OpenAdaptxt™

Document Title

KPTFramework - User's Guide

1. Introduction

1.1 Scope

This document provides an overview of the KPTFramework, its basic concepts and architecture, the functionality it provides, and how that functionality is accessed and managed. This document does not describe programming fundamentals such as data structures and types. For that information, and coding examples, the reader is referred to Ref 1.

1.2 References

Ref No.	Document Title	Applicable Version
1.	KPT Framework Reference Guide	Latest
2.	http://tools.ietf.org/rfc/bcp/bcp47.txt	Latest
3.	http://www.rfc-editor.org/rfc/rfc4647.txt	Latest

1.3 Change History

Document version	Date	Reason for change
V 1.0	09-May-2011	First version

1.4 Contents

OpenAdaptxt™

1.5 Tables

1.6 Figures

2. KPT Framework

2.1 Overview

The KPT Framework provides functionality that allows third-parties to develop text-entry applications.

The functionality provided by the Framework is contained in components that can be used or ignored as required. This ability allows developers to pick-and-choose the functionality they wish to provide in their applications.

Components may be added at run-time to extend an application's capabilities. These add-on components may provide extra functionality, or they may provide new or modified data. The concept of a component therefore includes data containers.

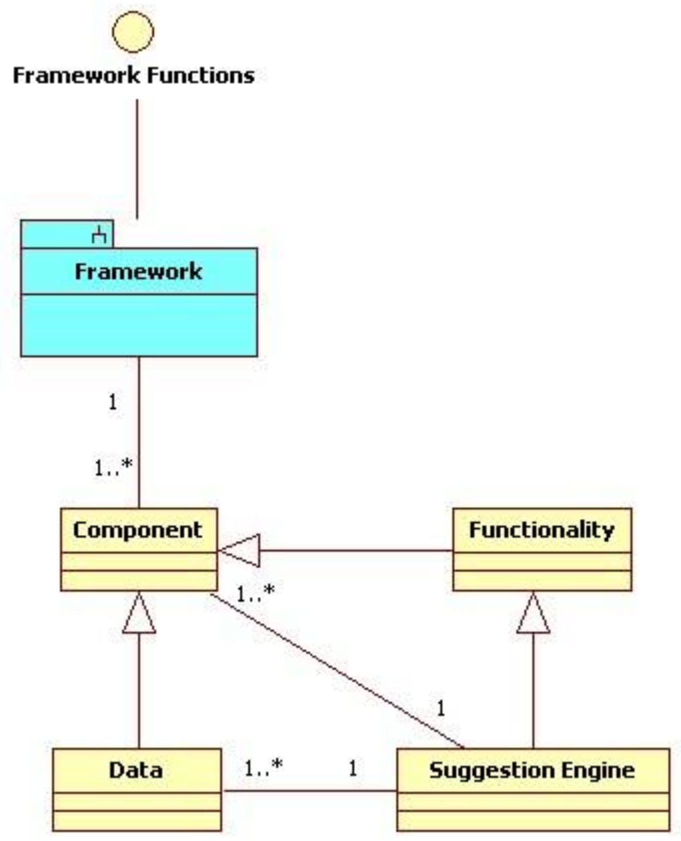


Figure 1. KPT Framework Architecture

A key component provided by the Framework is the Suggestion Engine. When used in conjunction with other components (see 3 below), the Suggestion Engine provides the ability to generate word suggestions in response to user input.

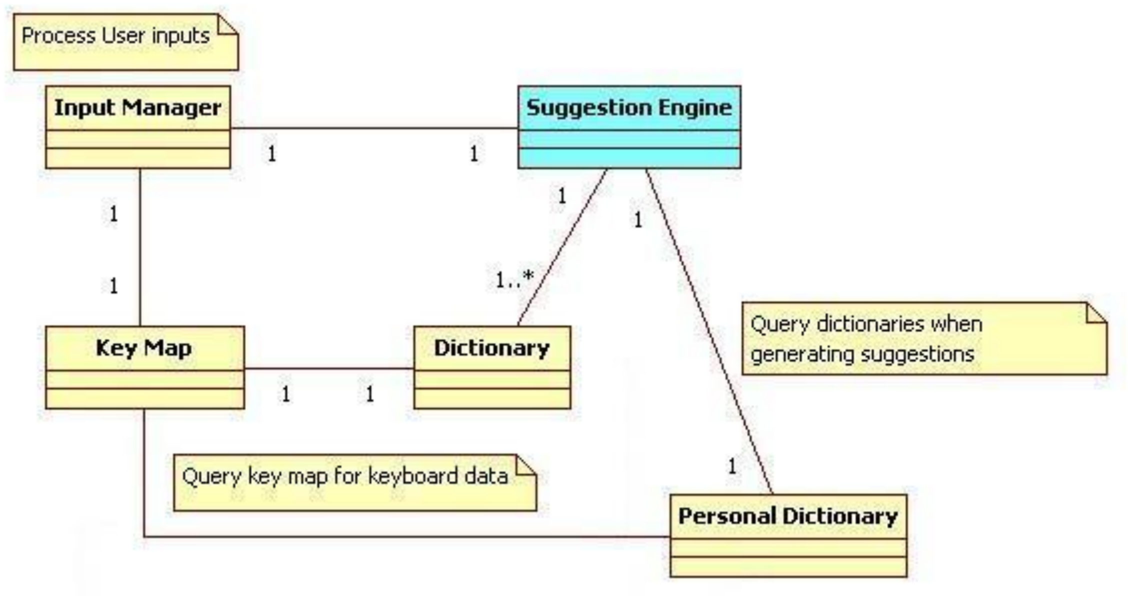


Figure 2. Suggestion Engine Associations

2.2 Basic Concepts

2.2.1 Suggestion Engine

In order for suggestions to be generated, the Framework's Input Manager is supplied with characters, and/or key strokes, as they are entered by a User. The input is processed by the Input Manager (see 5.99), after which suggested word completions can be requested from the Suggestion Engine (see 5.100). When a word has been completed, the Suggestion Engine can also suggest likely next words.

See Ref 1 for further information on the types of suggestions provided by the Suggestion Engine.

2.2.2 Input Manager

The Input Manager maintains a text input buffer. User-entered characters and/or keystrokes are inserted into the text input buffer for processing by the Input Manager. The text input buffer may also be updated and modified with words or sections of text. Commands for managing the text input buffer are described in 5.9.

2.2.3 Dictionaries

The Suggestion Engine makes use of dictionaries (see 5.45) when determining its suggestions. Dictionaries contain reference words for a language, either providing general coverage, or tailored for specific subjects; e.g. law, or medicine. Dictionaries also contain language intelligence, allowing the Framework and its components to handle each language in the correct way.

A number of dictionaries are provided with each Framework installation. Additional dictionaries can be installed using packages.

2.2.4 Packages

Packages (see 5.77) are objects containing one or more components, and are the mechanism through which components are added to, and removed from, the Framework.

2.2.5 Key Maps

The Framework can be configured to support text input from a wide variety of keyboard layouts. The required configuration data is supplied in key map files (see 5.8). A key map file provides the Framework with information on keyboard layouts. Each key map file may contain more than one layout, either because the target device supports more than one keyboard layout, or because information is required to support different languages and their character sets.

2.2.6 Personal Dictionary

A Personal Dictionary allows the Suggestion Engine to learn from, and adapt to, individual Users. Commands are provided to support viewing and managing of personal dictionaries.

2.2.7 Multi-threading

The Framework and its components are not designed to be re-entrant, thus some form of access control is required if the Framework is used in a multi-threaded environment; this is provided by the Framework locking mechanism (see 4.6 and 4.7).

2.2.8 Synchronisation

It is important that the user application ensures the contents of the Framework's text input buffer (see 5.9) reflect the contents of the window in which User text is being entered. The contents of the text input buffer are used by the Suggestion Engine when determining suggestions, thus if the buffer does not reflect the contents of the window the generated suggestions may appear incorrect.

2.2.9 Data Ownership

A number of Framework commands return data structures to the caller. In some instances responsibility for managing the data is retained by the Framework, but in other instances responsibility for the data is handed to the caller. The caller is then expected to free the memory used to store the data structures when it is convenient to do so (see 4.8).

2.2.10 Language Tags

The Framework uses a number of components, for example, dictionaries and key maps, whose contents depend on the language they are intended to support. To avoid the need for application developers to learn and make use of proprietary language identifiers, the Framework supports the standard BCP47 (see Ref 2 and 3).

BCP47 specifies that a language is defined by a language tag, comprising a sequence of one or more sub-tags, each of which refines or narrows the range of language identified by the overall tag. The following example of a language tag is taken from the standard:

```
sr-Latn-RS (Serbian written using the Latin script as used in Serbia)
```

An application using the Framework may at times wish to restrict the language-based resources to be used by some Framework functionality; for example, the application may request that suggestions are generated only in French. The Framework provides two options:

1. Select a single resource that best matches a language tag
2. Select all resources that match a language tag

Note: this document distinguishes between **language tags**, as defined in BCP47, and **language ids**, run-time identifiers assigned by the Framework.

2.2.11 Capitalisation

The Framework learns capitalisation to reflect user input. For example, if a dictionary contains the word "march" and the word "March" is entered, the capitalisation information for the word is updated. The Framework can also be asked to provide suggestions with specific capitalisation, see 5.10.2).

2.2.12 Correction Suggestions

Correction Suggestions are supplied by the Framework when it thinks changes should be made to entered text. There are two types of Correction Suggestion Error Correction and proximity suggestions – generated when the Input Manager believes the previous word was misspelled or mistyped

2.2.12.1 Enabling Correction Suggestions

Correction Suggestions are enabled as part of suggestion configuration, see 5.10.4.

3. Framework Functionality

The following tables describe the functionality provided by the Framework and its components. Base Framework functionality is necessary for the correct operation of the Framework when generating suggestions. Other Framework functionality can be used or ignored as desired.

Functionality	**Description**	**Ref**
Component Management	A component represents an object that provides data or functionality. Packages are used to install additional components.	5.3
Configuration	The Framework can be configured to run in a single- or multi-threaded environment.	5.2
Dictionaries	The Suggestion Engine uses dictionaries when generating suggestions. The capability is provided to select and restrict the dictionaries to be used. Additional dictionaries are made available by installing packages.	**Error! Reference source not found.**
Input Manager	The Input Manager processes User input into a form that can be utilised by other Framework components.	5.9
Key mappings	Key mapping data files allow the Framework to support multiple input methods and keyboard layouts.	5.8
Personal Dictionary	A Personal Dictionary is created for each User and stores words entered by a User that are not in any of the Framework's dictionaries. A Personal Dictionary thus allows the Suggestion Engine to generate suggestions tailored to a User. In addition, the Framework provides the capability to view and modify the contents of a Personal Dictionary.	5.6
Suggestion Engine	The Suggestion Engine can be asked to generate suggestions in response to User input.	5.10

Table 1. Base Framework Functionality

Functionality	**Description**	**Ref**
Error Manager	All Framework functions and commands return a status value. The Error Manager provides the capability to store and retrieve more detailed error information.	5.12
Learning	The Suggestion Engine learns from User input. It can also be asked to learn from a supplied text buffer. New words are added to the Personal Dictionary of the current user.	5.11

Output and Logging	The Framework provides the capability to log information for later analysis.	5.13
Packages	Packages are the mechanism through which components are added to, and removed from, the Framework. A package must be installed separately for each User.	5.7

Table 2. Other Framework Functionality

4. Framework Functions

The Framework functions are used to manage the Framework and access its functionality.

For a detailed description of each function, and information on how each function is expected to be used, see Ref 1.

4.1 KPTFwkCreate

The function KPTFwkCreate is called to create the Framework. This function can be called on start up and after a successful call to KPTFwkDestroy. If called at any other time this function does nothing and reports an error.

4.2 KPTFwkDestroy

The function KPTFwkDestroy is called to destroy the Framework. All resources used by the Framework are released. This function should only be called after a previous call to KPTFwkCreate returned successfully. If called at any other time this function does nothing and reports an error.

4.3 KPTFwkRunCmd

The function KPTFwkRunCmd is the means by which Framework functionality is accessed. The function is supplied with the identity of the command to be run and the parameters required to support that command.

4.4 KPTFwkCancelCmd

The function KPTFwkCancelCmd is used to cancel the currently active command. The command will cease when it safe to do so, and may therefore not cease immediately. This function is only meaningful for commands that are expected to run for some time and are therefore capable of being cancelled.

4.5 KPTFwkGetVersion

The function KPTFwkGetVersion provides Framework version information. The information supplied is the major version number, the minor version number, the patch number, and the build number. This function may be called at any time.

4.6 KPTFwkGetLock

If the Framework has been configured to support locking (see 5.2 below), it will automatically attempt to obtain a lock before carrying out a command.

A user application may obtain its own lock. This allows the application to obtain a lock before attempting a sequence of commands, thus ensuring those commands will run to completion without interruption. A lock may be obtained more than once by the same user application thread, but each successful lock must have a corresponding release (see below).

4.7 KPTFwkReleaseLock

If the Framework has been configured to support locking (see 5.2 below) it will automatically release a lock on completion of a command.

If a user application has obtained a lock before executing one or more commands it must also call this function to release each lock it previously obtained. It is important to note that the Framework remains locked until all locks have been released.

4.8 KPTFwkReleaseAlloc – Data Ownership

When a command supplies data, responsibility for that data is sometimes passed to the caller. The caller is then responsible for managing the memory allocated to store the data. Data for which the caller is responsible is clearly identifiable – it is supplied in a structure whose name finishes with the string “AllocT”; e.g. KPTComponentListAllocT. When the caller no longer requires the data it ***must*** be released using the function KPTFwkReleaseAlloc. Passing the data pointer to the *free()* function will lead to undefined behaviour.

4.8.1 Managing AllocT Structures

Allocation structures are used in the following way:

1. Declare the appropriate AllocT structure
2. Initialise the structure to zero
3. Run the desired command and pass the structure with the command
4. Repeat the previous step as required
5. Call KPTFwkReleaseAlloc to free all resources associated with the structure

Note that the structure must always be initialised once; it can be reused before being released.

5. Framework Commands

All Framework functionality is accessed through commands. Commands are passed to the Framework using the function KPTFwkRunCmd (see 4 above).

For detailed information on data structures and basic types see Ref 1.

5.1 Command Identifier Naming Convention

Framework command identifiers have the form KPTCMD_<component>_<operation>.

5.2 Configuring the Framework

The Framework may be used in single- or multi-threaded environments. To function properly in a multi-threaded environment the Framework must be configured to implement locking. This ensures that Framework commands can be accessed by only one thread at a time.

The following table describes the configuration commands supported by the Framework:

Command Identifier	**Description**
KPTCMD_CONFIG_GETLOCKINGSTATE	Get the current Framework locking state.
KPTCMD_CONFIG_SETLOCKINGSTATE	Set the Framework locking state.

Table 3. Configuration Commands

When locking is enabled, the Framework will attempt to obtain and release a lock when running a command. A user application is free to obtain and release additional locks as required, in order to ensure that a sequence of commands is not interrupted.

5.3 Component Management

A component represents an object that provides data or functionality to the Framework. Components are managed individually for each User, allowing each User to be configured with different sets of data and functionality.

The following table describes the component commands provided by the Framework:

Command Identifier	**Description**
KPTCMD_COMPONENT_GETAVAILABLE	Obtain a list of components installed for the current User.
KPTCMD_COMPONENT_GETLOADED	Obtain a list of the components presently loaded for the current User.
KPTCMD_COMPONENT_LOAD	Load a component for the current User.
KPTCMD_COMPONENT_UNLOAD	Unload a component for the current User.

Table 4. Component Commands

The command KPTCMD_COMPONENT_GETAVAILABLE supplies the following information for each component:

Field	Description
Component id	Run-time identifier for component.
Type	Component type.
Version	Component version.
Loaded state	True or false.
Active state	True or false.
Component-specific structure	Pointer to be cast as appropriate by caller.

Table 5. Component Information

5.4 Profile Management

The Framework is designed to maintain the profile. The Suggestion Engine learns from the user and adapts it further performance. The framework provides commands to load and save data.

The following table describes the profile related commands provided by framework

Command Identifier	Description
KPTCMD_PROFILE_RELOAD	Loads the component and data configuration
KPTCMD_PROFILE_SAVE	Saves the modified data and configuration settings.

Table 6. Profile Management

5.5 Dictionary Components

The Suggestion Engine uses dictionary components to provide support for different languages. In general, each language has one dictionary associated with it. Further dictionaries may be supplied for a language in order to provide support for specialised subjects; e.g. law, or medicine.

Dictionaries are made available to the Suggestion Engine in one of two ways:

1. as part of the installation
2. contained in packages (see 5.7)

When a User is first logged-in it is provided with all dictionaries provided as part of the Framework installation. Additional dictionaries can be added by installing packages.

NOTE: Dictionaries installed from a package are available only to the User that is current when the package is installed. A package must be installed separately for each User that wishes to use its dictionaries.

The following table describes the dictionary commands provided by the Framework:

Command Identifier	Description
KPTCMD_DICTIONARY_GETLANGLIST	Obtain a list of languages available from the set of loaded dictionaries – can be filtered or restricted by language tag.
KPTCMD_DICTIONARY_GETLIST	Obtain a list of dictionaries installed for the current User – can be filtered or restricted by language tag.
KPTCMD_DICTIONARY_GETSTATE	Obtain the state of a loaded dictionary.
KPTCMD_DICTIONARY_SETSTATES	Set the states of a number of dictionaries.

Table 7. Dictionary Commands

The command KPTCMD_DICTIONARY_GETLIST provides the following data for each installed dictionary:

Field	Description
Field mask	Identifies which fields are set.
File name	The dictionary file name.
Display name	The dictionary name for display purposes.
Language id	The dictionary language id
Type	The dictionary type; e.g. legal, medical etc.
Version	The dictionary version.
Loaded	True or false.

Table 8. Dictionary Information

Field	Description
Field mask	Identifies which fields are set.
Component id	Run-time identifier for component.
Active	True or false.
Priority	Highest priority is 0.

Table 9. Dictionary State Information

The above state information is also supplied for a loaded dictionary by KPTCMD_DICTIONARY_GETSTATE.

A dictionary's priority determines its relative importance when suggestions are being generated.

Dictionaries are loaded and unloaded for a User using the commands KPTCMD_COMPONENT_LOAD and KPTCMD_COMPONENT_UNLOAD (see 5.3).

5.6 Personal Dictionary

The purpose of a Personal Dictionary is to store words entered by a User that are not found in any of the User's dictionaries; i.e. a Personal Dictionary is how the Suggestion Engine learns a User's vocabulary. Each User has its own Personal Dictionary.

5.6.1 Populating a Personal Dictionary

Words are normally added to a Personal Dictionary by the Suggestion Engine as it learns from User input. The Framework provides the facility to explicitly add one or more words to the Personal Dictionary of the current User.

5.6.2 Viewing the contents of a Personal Dictionary

The contents of the current User's Personal Dictionary may be retrieved for display. The Framework can be asked to display the contents using a number of different sorting criteria (for details see Ref 1). Contents may be viewed in their entirety or supplied as a number of pages.

NOTE: if the contents of a Personal Dictionary are modified while a view is open the view becomes stale and all viewing commands will fail. The existing view must be closed and a new view opened.

5.6.3 Editing a Personal Dictionary

It is recognised that not all words in a Personal Dictionary are suitable for storing and suggesting; for example, a word may have been stored as new word because it is a misspelled or mistyped version of a word. The Framework therefore provides the ability to remove words from the Personal Dictionary of the current User. The following is a suggested procedure:

1. open a view on the Personal Dictionary

2. identify words to be removed (store the text or query Personal Dictionary for identifiers)
3. close the view
4. pass the word texts (or identifiers) to the Personal Dictionary for removal

Note: if words are removed while a view is open the view must be closed and re-opened.

The Framework also provides the capability to remove all words from a Personal Dictionary in a single operation.

5.6.4 Archiving the contents of a Personal Dictionary

The contents of a Personal Dictionary can be exported to a file for safe-keeping and later re-importing. The caller supplies a fully-qualified file name when exporting and importing. The output file has a proprietary format, thus importing can only take place form a file which was previously exported.

5.6.5 Personal Dictionary Commands

The following table describes the Personal Dictionary commands provided by the Framework:

Command Identifier	**Description**
KPTCMD_PERSONAL_ADDWORDS	Add one or more words to the current Personal Dictionary.
KPTCMD_PERSONAL_CLOSEVIEW	Close a view on the current Personal Dictionary.
KPTCMD_PERSONAL_COMMITWORDS	Commit words to the current Personal Dictionary. **NOTE**: not currently supported
KPTCMD_PERSONAL_EDITWORD	Modify a word in the current Personal Dictionary **NOTE**: not currently supported
KPTCMD_PERSONAL_EXPORTTOFILE	Write the contents of the current Personal Dictionary to a file.
KPTCMD_PERSONAL_GETCONFIG	Get the configuration settings for the current Personal Dictionary.
KPTCMD_PERSONAL_GETENTRYCOUNT	Get the number of entries in the current Personal Dictionary.
KPTCMD_PERSONAL_GETIDFORWORD	Query the current Personal Dictionary for a word's identifier.
KPTCMD_PERSONAL_GETIDSFORTAG	Query the current Personal Dictionary for all word identifiers with an associated tag. **NOTE**: not currently supported
KPTCMD_PERSONAL_GETTAGFORWORD	Query the current Personal Dictionary for a word's tag. **NOTE**: not currently supported
KPTCMD_PERSONAL_GETWORDFORID	Use an identifier to obtain a word from the current Personal Dictionary.
KPTCMD_PERSONAL_IMPORTFROMFILE	Add the contents of a file to the current Personal Dictionary. The file ***must*** previously have been exported from a Personal Dictionary.

KPTCMD_PERSONAL_OPENVIEW	Request a view on the current Personal Dictionary.
KPTCMD_PERSONAL_REMOVEWORDS	Remove one or more words from the current Personal Dictionary.
KPTCMD_PERSONAL_REMOVEALLWORDS	Remove all words from the current Personal Dictionary.
KPTCMD_PERSONAL_SETCONFIG	Set configuration options for the current Personal Dictionary.
KPTCMD_PERSONAL_VIEWPAGE	Page through a view of the current Personal Dictionary.

Table 10. Personal Dictionary Commands

5.7 Packages

Packages are the means by which components are made available to, or removed from, the Framework.

A package may contain one or more components. When a package is installed, all its components are installed. It is not possible to install individual components. Similarly, when a package is uninstalled, all its components are removed. If a package is installed for more than one User it will not be completely removed from the host device until it has been uninstalled for each User.

The Framework does not automatically install packages. Packages are installed only when the Framework is instructed to do so.

There is a degree of flexibility in the location of uninstalled packages as the Framework can be provided with the location to be searched when seeking available packages.

The following table describes the package management commands provided by the Framework:

Command Identifier	Description
KPTCMD_PACKAGE_GETAVAILABLE	Obtain a list of packages available for installation.
KPTCMD_PACKAGE_GETINSTALLED	Obtain a list of packages installed for the current User.
KPTCMD_PACKAGE_INSTALL	Install a package for the current User.
KPTCMD_PACKAGE_UNINSTALL	Uninstall a package for the current User.

Table 11. Package Management Commands

5.8 Key Mappings

The Framework is designed to support multiple input methods and keyboard layouts. This is achieved through the use of key mapping data files configured specifically for individual devices and languages.

Key mappings are primarily used to support ambiguous text entry, providing information on which characters are mapped to which keys.

Each key mapping is uniquely identified by:

1. language id
2. language tag

3. manufacturer
4. device
5. group

The group field identifies compatible key maps within a manufacturer-device pair.

Each key map layout contains an array of key data items, with each item defining all the characters on that key.

It is possible to have multiple language layouts in use at the same time, as long as they belong to the same group.

When a character is passed to the Input Manager it is supplied with associated character attributes and a key identifier. If the attributes specify that the key is ambiguous, Framework components use key mapping data to identify other characters associated with that key. This allows the Framework to support keyboards that have multiple characters on each key.

5.8.1 Active Key Map

The Framework has the concept of an active key map. The active key map is formed when the Framework automatically merges all the key maps specified using the command KPTCMD_KEYMAP_SETACTIVE. The active key map is used by the Input Manager and Suggestion Engine to process information. All of the key maps included in the active key map must belong to the same group.

It should be noted that Framework will not support ambiguous text entry unless an active key map has been set.

All key maps passed to this command must be open (see KPTCMD_KEYMAP_OPEN).

5.8.2 Creating a Custom Key Map

The Framework will normally be installed with one or more key maps appropriate to the host device. If the need arises for additional key maps to be created at run time the necessary commands are provided by the Framework.

The command sequence is:

1. KPTCMD_KEYMAP_OPEN - creates a key map when asked to do so
2. KPTCMD_KEYMAP_SETLAYOUT - populate the key map
3. KPTCMD_KEYMAP_SAVE - save the key map for later re-use

5.8.3 Key Mapping Commands

The following table describes the key map commands provided by the Framework:

Command Identifier	Description
KPTCMD_KEYMAP_CLOSE	Unloads a key map from memory. Fails if the key map is not open.
KPTCMD_KEYMAP_DELETE	Deletes a key map from disk. Fails if the key map is open.
KPTCMD_KEYMAP_GETACTIVE	Get the active key map.
KPTCMD_KEYMAP_GETAVAILABLE	Get identifiers of key maps on disk.

KPTCMD_KEYMAP_GETLAYOUT	Get the key map layout for a language and group.
KPTCMD_KEYMAP_GETOPEN	Get identifiers of open key maps.
KPTCMD_KEYMAP_OPEN	Load a key map from disk. Fails if the key map is already open. Creates a new key map if requested to do so.
KPTCMD_KEYMAP_QUERY	Query a key map.
KPTCMD_KEYMAP_SAVE	Save a key map to disk.
KPTCMD_KEYMAP_SETACTIVE	Set the active key map.
KPTCMD_KEYMAP_SETLAYOUT	Set the key map layout for a language and group.

Table 12. Key Map Commands

5.9 Input Manager

The Input Manager processes User-entered text into a form that can be used and understood by other Framework components. Text can be of two types – ambiguous or non-ambiguous. When ambiguous text is being entered the Framework maintains a composition object that is updated to reflect User inputs. When non-ambiguous text is being entered there is no need for a composition object as each character has a known representation.

This diagram illustrates the concepts associated with the text input buffer. The characters fc1, fc2, fc3 and fc6 are non-ambiguous. The characters ac4 and ac5 are ambiguous.

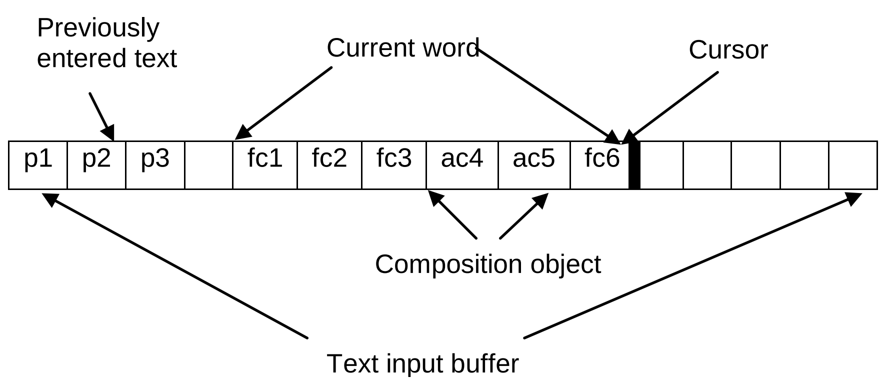


Figure 3. Text Input Buffer

5.9.1 Text Input Buffer

The Input Manager maintains the text input buffer. User-entered characters and/or keystrokes are inserted into the text input buffer and processed by the Input Manager. The text input buffer provides a history of entered text. It also handles the text currently being entered.

It is important to note that the text input buffer is the Input Manager's view on text that has been entered by a User. The calling application is responsible for updating any user interfaces as text is entered.

5.9.2 Current Word

The current word is composed of two types of character – ambiguous and non-ambiguous. Ambiguous characters are treated as having more than one representation and generally arise when text is being entered using a keyboard where each key contains more than one character.

5.9.3 Composition Object

The Input Manager allows only one part of the current word to be ambiguous; i.e. all ambiguous characters must be contiguous in the current word. The section of the current word containing ambiguous characters is known as the composition object.

5.9.4 Managing the Text Input Buffer

5.9.4.1 Text Input Buffer Cursor

The Input Manager maintains a cursor position for the text input buffer. The position of the cursor determines where operations on the text input buffer take place.

The following cause a change of cursor position:

1. data inserted into buffer
2. data removed from buffer
3. caller-requested position change

5.9.4.2 Inserting a Character

Each character entered by a User is passed to the Input Manager and inserted in the text input buffer at the cursor position. Characters are accompanied by an id and a set of attributes defining how they are to be handled. The composition object is updated as required.

Attribute	Description
Insert ambiguous	Insert character or text as ambiguous.
Insert cap sentence	Respect sentence casing if appropriate.
Insert cap	Insert character or text capitalised.
Insert no learning	Insertion should not cause learning to occur.
Insert map key id	Match id against a key id in a key map.
Insert map key vid	Match id against a vkey id in a key map.
Insert symbol	Character or text is fixed within the composition object; i.e. it is not ambiguous.

Table 13. Character Attributes

5.9.4.3 Inserting a String

A caller-supplied string can be inserted in the text input buffer. The text input buffer and composition object are updated as required. The cursor is placed at the end of the inserted string. Each character in the inserted string is supplied with an id and a number of attributes (see above) defining how it is to be handled. This operation is provided for applications that support cutting-and-pasting of text.

5.9.4.4 Inserting a Suggestion

When a User is entering text they have the option of accepting a suggested word or phrase to replace the text they are entering. The selected suggestion is passed to the Input Manager and the text input buffer is updated to replace the partially entered text. The Input Manager will insert a trailing space after the inserted suggestion if it is asked to do so. The cursor is placed at the end of the inserted text and the composition object is cleared.

When it is asked to insert a suggestion the Framework determines the actions required to allow the insertion to take place; i.e. what existing characters have to be removed and replaced with the suggestion. This information is passed back to the caller in order for them to make the corresponding changes in the displayed text.

5.9.4.5 Removing Characters

Characters can be removed from the text input buffer at the cursor position. Characters may be removed on both sides of the cursor. The composition object is updated as required.

5.9.4.6 Replacing Buffer Contents

A caller-supplied buffer can be used to replace part, or all, of the contents on the text input buffer. The cursor is positioned at the end of the inserted text. The composition object is updated as required.

5.9.4.7 Resetting the Buffer

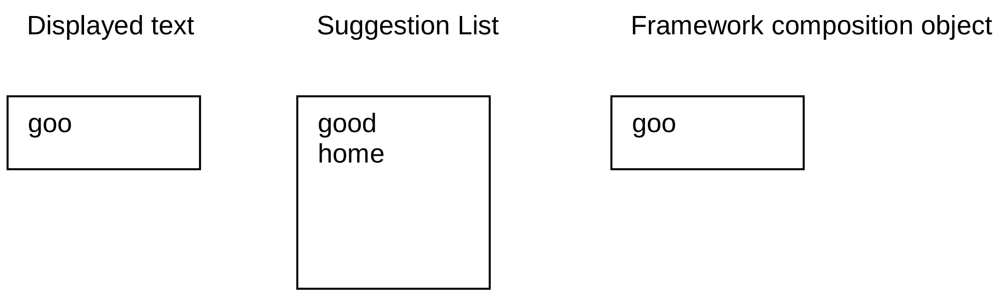
The contents of the text input buffer can be cleared with a single operation. An optional text string can be supplied to repopulate the text input buffer, in which case the cursor will be positioned at the end of the inserted text. This operation clears the composition object.

5.9.5 Synchronising to a Suggestion

This functionality is of use when ambiguous text is being entered.

5.9.5.1 Optimising Insertions

Consider the situation below. After three ambiguous characters have been entered the Framework has generated a number of suggestions.



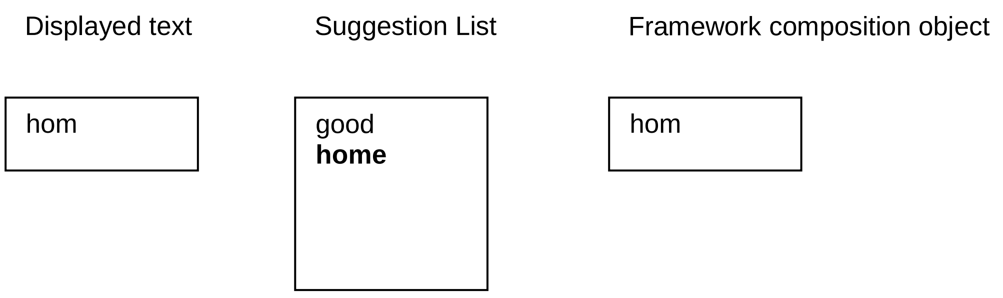
If the suggestion **good** is selected for insertion the only action required to modify the displayed text is to append the letter **d**. However, consider the situation when the User has scrolled through the suggestion list and selected the suggestion **home**. In this situation, the Framework will decide the actions necessary to insert the suggestion **home** require that three characters, **goo**, must first be deleted.

There is a more efficient solution. When the User selects the suggestion **home** for insertion:

1. Change the displayed text to **hom**, reflecting the number of characters entered

2. tell the Framework to synchronise to the suggestion **home**

As a result of these actions, the Framework composition object is updated to **hom**, reflecting the displayed text (as represented in the diagram below). The Framework can now determine that the only action required to insert the suggestion **home** is to append the letter **e**.



It is feasible to request synchronisation with each suggestion in a suggestion list as the User is scrolling, but it is more efficient to wait until a suggestion has actually been selected.

5.9.5.2 Last Known Good Suggestion

When the User enters sufficient ambiguous characters that the Framework can no longer find any suggestions, the most recently synchronised suggestion is used as a root for suggested words, with the remaining characters being supplied by subsequent User inputs.

The Framework automatically synchronises to the top suggestion in a suggestion list. This remains the most recently synchronised suggestion until the calling application requests synchronisation.

5.9.6 Input Manager Configuration

The Input Manager has four configuration item:

1. whether CR/LF sequences are merged into a single character. The Framework is designed to work on multiple platforms with different text entry applications. In order to synchronise the insertion point the Input Manager needs to know how line-endings affect the cursor position..

5.9.7 Input Manager Commands

The following table describes the Input Manager commands provided by the Framework:

Command Identifier	Description
KPTCMD_INPUTMGR_GETCONFIG	Get current configuration settings.
KPTCMD_INPUTMGR_GETCURRWORD	Get details of the current word object.
KPTCMD_INPUTMGR_GETCURSOR	Get cursor position and total number of characters in the text input buffer.
KPTCMD_INPUTMGR_PREINSERTCHAR	Returns details of corrections to be applied before inserting a character.
KPTCMD_INPUTMGR_INSERTCHAR	Add a character to the text input buffer.
KPTCMD_INPUTMGR_INSERTSTRING	Add a string to the text input buffer.
KPTCMD_INPUTMGR_INSERTSUGG	Insert a suggestion into the text input buffer.

KPTCMD_INPUTMGR_MOVECURSOR	Change the position of the cursor.
KPTCMD_INPUTMGR_REMOVE	Remove characters from the text input buffer.
KPTCMD_INPUTMGR_REPLACECONTENTS	Replace part of the text input buffer with the supplied text.
KPTCMD_INPUTMGR_RESET	Reset the contents of the text input buffer.
KPTCMD_INPUTMGR_SETCONFIG	Apply configuration settings.
KPTCMD_INPUTMGR_SYNCTOSUGG	Synchronises the current word with the supplied suggestion.
KPTCMD_INPUTMGR_UPDATECURRWORD	Update attributes for the current word object.

Table 14. Input Manager Commands

5.10 Suggestion Engine

The Suggestion Engine generates suggestions in response to text entered by a User. Suggestions are either words.

5.10.1 Word Suggestions

Word suggestions are based on the contents of the current word. It is not necessary that the current word is populated; the Suggestion Engine can provide word suggestions when no text has been entered.

5.10.2 Capitalisation and Suggestions

The Suggestion Engine allows the capitalisation of suggested words to be specified when suggestions are requested (see 5.10.4). The following options are supported:

1. Use stored capitalisation – apply capitalisation as defined in the source dictionary
2. Force lower – convert all letters to lower case
3. Force upper – convert all letters to upper case
4. Apply sentence case – capitalise the first letter
5. Honour user capitalisation – apply capitalisation as previously entered by a User

5.10.3 Restricting Suggestion Sources

Suggestions are drawn from all active User dictionaries and the User’s Personal Dictionary. The Framework provides the ability to restrict suggestions to a subset of active languages and dictionaries by specifying desired language or component ids respectively.

5.10.4 Suggestion Configuration

A number of configuration options are provided for word suggestions:

1. Maximum number of suggestions to be supplied
2. Number of elision completions allowed. An example of an elided form is “l’arbre”. Elision completions can be expensive to calculate.
3. Suggestion threshold. The Suggestion Engine can be asked to limit its suggestions to the most commonly used words by specifying a threshold, in the range 0 to 100. A value of 0 means all words will be considered. The set of words from which suggestions will be drawn decreases as the threshold is increased.

4. Include proximity suggestions
5. Include correction suggestions
6. Allow word completion. If disabled, the length of suggested words is determined by the number of characters entered.
7. Apply capitalisation as defined in the dictionary from which a suggestion is drawn
8. Suggest words wholly lower case
9. Suggest words wholly upper case
10. Use sentence case for first letter of suggested words
11. Use capitalisation as previously entered by a User

NOTE: The following suggestion configuration options are not currently supported – number of elision completions, suggestion threshold, suggest words wholly lower case.

5.10.5 Generating Suggestions

Suggestion generation is carried out in a few basic steps:

1. Configure the Suggestion Engine to generate desired suggestion types
2. Configure the Suggestion Engine to select the dictionaries and languages to be used
3. Insert the latest User input into the Input Manager
4. Ask the Suggestion Engine to generate suggestions

The following diagram is a basic illustration of the activities involved in generating suggestions:

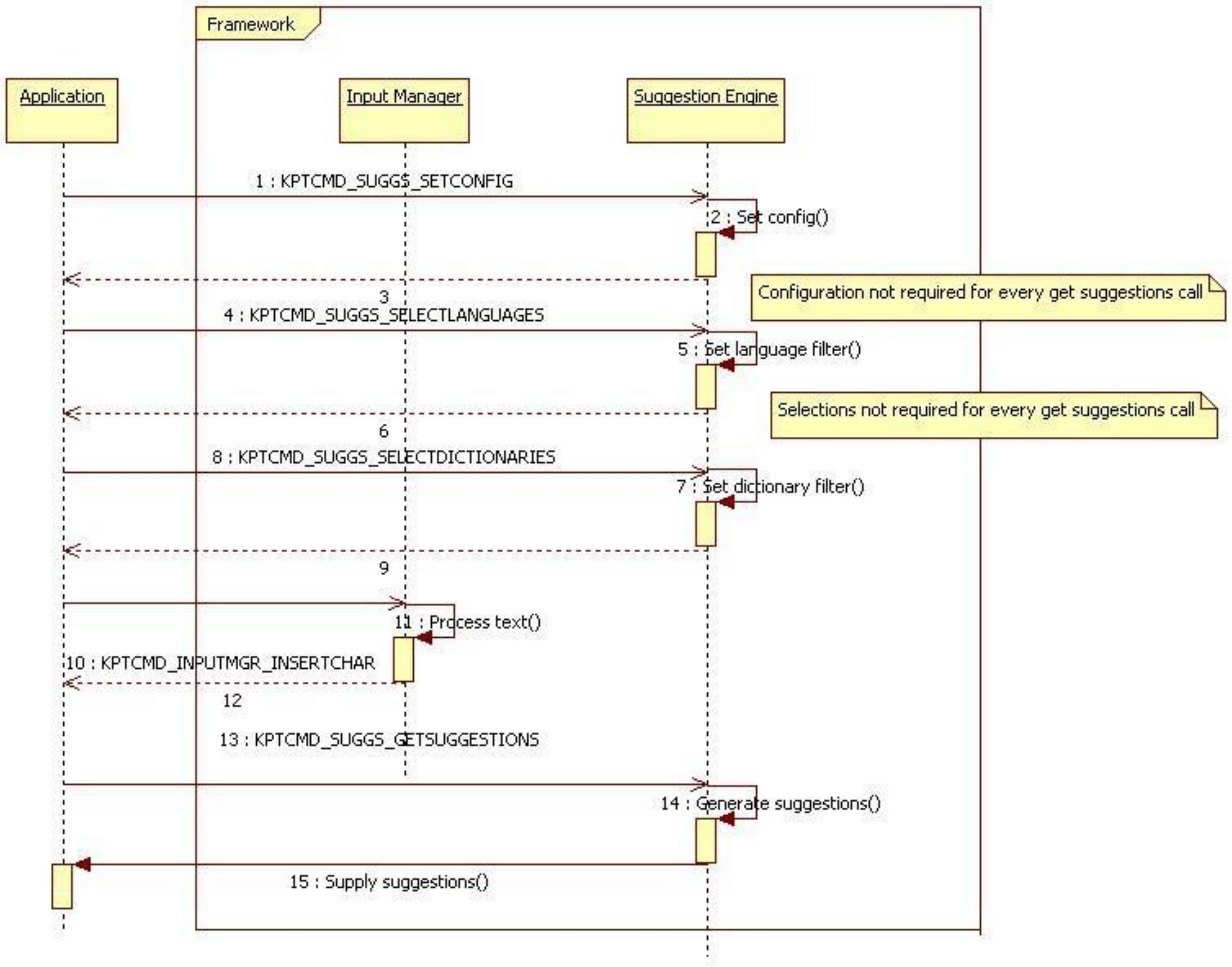

Framework
Application
Input Manager
Suggestion Engine
1 : KPTCMD_SUGGS_SETCONFIG
2 : Set config()
Configuration not required for every get suggestions call
3
4 : KPTCMD_SUGGS_SELECTLANGUAGES
5 : Set language filter()
Selections not required for every get suggestions call
6
8 : KPTCMD_SUGGS_SELECTDICTIONARIES
7 : Set dictionary filter()
9
11 : Process text()
10 : KPTCMD_INPUTMGR_INSERTCHAR
12
13 : KPTCMD_SUGGS_GETSUGGESTIONS
14 : Generate suggestions()
15 : Supply suggestions()

Figure 4. Generating Suggestions

5.10.6 Suggestion Commands

The following table describes the suggestion commands provided by the Framework:

Command Identifier	Description
KPTCMD_SUGGS_GETCONFIG	Obtain current suggestion configuration settings.
KPTCMD_SUGGS_GETSUGGESTIONS	Obtain a new set of suggestions.
KPTCMD_SUGGS_SELECTDICTIONARIES	Specify the dictionaries to be used when generating suggestions.
KPTCMD_SUGGS_SELECTLANGUAGES	Specify the languages to be used when generating suggestions.
KPTCMD_SUGGS_SETCONFIG	Configure suggestion generation.

Table 15. Suggestion Commands

5.11 Learning

The Suggestion Engine learns from the writing style of individual Users. It does this in order to continually improve the quality of its suggestions for each User.

5.11.1 Automatic Learning

The Suggestion Engine can be configured to learn automatically as a User types. Automatic learning can be disabled when not required; for example, when the User is typing in a password field.

5.11.2 Requested Learning

The Suggestion Engine can be asked to learn from the contents of the text input buffer. The Suggestion Engine can be asked to learn the entire buffer, the word containing the cursor, or the word before the previous space.

5.11.3 Learning from Supplied Text

The Suggestion Engine can be asked to learn from a buffer of supplied text. Learning may be done in a single pass, or in a number of chunks of data.

5.11.4 Learning Commands

The following table describes the Learning commands provided by the Framework:

Command Identifier	Description
KPTCMD_LEARN_BUFFER	Ask the Suggestion Engine to learn from the contents of a supplied text buffer.
KPTCMD_LEARN_CONTENTS	Ask the Suggestion Engine to learn from the text in its buffer.
KPTCMD_LEARN_GETOPTIONS	Obtain the current learning options.
KPTCMD_LEARN_SETOPTIONS	Set the learning options.

Table 16. Learning Commands

A progress callback function may optionally be provided when learning is requested, allowing the caller to monitor and possibly cancel a learning operation.

The learning configuration options are to enable and disable automatic learning.

5.12 Error Manager

Every Framework command returns a result. The result can be interrogated using the macros KPTRESULT_FAILED or KPTRESULT_SUCCEEDED to determine whether or not the command succeeded. If the command failed the result contains an error code. The error code provides information on what went wrong, but may not necessarily provide information on why the problem arose; e.g. the result may say that data couldn't be written to a file, but the underlying problem may be that the output file was read-only. To provide such detailed error information the Framework provides an error stack. The error stack provides a chain of error information from the point where a problem occurred to the point where a Framework command concludes.

The Error Manager automatically sends errors to the Output and Logging component using the category KPTCAT_ERROR (see 5.13.12); this category must therefore be enabled if error logging is desired.

The following table describes the Error Manager commands provided by the Framework:

Command Identifier	Description
KPTCMD_ERROR_CLEARSTACK	Clear the error stack.
KPTCMD_ERROR_POPSTACK	Read and remove the top error from the error stack.

Table 17. Error Stack Commands

5.13 Output and Logging

The Framework provides the ability to log error information and caller-supplied text. The basic output functionality accepts a text string for outputting and writes it to the currently active destinations. Additional data may be output if desired.

5.13.1 Output Destinations

The Framework provides three built-in output destinations:-

1. Default output is sent to the host platform debug output stream.
2. Console output is sent to the currently active console window (stdout).
3. Text file output is written to a file whose name is supplied by the calling application.

5.13.2 Custom Output

In addition to the above output destinations, the Framework supports the registering of a custom output callback function. If a callback function has been registered it is called whenever an output string is written. This allows the caller to implement their own logging functionality. The Framework supports a single output callback function.

5.13.3 Output Categories

The output functionality can be configured to write only a selected set of output categories. When a text string is supplied for outputting it is accompanied by information specifying which category, or categories, the output belongs to. If the output functionality is currently configured to output one of the supplied categories the text string will be written to the currently active destinations and a call made to the custom output callback function.

The following categories are provided by default: KPTCAT_ALL, KPTCAT_NONE, KPTCAT_ERROR, KPTCAT_WARNING, KPTCAT_INFORMATION, KPTCAT_DEBUG, KPTCAT_PERFORMANCE, and KPTCAT_ALGORITHM. In addition, there are a further 16 categories, KPTCAT_APP01 to KPTCAT_APP16 which the caller is free to map to their own data categories.

5.13.4 Enabling and Disabling Output

Output is enabled and disabled through the setting of desired categories, and the enabling and disabling of output destinations.

5.13.5 Output Format

The most basic form of output is the text string supplied for output. The Framework can be asked to provide the following additional information with each printed text string:- the time, the date, the error categories associated with the string, and the result of the operation that caused the string to be output. A fully populated output has the following format:

yyyy-mm-dd hh-mm-ss category result text.

In addition, each output may be indented to improve clarity. Further options are provided to add a new line after the output has been written, and to flush the output buffer after each write.

5.13.6 Output Commands

The following table describes the Output commands provided by the Framework:

Command Identifier	Description
KPTCMD_OUTPUT_ADDCUSTOMOUTPUT	Register an output callback function.
KPTCMD_OUTPUT_DISABLEOUTPUT	Disable a built-in output mechanism.
KPTCMD_OUTPUT_ENABLEOUTPUT	Enable a built-in output mechanism.
KPTCMD_OUTPUT_GETCATEGORIES	Get the information categories that are being output.
KPTCMD_OUTPUT_GETOPTIONS	Get the current output options.
KPTCMD_OUTPUT_MODIFYINDENT	Modify the indentation used when writing output.
KPTCMD_OUTPUT_REMOVECUSTOMOUTPUT	Remove a previously registered output callback function.
KPTCMD_OUTPUT_SETCATEGORIES	Set the information categories to be output.
KPTCMD_OUTPUT_SETOPTIONS	Set desired output options.
KPTCMD_OUTPUT_WRITE	Send text information for outputting.

Table 18. Output Commands