japanese-language Palm Powered handheld.
- Chapter 3, “Creating an FEP Shared Library,” describes how to modify a Sample FEP to create a new FEP shared library.
- Chapter 4, “Creating a User Dictionary Panel,” describes how to create a user dictionary Preferences panel based on the Sample User Dictionary Panel.
- Chapter 5, “The Text Services Manager API,” describes the Text Services Manager API.
- Chapter 6, “The Front-End Processor (FEP) API,” describes the FEP Shared Library API.

2

The FEP User Interface

This chapter describes a typical Japanese FEP user interface. The hardware interface is based on the interface used by current Japanese Palm Powered™ handhelds.

This chapter covers the following concepts:

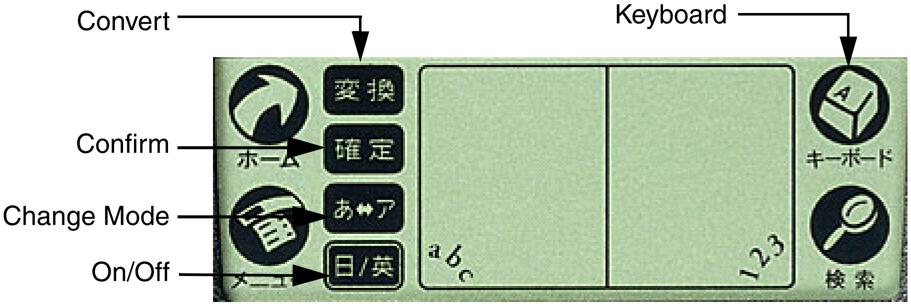
- Silkscreen Buttons
- Interactions with Forms and Fields

Silkscreen Buttons

Current Japanese Palm Powered handhelds use four silkscreen buttons in addition to those provided on English handhelds. Like other silkscreen buttons, tapping one of these buttons sends a `keyDownEvent` to the Palm OS®. This event then affects what the user sees on the screen.

The following figure shows an example of these buttons as they appear on a Palm Powered handheld that uses the default Japanese FEP:

Figure 2.1 Silkscreen Area of a Typical Japanese handheld



The four buttons correspond to "Convert," "Confirm," "Change Mode," and "On/Off."

Convert triggers conversion of the text in the active input area 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).

Change Mode toggles the FEP's input mode. For the default Japanese FEP, it toggles it between Hiragana and Katakana. If there's 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).

On/Off turns the FEP on or off.

The **Keyboard** button labeled in Figure 2.1 just brings up a Japanese version of the standard virtual keyboard.

Interactions with Forms and Fields

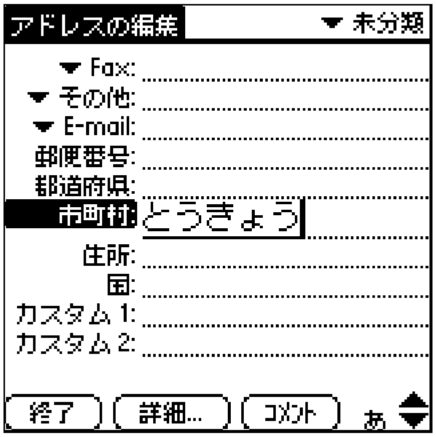
In order to capture and convert text, the FEP must interact extensively with Palm OS and with an application's forms and fields. The following sections describe some of the user interface features of a typical Japanese FEP in the Palm OS.

Inline Text Conversion

In this section, we will use the example of entering a name into Address Book to illustrate the software interface of a typical Japanese FEP.

Suppose you want to enter the word "toukyou" (Romaji characters for "Tokyo") in the City Name field of an Address Book entry. Typing in the two syllables, "tou" and "kyou," results in the Hiragana characters shown in Figure 2.2.

Figure 2.2 "Tokyo," Pre-Conversion



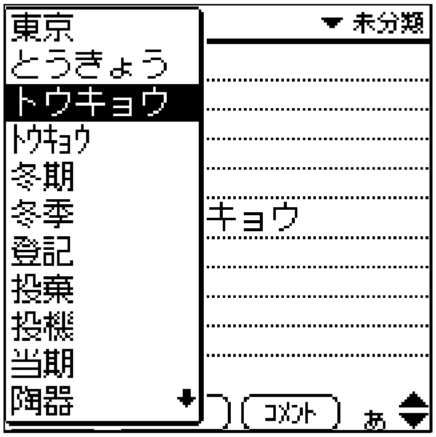
▼ Fax:
▼ その他:
▼ E-mail:
郵便番号:
都道府県:
市町村: とうきょう
住所:
国:
カスタム 1:
カスタム 2:

To get the correct Kanji character, you enter the Graffiti® 2 “space” character or tap the Convert button, which substitutes alternative Kanji characters until the ones you want appear.

Options Pop-Up List

When you enter the “space” character or tap the Convert silkscreen button, the correct character may not appear immediately. If you enter “space” or tap the Convert button enough times, the options pop-up list appears. See Figure 2.3 for an example.

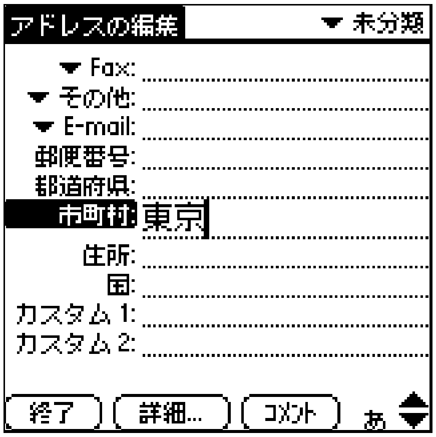
Figure 2.3 Options Pop-Up List



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 2.4.

Figure 2.4 Completed Conversion of "Tokyo"



Fax:
その他:
E-mail:
郵便番号:
都道府県:
市町村: 東京
住所:
国:
カスタム 1:
カスタム 2:

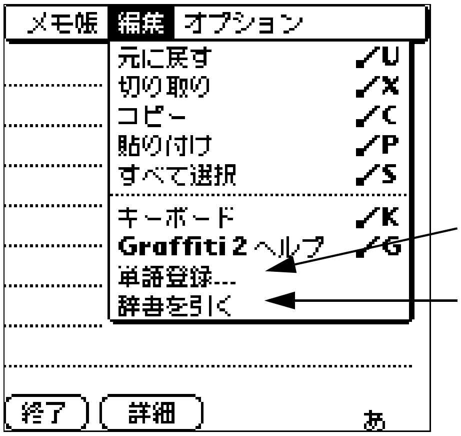
The Edit Menu items

The default Japanese Palm OS automatically adds two new menu items to any Edit menu that ends with the Graffiti 2 Help item. These items call special dictionaries.

Typical Japanese Palm Powered handhelds include a built-in Japanese-English dictionary for the user's reference. In addition, the FEP supports a User Dictionary. This User Dictionary is usually empty when the user first boots the handheld. The user can then add words and their converted forms to the User Dictionary to make conversion more automatic.

Figure 2.5 illustrates the Edit menu items on a typical Japanese handheld.

Figure 2.5 Japanese Palm OS Edit Menu Commands



Add word to User Dictionary

View Japanese-English Dictionary (only works if this dictionary is installed)

User Dictionary Preferences Panel

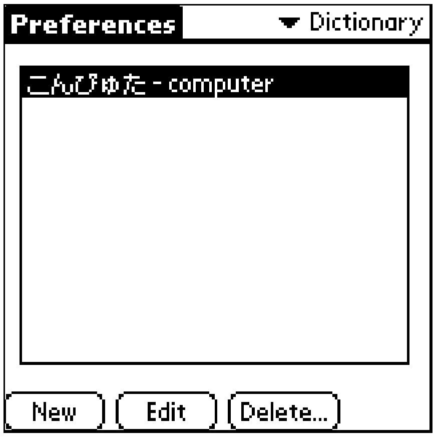
The User Dictionary Preferences Panel lets users add words to the User Dictionary and maintain the list of added words.

You can access the User Dictionary Preferences Panel, which contains the list of added words, only from the Preferences application. It is usually listed as “Dictionary” or the corresponding word in the language used on the handheld.

The following figure shows the User Dictionary Preferences Panel from a handheld running the Sample FEP included in the Sample FEP Kit[1]. The user interface is in English by default.

[1] FEP developers can request the Sample FEP by contacting PalmSource at devinfo@corp.palm.com.

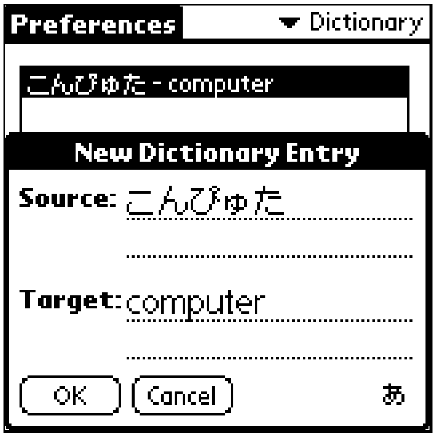
Figure 2.6 Sample FEP User Dictionary Preferences Panel



こんぴゅた - computer

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

Figure 2.7 Sample FEP User Dictionary New Entry Dialog



こんぴゅた - computer

Source: こんぴゅた

Target: computer

3

Creating an FEP Shared Library

Palm provides a Sample FEP shared library to help you get started with creating your own FEP. This Sample FEP Kit also includes a test application that will help you debug and test your FEP. FEP developers can request the Sample FEP by contacting PalmSource at devinfo@corp.palm.com.

This chapter discusses the following topics:

- The Sample FEP
- Event Flow in an FEP
- FEP Type and Creator ID
- Modifying the Sample FEP
- Debugging the FEP
- Testing the FEP

The Sample FEP

The Sample FEP shipped with this document is a modified version of the Japanese FEP shipped with Palm OS® 4.0. Palm provides this sample code as a starting place for creating your own FEP for Japanese or another language. The following sections describe the basic structure of the Sample FEP project.

Sample FEP File List

The files in the Sample FEP CodeWarrior project (`SampleFEP.mcp`) are described in the following table.

Table 3.1 Sample FEP File List

File Name	Description
Directory: \Incs	
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: \PalmOS Private	
AttentionPrv.h	Private Palm OS header file required for the definition of the `AttnEffectOfEvent` function.
TextServicesPrv.h	Private Palm OS header file that contains most of the defines, data structures, and function declarations required to implement the FEP API. See Appendix A for the contents of this header file.
Directory: \Rsc	
SampleFEP.rsrc	Mac OS resource file that contains the version (tver=1000) and name (tAIN=1000) resources for the shared library.

Table 3.1 Sample FEP File List *(continued)*

File Name	Description
SampleFepDict.rsrc	Mac OS resource file that contains most of the FEPI dictionary resources used by the sample FEP's conversion engine. The file `/Src/SampleFepOkuri.r` contains text-based definitions for two of the FEP dictionary resources (FEPI=1001 and 1002). **Note:** The data in `SampleFepDict.rsrc` is specific to the sample FEP's implementation of a conversion engine. This data is **not** intended to be used in any FEP that you create.
Directory: \Src	
PalmSL.cpp	Source file that contains run-time support for a multi-segment shared library. This file **must** be linked first.
PalmSL.h	Header file for `PalmSL.cpp`. This file defines the special macros and functions that a multi-segment shared library needs.
Prefix.h	Header file that is automatically included before each source file is compiled. Note that this inclusion must be explicitly specified in the project's C/C++ preference settings.
SampleFep.cpp	Source code for Sample FEP top-level functions.
SampleFepDispatch.cpp	Source code for the Sample FEP shared library dispatch table.
SampleFepEngine.cpp	Source code for Sample FEP low-level conversion functions.
SampleFepEngine.h	Header file for the Sample FEP conversion engine.
SampleFepEvents.cpp	Source code for Sample FEP event-handling functions.

Table 3.1 Sample FEP File List *(continued)*

File Name	Description
SampleFepEvents.h	Header file for the Sample FEP event handling functions.
SampleFepOkuri.r	Source file that contains some dictionary resource data (FEPI = 1001 & 1002) in text format. This file demonstrates an alternative method of including resources in the shared library, instead of using Mac OS resource files (which are sometimes difficult to work with on a Windows computer). **Note:** The data in `SampleFepOkuri.r` is specific to the sample FEP's implementation of a conversion engine. This data is **not** intended to be used in any FEP that you create.
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.
SampleFepUserDictAccess.cpp	Source file that contains functions used by the User Dictionary Preferences panel to access the FEP's user dictionary data. These functions implement the custom FEP APIs that are declared in `/Incs/SampleFep.h` (for example, `SampleFepUserDictOpen`).
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.

The TestSampleFep Project

The TestSampleFep CodeWarrior project is also included in the Sample FEP Kit. This project creates an application that lets you switch between the default FEP and your new FEP for testing purposes. You probably do not need to modify this project. The sections on "Debugging the FEP" on page 25 and "Testing the FEP" on page 27 describe this project in more detail.

Sample FEP Targets

The Sample FEP CodeWarrior project has three targets, which are described in the following sections.

SampleFep

The SampleFep target builds the actual shared library PRC.

SampleFepSim

The SampleFepSim target builds a Macintosh OS library that can be linked into the TestSampleFep project for the Palm Simulator target. When you install this library, the SampleFep replaces the default FEP. It allows you to do easy source-level debugging (just like any other Palm Simulator project).

For more information about the TestSampleFep project, see "Debugging the FEP" on page 25.

NOTE: The Palm Simulator is supported only for the Macintosh OS. For more information about using the Simulator, see the *Palm OS Programming Development Tools Guide*.

SampleFepSimResources

The SampleFepSimResources target builds the `SampleFepSim.rsrc` file, which contains the dictionary data resources. This file is used by the TestSampleFep project, and is included through the `AppResourceList` file list found in the `/TestSampleFep/Src/TestSampleFepRsc.c` file.

FEP Code Structure

The FEP code is structured into three layers:

- `SampleFep.cpp` implements all of the top-level FEP functions described in Chapter 6, "The Front-End Processor (FEP) API."
- 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.

Event Flow in an FEP

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

Boot Sequence

At boot time, the following sequence of calls is used to load the correct FEP:

1. When the handheld is reset, `TsmInit` calls `TsmGetSystemFepCreator` (if the Palm OS 5 feature set is not present, it calls `TsmGetSystemFep`) to determine the name of the FEP. It loads this FEP using `SysLibLoad`, and then calls `TsmLibGetFepInfo` 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 how the FEP version number, read about the `TsmFepInfoType` data structure.
2. If the FEP should be loaded, then `TsmInit` calls `TsmLibFepOpen`. If that call succeeds, then the FEP is set as the current FEP.

System Events

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

- Whenever SysHandleEvent is called, it calls TsmHandleEvent. If the event is a Text Services Manager virtual keyDownEvent (for example, one of the four Text Services Manager silkscreen 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 TsmFepHandleEvent (if the Palm OS 5 feature set is not present, it calls the current FEP’s TsmLibFepHandleEvent function directly). 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 TsmFepHandleEvent (if the Palm OS 5 feature set is not present, it calls the current FEP’s TsmLibFepHandleEvent function). 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 instructions that tell 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 `TsmFepCommitAction` (if the Palm OS 5 feature set is not present, 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), `TsmFepTerminate` is called (if the Palm OS 5 feature set is not present, it calls `TsmLibFepTerminate`). 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 `TsmFepReset` to put the FEP into a known, safe state (if the Palm OS 5 feature set is not present, it calls `TsmLibFepReset`). 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 posts 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

Prior to Palm OS 5, the FEP creator ID could be anything—although `sysFileCTextServices` (`'tsml'`) was suggested. Similarly, the type could be anything and the name could be anything, with the one caveat that the name in the PRC header had to match the library name.

Starting with Palm OS 5, the creator ID has to be unique, and should be assigned like any other creator ID by registering at http://www.palmos.com/dev/creatorid/. The FEP's type is required to be sysFileTFep('libt').

The name can still be anything you like. However, to make it easier to create a version of an FEP that works with both Palm OS 4.x and Palm OS 5, set the type and creator to what's needed for Palm OS 5 and then set the name to the 4-character creator ID.

Note that on Palm OS 5, SysLibFind will fail for ARM libraries, such as the native Palm FEP. The developer should then check if the name returned by TsmGetCurrentFep is four characters long. If so, then they need to call SysLibLoad with 'libt' as the type and the returned name as the creator.

In the native ARM environment, be aware that TsmGetCurrentFep, TsmGetSystemFep, TsmSetCurrentFep, and TsmSetSystemFep are deprecated; instead you identify the FEP by its creator ID and use the new functions TsmGetCurrentFepCreator, TsmGetSystemFepCreator, TsmSetCurrentFepCreator, and TsmSetSystemFepCreator. These last four functions are not implemented in PACE.

Modifying the Sample FEP

FEPs can be difficult to write for several reasons. FEPs are shared libraries, which are hard to debug, are limited in code size, and can have problems with globals if you aren't careful. If possible, use the Palm Simulator to debug the FEP because the Simulator provides symbolic debugging support.

Creating Multi-Segment FEPs

The Sample FEP is a multi-segment shared library. You might need to use multiple segments if your FEP exceeds the 64 kB limit for code in a single-segment Palm OS shared library. In the case of the Sample FEP, the FEP engine has been moved to a second segment, while the rest of the FEP remains in the first segment.

One limitation of this approach is that you can't debug the FEP on a handheld, or in POSE, using CodeWarrior. You can use the low-level debugger (PalmDebugger), however.

Handling Global Data

Normally, Palm OS shared libraries cannot use global variables. However, `/SampleFep/Src/PalmSL.cpp` contains code that lets a shared library use globals by making them a4-relative instead of a5-relative. This work around makes porting much easier, but makes the a4 register unavailable for other uses.

For more information about shared libraries and global variables, search the Palm Developer Knowledge Base (http://www.palmos.com/dev/support/kb/).

Changing the Locale

The Sample FEP is designed for the Japanese language. 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. In the CodeWarrior project, 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 four special Text Services Manager buttons in the silkscreen area. The four buttons correspond to "Convert," "Confirm," "Change Mode," and "On/Off." Tapping one of these buttons generates a `keyDownEvent` with one of the following characters: `vchrTsm1`, `vchrTsm2`, `vchrTsm3`, and `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 an 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 SampleFep code for examples 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 Shift Indicator

The FEP is given a chance to draw its own mode indicator in the space normally occupied by the Graffiti® 2 shift indicator. If the FEP is off, then the regular shift 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 Palm 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 User Dictionary Panel. Then, when the user enters that name, the corresponding yomi text will always appear in the yomi text field. For more information about the User Dictionary Panel, see Chapter 4, "Creating a User Dictionary Panel."

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 (typically Hiragana characters). This expanded text will later get converted into fewer bytes of Kanji and Hiragana.

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 User Dictionary Preferences panel. 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 to User Dictionary" command from the Edit menu, the `userDictLaunchCmdRegister` command launches the User Dictionary Preferences Panel with the creator `'udic'`. Although the FEP can use any registered creator ID for its Preferences panel, using `'udic'` takes advantage of the current Japanese Palm OS support for the "Add Word to User Dictionary" command in the Edit menu.

See the `UserDictRegisterParamsType` structure in `/SampleUserDict/Incs/SampleUserDict.h` for the structure that Palm OS passes to the Preferences panel with the `userDictLaunchCmdRegister` launch code. This structure contains information about the new word to be added to the user dictionary.

Debugging the FEP

The easiest way to debug the FEP is to use the Palm Simulator. Some applications, such as Memo Pad, have extra Japanese build targets to include components such as the Shift-JIS locale. Replace the Text Services project in the Memo Pad project with your own Text Services project, and replace the `TextServices.lib` Simulator library with the corresponding library created by your FEP project.

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

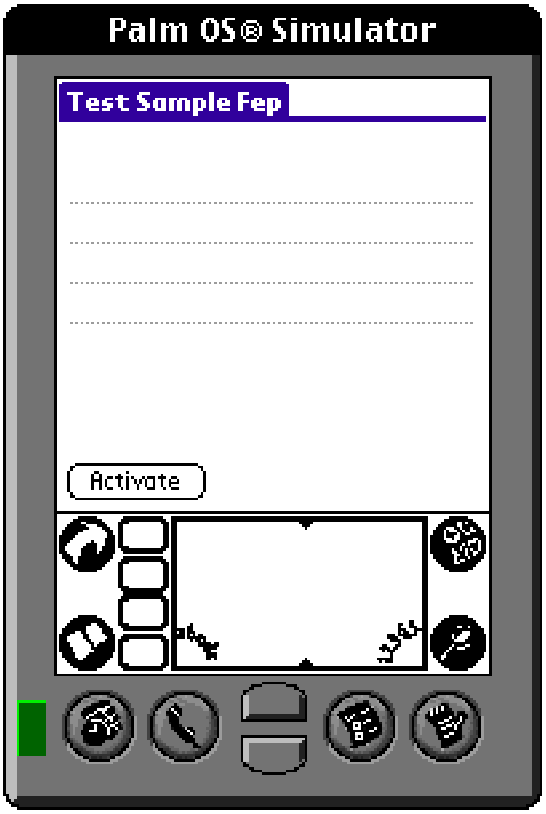
Debugging the FEP on a Macintosh

The following instructions describe how to debug the FEP using the Palm Simulator:

1. Open the TestSampleFep.mcp project, and ensure that the correct FEP files have been linked to the project:
   a. The FEP CodeWarrior project file should be listed under the Sub-projects category.
   b. The built Simulator target for the FEP CodeWarrior project (for example, `SampleFepSim.lib`) should be listed under the Palm OS Simulator category.
   c. The built Simulator target of the appropriate locale module (for example, `ShiftJISLocale(Sim).lib`) should be listed under the Palm OS Simulator category. It should appear after the FEP Simulator target and before any other Simulator files.
2. Select the TestSampleFepSim target.
3. Ensure that the CodeWarrior Debugger is enabled.

4. Build the project. This will build both the TestSampleFep and FEP projects. The Simulator window will appear as shown in the following picture:

Figure 3.1 Simulator Running Test Sample FEP Application



5. Use the Simulator's debugging tools to test the FEP.

For more information about the Palm Simulator, see the *Palm OS Programming Development Tools Guide*.

Debugging the FEP on a PC

If your FEP shared library uses only a single segment, you can debug it on a PC using the standard tools: the CodeWarrior Debugger, Palm Debugger, and Palm OS Emulator. Palm OS Emulator on the PC is capable of debugging single-segment shared libraries.

To test multiple-segment shared libraries, you must use the Palm Simulator, which is only available on the Macintosh platform.

Testing the FEP

The following procedure describes how to test the FEP using the TestSampleFep application on a handheld or in Palm OS Emulator.

1. Open the CodeWarrior project for the FEP shared library, select the standard target (for example, SampleFep), and build it.
2. Copy the resulting PRC to a handheld with the appropriate language enabled, or copy it to an Emulator session that is using a ROM with the appropriate language enabled.

NOTE: To run the Sample FEP, the handheld or Emulator session must use a Japanese Palm OS 4.0 ROM; earlier versions are not compatible with the Sample FEP.

3. Open the `TestSampleFep.mcp` project, select the “TestSampleFep” target, and build it.
4. Copy the resulting `TestSampleFep.prc` to your handheld or Palm OS Emulator session.
5. Launch the TestSampleFep application.

   The only button in the main form should say “Activate.” Tap this 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.
6. To deactivate your FEP, launch the TestSampleFep application. The main form button should say “Deactivate.” Tap this button to make the system FEP the current FEP. You can now safely delete or update your sample FEP.

4

Creating a User Dictionary Panel

The Sample FEP Kit[1] also includes a sample User Dictionary Preferences Panel. This panel lets the user add words and their matching conversions to the FEP user dictionary. These customized entries make text conversion easier and more automatic.

This chapter discusses the following topics:

- The Sample User Dictionary Panel
- Modifying the Sample User Dictionary Panel
- Debugging the User Dictionary Panel
- Testing the User Dictionary Panel

[1] FEP developers can request the Sample FEP by contacting PalmSource at devinfo@corp.palm.com.

The Sample User Dictionary Panel

The following sections describe the basic structure of the Sample User Dictionary Preferences panel project. The User Dictionary Preferences panel provides end-user functionality for viewing, deleting, and editing entries found in the Sample FEP's user dictionary. It also lets the user add entries to the dictionary.

Sample User Dictionary Panel File List

The files in the SampleUserDict CodeWarrior project (`SampleUserDict.mcp`) are described in the following table.

Table 4.1 Sample FEP File List

File Name	Description
Directory: \Incs	
Locale_jpJP.h	The Japanese locale prefix file. Must be included if the project uses Japanese resources. Otherwise, include the appropriate locale prefix file for your target locale. The U.S. English (enUS) locale does not require a prefix.
SampleUserDict.h	Header file for the Sample User Dictionary panel, which defines a custom launch code and parameter block for adding a new word to the user dictionary.
Directory: \Rsc	
SampleUserDictEdit.rsrc	Mac OS resource file that defines the "Edit" function of the User Dictionary panel.
SampleUserDictList.rsrc	Mac OS resource file that defines the "List" function of the User Dictionary panel.
SampleUserDictMisc.rsrc	Mac OS resource file that defines all alerts for the User Dictionary panel.
SampleUserDictPanel.rsrc	Mac OS resource file that contains the version (tver=1000) and name (tAIN=1000) resources for the panel.

Table 4.1 Sample FEP File List *(continued)*

File Name	Description
Directory: \Src	
SampleUserDict.c	Source code for the User Dictionary panel.
SampleUserDictRsc.c	Resource files definition list (for use by the Palm Simulator). Used only when compiling the Simulator target.

Sample User Dictionary Panel Targets

The CodeWarrior project for the Sample User Dictionary Panel has three targets, which are described in the following sections.

SampleUserDict

The SampleUserDict target builds the actual Preferences panel PRC.

SampleUserDictSim

The SampleUserDictSim target builds a Palm Simulator application. The Simulator lets you do easy source-level debugging.

NOTE: The Palm Simulator is supported only for the Macintosh OS. For more information about using the Simulator, see the *Palm OS Programming Development Tools Guide*.

SampleUserDictResources

The SampleUserDictResources target builds the `SampleUserDictMerge.rsrc` file, which contains the panel resources. This file is used by the SampleUserDictSim project target, and is included in the `AppResourceList` file list found in the `/SampleUserDict/Src/SampleUserDictRsc.c` file

Modifying the Sample User Dictionary Panel

The following sections describe how to modify the Sample User Dictionary Panel to create your own panel.

The Sample User Dictionary Panel is specific to the Palm-default Japanese FEP. It lets the user view, edit, and delete entries in the default Japanese FEP user dictionary. It does this by making special calls to the FEP.

Assigning Creator and Database Types

The PRC database type for the user dictionary panel must be "panl."

The creator can be any properly registered value. However, if it is set to "udic," then your user dictionary panel will override the default FEP panel as long as it has a higher version number in the DB header.

The only reason you might want to do this is to automatically take advantage of the current Japanese OS support for the "Add Word To User Dictionary" command. This command ("Add Word") is automatically added to every system Edit menu. When the user selects this menu command, the panel with creator "udic" is launched with the `userDictLaunchCmdRegister` command. Look at the `UserDictRegisterParamsType` structure in the `SampleUserDict` project for an example. This structure contains pointers to the new FEP user dictionary word and length.

Modifying the Panel Resource Files

The Sample User Dictionary Panel project contains four Macintosh OS resource files. These files are described in the section, "Sample User Dictionary Panel File List" on page 30.

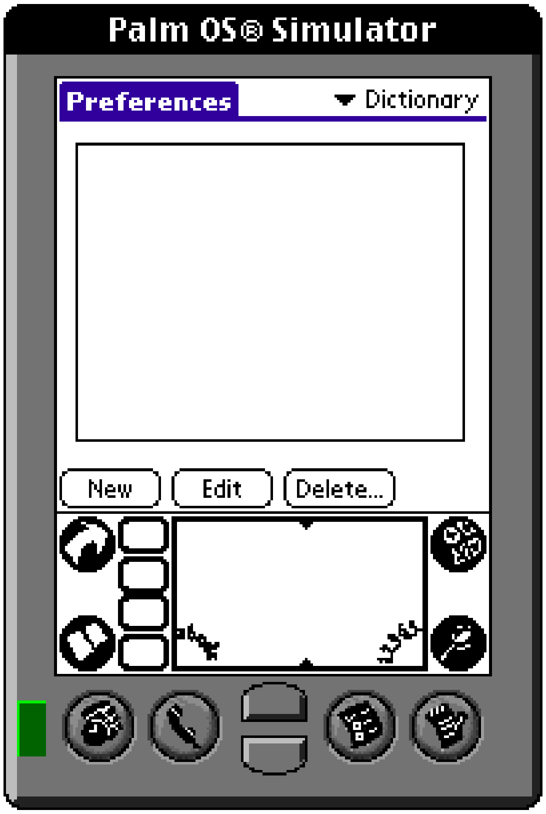
To edit these files, open them using the latest version of Constructor for the Palm OS®. The text in the Sample User Dictionary Panel forms is in English so that you can translate it into any language you choose.

Debugging the User Dictionary Panel

The following instructions describe how to debug the User Dictionary Panel using the Palm Simulator:

1. Open the SampleUserDict.mcp project, and ensure that the correct FEP files have been linked to the project:
   a. The FEP CodeWarrior project file should be listed under the Sub-projects category.
   b. The built Simulator target for the FEP CodeWarrior project (for example, `SampleFepSim.lib`) should be listed under the Palm OS Simulator category.
   c. The built Simulator target of the appropriate locale module (for example, `ShiftJISLocale(Sim).lib`) should be listed under the Palm OS Simulator category. It should appear after the FEP Simulator target and before any other Simulator files.
2. Select the User Dictionary Panel's Simulator target (for example, SampleUserDictSim).
3. Ensure that the CodeWarrior Debugger is enabled.
4. Build the project. This will build both the User Dictionary Panel and FEP projects. The Simulator window will appear. It will look similar to the example in the following picture:

Figure 4.1 Simulator Running Sample User Dictionary Panel



5. Use the Simulator's debugging tools to debug the User Dictionary Panel.

For more information about the Palm Simulator, see the *Palm OS Programming Development Tools Guide*.

Testing the User Dictionary Panel

The following procedure describes how to test the FEP using the Test Sample Fep application on a handheld or in Palm OS Emulator.

1. Open the CodeWarrior project for the FEP shared library, select the standard target (for example, SampleFep), and build it.
2. Copy the resulting PRC to a handheld with the appropriate language enabled, or copy it to an Emulator session that is using a ROM with the appropriate language enabled.

NOTE: To run the Sample FEP, the handheld or Emulator session must use a Japanese Palm OS 4.0 ROM; earlier versions are not compatible with the Sample FEP.

3. Open the `TestSampleFep.mcp` project.
4. Select the "TestSampleFep" target, and build it.
5. Copy the resulting `TestSampleFep.prc` to your handheld or Palm OS Emulator session.
6. Open the User Dictionary Panel project, select the standard target (for example, SampleUserDict), and build it.
7. Copy the resulting PRC to your handheld or Palm OS Emulator session.
8. Launch the TestSampleFep application.
9. Tap the Activate button to activate your FEP.
10. Add entries to the user dictionary and ensure that they get used during conversion.

    For example, on a Japanese handheld, you might add the word "konpyuta" in Hiragana and its English match, "computer." Type the word "konpyuta" in Memo Pad and ensure that the word "computer" appears when you tap the "Convert" button.

5

The Text Services Manager API

This chapter provides information about the public and private Text Services Manager APIs. The public functions are declared in `TextServicesMgr.h`, and the private functions are declared in `TextServicesPrv.h` (see Appendix A).

The Text Services Manager provides the caller with an API for interacting with various text services, including FEPs. The private functions are useful for managing multiple FEPs.

This chapter discusses the following topics:

- Public Text Services Manager Data Structures
- Public Text Services Manager Functions
- Private Text Services Manager Functions

Public Text Services Manager Data Structures

TsmFepModeType

The `TsmFepModeType` type specifies the input modes used by the functions `TsmGetFepMode` and `TsmSetFepMode`.

```
typedef UInt16 TsmFepModeType;
```

`TsmFepModeType` can be set to one of the defined modes shown in the following table.

Constant	Value	Description
`tsmFepModeDefault`	0	The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.
`tsmFepModeOff`	1	Indicates that there is no active FEP input mode (the FEP is off).
`tsmFepModeCustom`	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.

Public Text Services Manager Functions

TsmGetFepMode

Purpose Return the current input mode for the active front-end processor (FEP).

Prototype `TsmFepModeType TsmGetFepMode(void *nullParam)`

Parameters `-> nullParam` An unused status pointer that must be set to `NULL`.

Result 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 then restore the saved mode when the application has finished using a special text field.

Compatibility This function was added in Palm OS® 3.5. In 3.5, the `nullParam` parameter could take a non-`NULL` value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to `NULL`. Any other value will generate a non-fatal alert.

See Also `TsmSetFepMode`

TsmSetFepMode

Purpose Set the input mode for the active front-end processor (FEP) to be the mode defined by the parameter `inNewMode`.

Prototype

```
TsmFepModeType TsmSetFepMode(void *nullParam,
TsmFepModeType inNewMode)
```

Parameters

-> `nullParam`	An unused status pointer that must be set to `NULL`.
-> `inNewMode`	The new FEP input mode.

Result 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 the field will always be 7-bit ASCII, then 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, then you cannot use a numeric field, because the field code will prevent the user from entering this character (it's not a digit or a period). In that case, you should make it a regular field, and the application should screen the characters. In this situation, the application should disable the FEP when such a pseudo-numeric field is active.

IMPORTANT: A mode change is currently enqueued as a `keyDownEvent` so that the field and FEP can remain properly synchronized, and so that the mode change will not affect any pending `keyDownEvents`. Therefore the mode change does not happen until the enqueued `keyDownEvent` is passed to `FrmHandleEvent`; if you call `TsmGetFepMode` immediately after calling `TsmSetFepMode`, you will not see a mode change.

In Palm OS 4.0 there are also some limitations with changing the mode: there must be an active form; and if there is an active field in the form, it must not be a numeric field. These limitations were removed in Palm OS 4.1.

Compatibility This function was added in Palm OS 3.5. In 3.5, the `nullParam` parameter could take a non-`NULL` value, allowing the caller to maintain its own status record. In Palm OS 4.0, this parameter is unused and must be set to `NULL`. Any other value will generate a non-fatal alert.

See Also `TsmGetFepMode`, `EvtEnqueueKey`, `FrmHandleEvent`

Private Text Services Manager Functions

TsmDrawMode

Purpose Ask the Text Services Manager if a text service needs to draw the shift indicator to indicate the FEP's current input mode.

Prototype

```
Boolean TsmDrawMode(UInt16 state, Coord x,
Coord y)
```

Parameters

`-> state`	Shift indicator state.
`-> 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).

Result Returns `true` if a text service drew the mode indicator.

Compatibility Implemented only in Palm OS 4.0 and later.

See Also `TsmLibFepDrawModeIndicator`

New TsmFepCommitAction

Purpose Call through to the current FEP's `TsmLibFepCommitAction` function, which 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.

Prototype

```
Err TsmFepCommitAction
(TsmFepStatusType *ioStatusP)
```

Parameters

`<-> ioStatusP`	Status pointer for this context.

Result Returns `errNone` if the call was successful. Returns `tsmErrFepReentrancy` if the FEP is currently running code 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.

Compatibility Implemented only if the Palm OS 5 feature set is present.

New TsmFepHandleEvent

Purpose Call through to the current FEP's `TsmLibFepHandleEvent` function, which 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.

Prototype

```
Err TsmFepHandleEvent
(const SysEventType *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`	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.

Result 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.

Compatibility Implemented only if the Palm OS 5 feature set is present.

See Also `eventsEnum`, `SysEventType`

New

TsmFepMapEvent

Purpose Call through to the current FEP's `TsmLibFepMapEvent` function, which determines whether or not an event should be remapped by the FEP and, if the event needs to be remapped, posts the remapped event to the event queue.

Prototype

```
Boolean TsmFepMapEvent
(const TsmFepStatusType *inStatusP,
const SysEventType *inEventP, Boolean inProcess)
```

Parameters

`-> inStatusP` A pointer to the FEP status record.

`-> inEventP` A pointer to the event record.

`-> inProcess` A value of `true` indicates that the remapped event should be posted to the event queue.

Result Return `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).

Compatibility Implemented only if the Palm OS 5 feature set is present.

New TsmFepOptionsList

Purpose Pop up the list of options for the FEP.

Prototype

```
Int16 TsmFepOptionsList (UInt16 iNumOptions,
UInt16 iCurOption, UInt16 iListWidth,
UInt32 iCallerData,
FepOptionsDrawDataFuncType iDrawProc)
```

Parameters

`-> iNumOptions`	Number of options in list.
`-> iCurOption`	The currently selected item.
`-> iListWidth`	Width of the list in pixels.
`-> iCallerData`	User data passed to the list-drawing callback function.
`-> iDrawProc`	List-drawing callback function.

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

Comments In order to use this function you must supply a callback function that draws an item in the list. When `TsmFepOptionsList` is called, it will call your callback function for each item in the list supplying the item number, a set of boundaries in which you should draw, and a pointer to the user data block specified in the

TsmFepOptionsList call. Your callback function should have the following prototype:

```
typedef void FepOptionsDrawDataFuncType (Int16 itemNum,
const RectangleType * bounds, void *userData)
```

Compatibility Implemented only if the Palm OS 5 feature set is present.

New

TsmFepReset

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

Prototype `Err TsmFepReset (TsmFepStatusType *ioStatusP)`

Parameters `<-> ioStatusP` Status pointer for this context.

Result 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).

Compatibility Implemented only if the Palm OS 5 feature set is present.

New

TsmFepTerminate

Purpose Call 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.

Prototype

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

Parameters

`<-> ioStatusP`	Pointer to the FEP's status record.
`<- outActionP`	Pointer to the FEP's action record.

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

Compatibility Implemented only if the Palm OS 5 feature set is present.

TsmGetCurrentFep

Purpose Get the name of the current FEP.

Prototype

```
Boolean TsmGetCurrentFep(Char *oFepNameP)
```

Parameters

`<- oFepNameP`	Pointer to the current FEP name. If there is no current FEP, then this pointer contains the empty string ("").

Result Returns `true` if there is a current FEP. Stores the name of the current FEP in the `oFepNameP` parameter.

Comments The string buffer passed to `TsmGetCurrentFep` should be at least `dmDBNameLength` bytes long.

Compatibility Implemented only in Palm OS 4.0 and later. In Palm OS 5, in the ARM-native environment, this function is deprecated; use `TsmGetCurrentFepCreator` instead. If you call this function when the Palm OS 5 feature set is present, instead of receiving the FEP's name it returns the FEP creator as the "name."

Note that on Palm OS 5, `SysLibFind` will fail for ARM libraries such as the native Palm FEP. The developer should then check if the name returned by `TsmGetCurrentFep` is four characters long. If so, then they need to call `SysLibLoad` with `'libt'` as the type and the returned name as the creator.

See Also TsmSetCurrentFep

New

TsmGetCurrentFepCreator

Purpose Get the creator ID of the current FEP.

Prototype

```
Boolean TsmGetCurrentFepCreator
(UInt32 *oFepCreatorP)
```

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

Result Returns `true` if there is a current FEP.

Compatibility Implemented only if the Palm OS 5 feature set is present, and only in the native ARM environment.

See Also TsmGetCurrentFep

TsmGetSystemFep

Purpose Get the name 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.

Prototype

```
Boolean TsmGetSystemFep(Char *oFepNameP)
```

Parameters `<- oFepNameP` Pointer to the system FEP name. If there is no system FEP, then this pointer contains the empty string (“”).

Result Returns `true` if there is a system FEP. Stores the name of the system FEP in the `oFepNameP` parameter.

Comments The string buffer passed to TsmGetSystemFep should be at least `dmDBNameLength` bytes long.

Note that on Palm OS 5, `SysLibFind` will fail for ARM libraries such as the native Palm FEP. The developer should then check if the name returned by TsmGetCurrentFep is four characters long. If so, then they need to call `SysLibLoad` with `'libt'` as the type and the returned name as the creator.

Compatibility Implemented only in Palm OS 4.0 and later. In Palm OS 5, in the ARM-native environment, this function is deprecated; use TsmGetSystemFepCreator instead. If you call this function when the Palm OS 5 feature set is present, instead of receiving the FEP's name it returns the FEP creator as the "name."

See Also TsmSetSystemFep

New TsmGetSystemFepCreator

Purpose Get 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.

Prototype

```
Boolean TsmGetSystemFepCreator
(UInt32 *oFepCreatorP)
```

Parameters `<- oFepCreatorP`

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

Result Returns `true` if there is a system FEP.

Compatibility Implemented if the Palm OS 5 feature set is present, and only in the native ARM environment.

See Also TsmGetSystemFep, TsmSetSystemFepCreator

TsmHandleEvent

Purpose Ask the Text Services Manager if a text service (such as an FEP) wants to handle the event before the application handles it.

Prototype

```
Boolean
TsmHandleEvent(const SysEventType *inEventP,
Boolean inProcess)
```

Parameters

`-> inEventP`	A pointer to an event record.
`-> inProcess`	A value of `true` indicates that the event should be handled. A value of `false` indicates that no event processing is done, but the return result is still valid.

Result Returns `true` if the event was handled.

See Also `FldHandleEvent`, `SysHandleEvent`, `TsmLibFepHandleEvent`

TsmInit

Purpose Initialize the Text Services Manager. Locate and load available text services (such as an FEP).

Prototype `void TsmInit(void)`

Parameters None.

Result Returns nothing.

Compatibility Implemented only in Palm OS 4.0 and later.

See Also `SysLibLoad`, `TsmGetSystemFep`, `TsmLibFepOpen`

TsmSetCurrentFep

Purpose Set the name of the current FEP, and make it active.

Prototype `Err TsmSetCurrentFep(const Char *iFepNameP)`

Parameters `-> iFepNameP` Pointer to the current FEP name.

Result Returns `errNone` if the current FEP was changed to the FEP with the name that was passed in `iFepNameP`. Otherwise, it returns one of the following Palm OS 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.

Compatibility Implemented only in Palm OS 4.0 and later. In Palm OS 5, in the ARM-native environment, this function is deprecated; use `TsmSetCurrentFepCreator` instead. If you call this function when the Palm OS 5 feature set is present, it checks the name string to see if it is a four-character string. If it is, `TsmSetCurrentFep` calls `TsmSetCurrentFepCreator` using the passed name as the creator ID.

See Also `TsmGetCurrentFep`

New TsmSetCurrentFepCreator

Purpose Set the current FEP to be the FEP with the specified creator ID, opening it up and making it usable.

Prototype `Err TsmSetCurrentFepCreator(UInt32 iFepCreator)`

Parameters `-> iFepCreator` FEP plugin library creator ID.

Result Returns `errNone` if the current FEP was changed to the FEP with the specified creator ID. Otherwise, it returns one of the following Palm OS 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`.

Compatibility Implemented only if the Palm OS 5 feature set is present, and only in the native ARM environment.

See Also `TsmGetCurrentFepCreator`, `TsmSetCurrentFep`

TsmSetSystemFep

Purpose Set the name of the system FEP.

Prototype `void TsmSetSystemFep(const Char *iFepNameP)`

Parameters `-> iFepNameP` Pointer to the new system FEP name.

Result Returns nothing.

Compatibility Implemented only in Palm OS 4.0 and later. In Palm OS 5, in the ARM-native environment, this function is deprecated; use `TsmSetSystemFepCreator` instead. If you call this function when the Palm OS 5 feature set is present, it checks the name string to see if it is a four-character string. If it is, `TsmSetSystemFep` calls `TsmSetSystemFepCreator` using the passed name as the creator ID.

See Also `TsmGetSystemFep`

New

TsmSetSystemFepCreator

Purpose Set the creator ID of the system FEP.

Prototype `void TsmSetSystemFepCreator (UInt32 iFepCreator)`

Parameters `-> iFepCreator` FEP plugin library creator ID.

Result Returns nothing.

Compatibility Implemented only if the Palm OS 5 feature set is present, and only in the native ARM environment.

See Also `TsmGetSystemFepCreator`, `TsmSetSystemFep`

6

The Front-End Processor (FEP) API

This chapter provides information about the FEP API as declared in TextServicesPrv.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 are structures defined in the header files Event.h and SysEvent.h.

This chapter discusses the following topics:

- Related Palm OS Data Structures
- FEP Events
- FEP Data Structures
- FEP Functions

Related Palm OS Data Structures

eventsEnum

The `eventsEnum` enum defines all of the possible UI event types. Only the three relevant types are listed in this section. The other event types are described in "Part 1: User Interface" of the *Palm OS Programmer's API Reference*.

```
typedef enum {
  ...
  tsmConfirmEvent = 35,
  tsmFepButtonEvent,
  tsmFepModeEvent,
  ...
} eventsEnum;
```

These event types are discussed in "FEP Events" on page 55.

SysEventType

The `SysEventType` structure contains all the data associated with a system event. All event types have some common data. Most events also have data specific to those events. The specific data uses a union that is part of the `SysEventType` data structure. The union can have up to eight words of specific data.

This structure's common data is documented in the description of the `EventType` structure in "Part 1: User Interface" of the *Palm OS Programmer's API Reference*. The section "FEP Events" gives details on the important data associated with each type of Text Services Manager event.

```
typedef struct SysEventType {
   SysEventsEnum  eType;
   Boolean        penDown;
   UInt8          tapCount;
   Coord          screenX;
   Coord          screenY;
   union {
      ...
```

```
            struct _TSMConfirmType        tsmConfirm;
            struct _TSMFepButtonType    tsmFepButton;
            struct _TSMFepModeEventType   tsmFepMode;
            } data;
      } SysEventType;
```

FEP Events

tsmConfirmEvent

The tsmConfirmEvent should be generated by an FEP when converted text has been confirmed, either explicitly by the user or as a result of the field losing the focus. 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.

For this event, the data field contains the following structure:

```
struct _TSMConfirmType {
  Char         *yomiText;
  UInt16         formID;
};
```

Field Descriptions

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.

tsmFepButtonEvent

Tapping on a Text Services Manager silkscreen 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 silkscreen buttons, `TsmHandleEvent` remaps the event to be a `tsmFepButtonEvent`.

For this event, the `data` field contains the following structure:

```
struct _TSMFepButtonType {
  UInt16       buttonID;
};
```

Field Descriptions

`buttonID`

The `buttonID` field can have the following values in the default Palm OS Japanese FEP and in the Sample FEP. Some FEPs may not use all of these values:

- `tsmFepButtonConvert0`
- `tsmFepButtonConfirm1`
- `tsmFepButtonKana2`
- `tsmFepButtonOnOff3`
- `tsmFepButtonShorten4`
- `tsmFepButtonLengthen5`

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

tsmFepModeEvent

The `tsmFepModeEvent` is 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 via events to ensure proper FEP/field code synchronization.

For this event, the `data` field contains the following structure:

```
struct _TSMFepModeEventType {
  UInt16         mode;
};
```

Field Descriptions

The `mode` field can have the following values (defined in the `TsmFepModeType` type definition):

Constant	Value	Description
`tsmFepModeDefault`	0	The default input mode for the FEP. For example, with the Japanese FEP, the default mode is Hiragana.
`tsmFepModeOff`	1	Indicates that there is no active FEP input mode (the FEP is off).
`tsmFepModeCustom`	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.

FEP Data Structures

TsmFepInfoType

The `TsmFepInfoType` structure is passed to the `TsmLibFepHandleEvent` function. It contains extra information required by the FEP to correctly handle the event. The `TsmFepInfoType` structure is filled in by the `TsmLibGetFepInfo` function, which is usually called before the FEP is actually opened. Therefore, the FEP does not need to update any of the fields in this structure.

The structure is defined as follows:

```
typedef struct {
  UInt32       apiVersion;
  UInt32       libVersion;
  UInt32       libMaker;
```

```
   UInt16       encoding;
   UInt16       language;
   UInt16       stackExtra;
   UInt16       fieldExtra;
 } TsmFepInfoType;
```

Field Descriptions

apiVersion — This field should always be set to tsmAPIVersion, which is a constant defined in TextServicesPrv.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. If the Palm OS 5 feature set is present this should 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 PalmLocale.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 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

The `TsmFepStatusType` 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. It is defined as follows:

```
typedef struct {
  UInt16       refnum;
  Char*        inlineText;
  UInt16       convertedLen;
  UInt16       pendingLen;
  UInt16       selectStart;
  UInt16       selectEnd;
  UInt16       clauseStart;
  UInt16       clauseEnd;
} TsmFepStatusType;
```

Field Descriptions

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

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

`convertedLen` The amount of text (in bytes) at the beginning of the `inlineText` data that has been converted.

`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.

TsmFepEventType

The `TsmFepEventType` 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. The structure is defined as follows:

```
typedef struct {
  Int16        penOffset;
  Boolean      penLeading;
  Boolean      formEvent;
  UInt16       maxInline;
  Char*        primeText;
  UInt16       primeOffset;
  UInt16       primeLen;
} TsmFepEventType;
```

Field Descriptions

`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.

`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 then 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.

TsmFepActionType

The FEP functions `TsmLibFepHandleEvent` and `TsmLibFepTerminate` use the `TsmFepActionType` structure to tell the caller what needs to be done to update the text display to synchronize it with the FEP. The structure is defined as follows:

```
typedef struct {
  UInt16       tsmRefnum;
  UInt16       dumpLength;
  UInt16       primedLength;
  Boolean      updateText;
```

```
    Boolean      updateSelection;
    Boolean      updateFepMode;
    Boolean      handledEvent;
  } TsmFepActionType;
```

Field Descriptions

tsmRefnum — A value set by the Text Services Manager to identify the FEP. It is not the same as the FEP's reference number. Currently this value is unused, but in the future it will be filled in by the Text Services Manager before calling the FEP.

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.

updateFepMode — This field is true whenever the FEP mode has changed, indicating that the shift indicator has to be redrawn.

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.

FEP Functions

TsmLibFepOpen

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

Prototype

```
Err TsmLibFepOpen(UInt16 inRefnum,
TsmFepStatusType **outStatusP)
```

Parameters

`-> inRefnum`	Library reference number.
`<- outStatus`	Pointer to the new instance (status) record.

Result Return `errNone` if the call was successful; otherwise, return a standard Palm OS error code that indicates the nature of the problem, such as `memErrNotEnoughSpace` or `dmErrCantFind`.

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

See Also `TsmLibFepClose`

TsmLibFepClose

Purpose Deallocate a `TsmFepStatusType` structure previously returned by `TsmLibFepOpen`. If this closes the last active session, then get rid of any shared information (for example, unlock dictionary data).

Prototype

```
Err TsmLibFepClose(UInt16 inRefnum,
TsmFepStatusType *ioStatusP)
```

Parameters

`-> inRefnum`	Library reference number.
`<-> ioStatus`	Status pointer for this context.

Result Return one of the following:

`errNone`	No error; call succeeded.
`tsmErrFepNotOpen`	The FEP is not open.
`tsmErrFepReentrancy`	The FEP is currently running code.
`tsmErrFepStillOpen`	The FEP has additional contexts that are active.

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

See Also `TsmLibFepOpen`

TsmLibFepSleep

Purpose Put the FEP to sleep.

Prototype

```
Err TsmLibFepSleep(UInt16 inRefnum)
```

Parameters `-> inRefnum` Library reference number.

Result Return the result of the call. (This function typically does nothing and returns `errNone`.)

Comments Because no FEPs currently use any handheld hardware which draws power, this function is not currently used. However, all shared libraries are required to implement this call.

See Also `TsmLibFepWake`

TsmLibFepWake

Purpose Wake up the FEP.

Prototype `Err TsmLibFepWake(UInt16 inRefnum)`

Parameters `-> inRefnum` Library reference number.

Result Return the result of the call. (This function typically does nothing and returns `errNone`.)

Comments Because no FEPs currently use any handheld hardware which draws power, this function is not currently used. However, all shared libraries are required to implement this call.

See Also `TsmLibFepSleep`

TsmLibFepCommitAction

Purpose Unlock or deallocate any buffers used to pass information back to the caller in the `TsmFepStatusType` record as a result of a `TsmLibFepHandleEvent` or `TsmLibFepTerminate` call.

Prototype

```
Err TsmLibFepCommitAction(UInt16 inRefnum,
TsmFepStatusType *ioStatusP)
```

Parameters

`->inRefnum` Library reference number.

`<-> ioStatusP` Status pointer for this context.

Result Return `errNone` if the call was successful. Return `tsmErrFepReentrancy` if the FEP is currently running code 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.

See Also `TsmFepCommitAction`

TsmLibFepDrawModeIndicator

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

Prototype

```
Boolean
TsmLibFepDrawModeIndicator(UInt16 inRefnum,
const TsmFepStatusType *inStatusP, UInt16 state,
Int16 x, Int16 y)
```

Parameters

`-> inRefnum`	Library reference number.
`-> inStatusP`	A pointer to the FEP status record.
`-> state`	Shift indicator state.
`-> 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).

Result Return `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"

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.

TsmLibFepHandleEvent

Purpose Tell the caller whether or not the FEP completely handled the event. Update the TsmFepStatusType record as appropriate, and set fields in the TsmFepActionType record to tell the caller what needs to be updated.

Prototype

```
Err TsmLibFepHandleEvent(UInt16 inRefnum,
const SysEventType *inEvent,
const TsmFepEventType *inTsmEventP,
TsmFepStatusType *ioStatusP,
TsmFepActionType *outActionP)
```

Parameters

`-> inRefnum` Library reference number.

`-> inEvent` 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` 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.

Result 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 `eventsEnum`, `SysEventType`, `TsmFepHandleEvent`

TsmLibFepMapEvent

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

Prototype

```
Boolean TsmLibFepMapEvent(UInt16 inRefnum,
const TsmFepStatusType *inStatusP,
const SysEventType *inEvent, Boolean inProcess)
```

Parameters

`-> inRefnum`	Library reference number.
`-> inStatusP`	A pointer to the FEP status record.
`-> inEvent`	A pointer to the event record.
`-> inProcess`	A value of `true` indicates that the remapped event should be posted to the event queue.

Result Return `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).

See Also `TsmFepMapEvent`

TsmLibFepReset

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

Prototype

```
Err TsmLibFepReset(UInt16 inRefnum,
TsmFepStatusType *ioStatusP)
```

Parameters

`-> inRefnum`	Library reference number.
`<-> ioStatusP`	Status pointer for this context.

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

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

See Also `TsmFepReset`

TsmLibFepTerminate

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

Prototype

```
Err TsmLibFepTerminate(UInt16 inRefnum,
TsmFepStatusType *ioStatusP,
TsmFepActionType *outActionP)
```

Parameters

`-> inRefnum`	Library reference number.
`<-> ioStatusP`	Pointer to the FEP's status record.
`<- outActionP`	Pointer to the FEP's action record.

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

See Also `TsmFepTerminate`

TsmLibGetFepInfo

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

Prototype

```
Err TsmLibGetFepInfo(UInt16 inRefnum,
TsmFepInfoType *outInfoP)
```

Parameters

`-> inRefnum`	Library reference number.
`<- outInfoP`	Pointer to the information record to be filled in.

Result Return the result of 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.

TsmLibGetFepMode

Purpose Get the FEP's current input mode.

Prototype

```
TsmFepModeType TsmLibGetFepMode(UInt16 inRefnum,
const TsmFepStatusType *inStatusP)
```

Parameters

`-> inRefnum`	Library reference number.
`-> inStatusP`	A pointer to the FEP status record.

Result Return the current FEP mode. See the description of `TsmFepModeType` for the list of mode values.

Comments This mode corresponds to what the user sees in the shift indicator. The two standard modes are “off” and “default.” For Japanese FEPs, “default” should mean Hiragana. Other modes can be supported, and each can have a different range of values specific to each FEP implementation.

See Also `TsmGetFepMode`, `TsmSetFepMode`

A

TextServicesPrv.h

WARNING! This file is provided here for illustrative purposes only. It is normally considered “private” as 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) 1994-2000 Palm, Inc. or its subsidiaries.
 * All rights reserved.
 *
 * File: TextServicesPrv.h
 *
 * Release:
 *
 * Description:
 *        Private Header for the Text Services Manager
 *
 *
 ******************************************************************************/

#ifndef __TEXTSERVICESPRV_H__
#define __TEXTSERVICESPRV_H__

#include <PalmTypes.h>
#include <CoreTraps.h>
#include <LibTraps.h>
#include <SysEvent.h>
#include <SystemMgr.h>
#include <GraffitiShift.h>
#include <TextServicesMgr.h>

/************************************************************************
 * Private constants
 ***********************************************************************/

#define tsmAPIVersion (sysMakeROMVersion(4, 0, 0, sysROMStageRelease, 0))
```

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

// ID of TSM preference that contains the name of the system FEP.
#define tsmSystemFepPrefsID 1
#define tsmSystemFepPrefsVers  0

// 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 tsmFtrNumFepFieldExtra 129

// Tsm Fep library traps, for calling input methods.
enum {
    tsmLibTrapFepOpen = sysLibTrapOpen,
    tsmLibTrapFepClose = sysLibTrapClose,
    tsmLibTrapFepSleep = sysLibTrapSleep,
    tsmLibTrapFepWake = sysLibTrapWake,

    tsmLibTrapGetFepInfo = sysLibTrapCustom,
    tsmLibTrapFepHandleEvent,
    tsmLibTrapFepMapEvent,
    tsmLibTrapFepTerminate,
    tsmLibTrapFepReset,
    tsmLibTrapFepCommitAction,
    tsmLibTrapFepDrawModeIndicator,
    tsmLibTrapGetFepMode,

    tsmLibTrapFepReserved0,
    tsmLibTrapFepReserved1,
    tsmLibTrapFepReserved2,
    tsmLibTrapFepReserved3,
    tsmLibTrapFepReserved4,
    tsmLibTrapFepReserved5,
    tsmLibTrapFepReserved6,
    tsmLibTrapFepReserved7,
    tsmLibTrapFepReserved8,
    tsmLibTrapFepReserved9,
    tsmLibTrapFepCustom            // First custom Tsm Fep trap starts here.
};

// 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)

/***********************************************************************
 * Private types
 ***********************************************************************/

// Structure returned by TsmLibGetFepInfo routine.
typedef struct {
    UInt32 apiVersion;  // Tsm API implemented by library.
    UInt32 libVersion;  // Custom API implemented by library.
    UInt32 libMaker;  // Who made this input method?
    UInt16 encoding;  // e.g. encoding_CP1252
    UInt16 language;  // e.g. JAPANESE

    // New in 4.0
    UInt16 stackExtra;  // Extra stack space needed by FEP
    UInt16 fieldExtra;  // Extra field space needed by FEP.
} TsmFepInfoType;

// Structure passed to TsmFepHandleEvent routine.
typedef struct {
    Int16 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 maxInline; // Max allowable size of inline, in bytes.
    Char * primeText; // ptr to selected text (if inline not active)
    UInt16 primeOffset; // Offset to selected text.
    UInt16 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 {
    UInt16 refnum; // Refnum of FEP shared library.

    Char *inlineText; // ptr to inline text.

    UInt16 convertedLen; // Length of converted text.
    UInt16 pendingLen;  // Length of unconverted (pending) text.

    UInt16 selectStart; // Start of selection range.
    UInt16 selectEnd;  // End of selection range (can extend past
                       // end of inline text)

    UInt16 clauseStart; // Start of converted clause highlighting
    UInt16 clauseEnd;  // End of converted clause highlighting
} TsmFepStatusType;

// Structure returned by TsmLibFepHandleEvent/TsmLibFepTerminate routines
// and passed to the TsmLibFepCommitAction routine. 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 {
    UInt16 tsmRefnum;    // TSM ID for FEP (NOT a shared lib refnum)

    UInt16 dumpLength;    // Length of text to dump (or zero)
    UInt16 primedLength;   // Length of priming text used by FEP

    Boolean updateText;    // True -> update inline text.
    Boolean updateSelection;  // True -> update selection range.
    Boolean updateFepMode;   // True -> update Fep mode indicator.

    Boolean handledEvent;   // True -> Fep handled event.
} TsmFepActionType;

/********************************************************************
 * Private routines
 ********************************************************************/

#ifdef __cplusplus
extern "C" {
#endif

#if (EMULATION_LEVEL == EMULATION_NONE)
```

```
// This is the routine that gets called by the trap dispatcher for all
// of the TsmXXX routines, where the selector is in register D2.w.
void TsmDispatch(void);
#endif

#if (EMULATION_LEVEL != EMULATION_NONE)
// This is the routine that gets called when loading a FEP on the
// Simulator. TextServicesEmu.c has a dummy entry to prevent link
// errors, while the real routine is in the actual FEP code (and
// thus generates a link warning when a project file includes the
// FEP library in its Simulator target).
Err TsmLibPrvInstallDispatcher(UInt16 refNum, SysLibTblEntryPtr entryP);
#endif

// Initialize the Text Services Manager. Load available text services.
void TsmInit(void)
        TSM_TRAP(tsmInit);

// See if Text Services wants to handle the event.
Boolean TsmHandleEvent(const SysEventType* inEventP, Boolean inProcess)
        TSM_TRAP(tsmHandleEvent);

// Give text services a chance to draw the GSI as a mode indication.
Boolean TsmDrawMode(UInt16 state, Coord x, Coord y)
        TSM_TRAP(tsmDrawMode);

// Get the name of the system FEP.
Boolean TsmGetSystemFep(Char* oFepNameP)
        TSM_TRAP(tsmGetSystemFep);

// Set the name of the system FEP.
void TsmSetSystemFep(const Char* iFepNameP)
        TSM_TRAP(tsmSetSystemFep);

// Get the name of the current FEP.
Boolean TsmGetCurrentFep(Char* oFepNameP)
        TSM_TRAP(tsmGetCurrentFep);

// Set the name of the current FEP, and make it active.
Err TsmSetCurrentFep(const Char* iFepNameP)
        TSM_TRAP(tsmSetCurrentFep);

/***********************************************************************
 * FEP Shared Library routines
 ***********************************************************************/

// 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.
Err TsmLibFepOpen(UInt16 inRefnum, TsmFepStatusType** outStatusP)
    SYS_TRAP(tsmLibTrapFepOpen);

// Close down an instance of the Fep. The Fep is responsible
// for disposing of the TsmFepStatusType which it allocated in TsmLibFepOpen().
Err TsmLibFepClose(UInt16 inRefnum, TsmFepStatusType* ioStatusP)
    SYS_TRAP(tsmLibTrapFepClose);

// TsmLibFepSleep and TsmLibFepWake do nothing.
Err TsmLibFepSleep(UInt16 inRefnum)
    SYS_TRAP(tsmLibTrapFepSleep);

Err TsmLibFepWake(UInt16 inRefnum)
    SYS_TRAP(tsmLibTrapFepWake);

// Return information about the Fep in the TsmFepInfoType structure.
Err TsmLibGetFepInfo(UInt16 inRefnum, TsmFepInfoType* outInfoP)
    SYS_TRAP(tsmLibTrapGetFepInfo);

// 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).
Err TsmLibFepHandleEvent(UInt16 inRefnum,
    const SysEventType* inEventP,
    const TsmFepEventType* inTsmEventP,
    TsmFepStatusType* ioStatusP,
    TsmFepActionType* outActionP)
    SYS_TRAP(tsmLibTrapFepHandleEvent);

// Decide if <inEvent> should be remapped to some other event. If so, return
// true. If we return true, and <inProcess> is true, then go ahead and perform
// the remapping by posting a new event with the remapped info.
Boolean TsmLibFepMapEvent(UInt16 inRefnum,
    const TsmFepStatusType* inStatusP,
    const SysEventType* inEventP,
    Boolean inProcess)
    SYS_TRAP(tsmLibTrapFepMapEvent);

// 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.
Err TsmLibFepTerminate(UInt16 inRefnum, TsmFepStatusType* ioStatusP,
    TsmFepActionType* outActionP)
    SYS_TRAP(tsmLibTrapFepTerminate);
```

```
// 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.
Err TsmLibFepReset(UInt16 inRefnum, TsmFepStatusType* ioStatusP)
    SYS_TRAP(tsmLibTrapFepReset);

// 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>.
Err TsmLibFepCommitAction(UInt16 inRefnum, TsmFepStatusType* ioStatusP)
    SYS_TRAP(tsmLibTrapFepCommitAction);

// Draw the Fep mode indicator at location <x,y>.
Boolean TsmLibFepDrawModeIndicator(UInt16 inRefnum,
    const TsmFepStatusType* inStatusP,
    UInt16 state,
    Int16 x,
    Int16 y)
    SYS_TRAP(tsmLibTrapFepDrawModeIndicator);

// Get the Fep mode.
TsmFepModeType TsmLibGetFepMode(UInt16 inRefnum,
    const TsmFepStatusType* inStatusP)
    SYS_TRAP(tsmLibTrapGetFepMode);

// Standard declaration for unimplemented Tsm routines. They all return an Err,
// and their first parameter is a refnum (followed by zero..n additional
// parameters).
Err TsmLibReserved(UInt16 inRefnum);

#ifdef __cplusplus
}
#endif

#endif    // __TEXTSERVICESPRV_H__
```

Index

A

active input area 2
API
  FEP 53
  Text Services Manager 37
API version number 58
auto-yomi events 23, 55

B

boot sequence 18
buttons
  Change Mode 6
  Confirm 6
  Convert 6
  On/Off 6
  silkscreen 6, 56

C

Change Mode button 6
clause 6
Confirm button 6
confirmation 3
conversion, of text 2, 7, 29
Convert button 6

D

debugging the FEP 25, 26
debugging the User Dictionary panel 33

E

Edit menu 9
Emulator 26, 27
event flow 18
event handling 20
Event.h 53
events
  auto-yomi 23, 55
  types 54
eventsEnum 54
EventType 54

F

FEP
  code structure 18
  debugging on a PC 26
  debugging on Macintosh 25
  definition 2
  event flow 18
  multi-segment 21
  resetting 20
  size limit 21
  testing 27
FEP API 53
field
  extending size 24
  processing raw text 2
  with special text 39
field-level events 19
FldHandleEvent 19, 20
form 2, 55
front-end processor. *See* FEP 2

G

globals 21, 22

H

Hiragana 2

I

inline input. 2
input method 2
input mode 6, 38

K

Kanji 2
Katakana 2
keyDownEvent 19, 22, 23, 40

L

locale 22
Locale Manager 58

M

mode indicator 23, 67
multi-segment FEPs 21
multi-segment libraries
    debugging 27

O

On/Off button 6
options pop-up list 3, 8, 16

P

Palm OS Emulator 26, 27
Palm Simulator 17, 21, 25, 33
penDownEvent 61
priming text 6, 63

R

raw text 2
remap characters 44, 69
resetting the FEP 20
resource files
    FEP 14, 15
    User Dictionary panel 30
Romaji 2

S

Sample FEP
    code structure 18
    file list 14
    modifying 21
    support for User Dictionary 10
    targets 17
Sample FEP Kit 10, 14
Sample FEP structure 14
Sample User Dictionary Prefs Panel 29
SampleFep project 17
SampleFep.cpp 53
SampleFEP.mcp 14
SampleUserDict project 31
shared libraries 3, 21
shift indicator 23, 41, 67
silkscreen buttons 6, 56
Simulator 17, 21, 25, 33
space and linefeed characters 23
SysEvent.h 53
SysEventType 54
SysHandleEvent 22
sysInvalidRefNum 50, 51
SysLibLoad 18
system events 19

T

testing the FEP 27
testing the User Dictionary panel 34
TestSampleFep 25, 27
TestSampleFep project 17
text service 2
Text Services Manager 2
Text Services Manager API 37
TextServicesMgr.h 37
TextServicesPrv.h 37, 53
TSM 2
tsmConfirmEvent 20, 23, 55
TsmDrawMode 41
tsmErrFepNeedCommit 43, 68
tsmErrFepReentrancy 42, 43, 45, 46, 66, 68, 70, 71
tsmErrFepStillOpen 50, 51
tsmErrFepWrongAPI 50, 51
TsmFepActionType 62
tsmFepButtonEvent 19, 56
TsmFepCommitAction 41
TsmFepEventType 61
TsmFepHandleEvent 42
TsmFepInfoType 18, 57
TsmFepMapEvent 43
tsmFepModeCustom 38, 57
tsmFepModeDefault 38, 57
tsmFepModeEvent 19, 56
tsmFepModeOff 38, 39, 57
TsmFepModeType 38
TsmFepOptionsList 44
TsmFepReset 45
TsmFepStatusType 59
TsmFepTerminate 45
TsmGetCurrentFep 46
TsmGetCurrentFepCreator 47

TsmGetFepMode 38
TsmGetSystemFep 18, 47
TsmGetSystemFepCreator 48
TsmHandleEvent 19, 49
TsmInit 18
TsmLibFepClose 64
TsmLibFepCommitAction 20, 66
TsmLibFepDrawModeIndicator 67
TsmLibFepHandleEvent 19, 20, 23, 68
TsmLibFepMapEvent 19, 69
TsmLibFepOpen 18, 64
TsmLibFepReset 20, 70
TsmLibFepSleep 65
TsmLibFepTerminate 20, 23, 70
TsmLibFepWake 66
TsmLibGetFepInfo 18, 24, 71
TsmLibGetFepMode 71
TsmSetCurrentFep 50
TsmSetCurrentFepCreator 50
TsmSetFepMode 39
TsmSetSystemFep 51
TsmSetSystemFepCreator 52

U

User Dictionary
- functions 24
- interface 9

User Dictionary Preferences panel
- adding entries 24
- creator ID 24, 32
- file list 30
- interface 10
- sample project 29
- testing 34

V

virtual characters 22, 56

Y

yomi characters 23