IME Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purposes and Features

IME is a software component that provides text input functions. By using IME, the application can obtain the results of text input by the user.

Two libraries have been provided to use IME from an application. The first one is the IME Dialog library, one of the Common Dialog libraries, while the second one is libime. The application selects the library to be used based on the purpose of use.

IME Dialog Library and libime

The following is an explanation of the IME Dialog library and libime.

IME Dialog Library

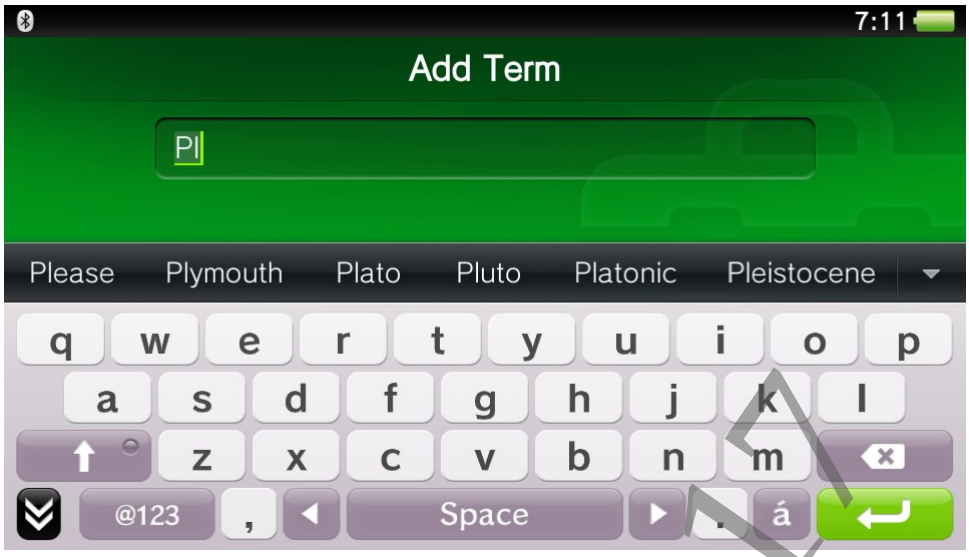
One of the functions of the Common Dialog library, in addition to IME it provides a text box and a title bar. The application can retrieve the input text as a result without implementing character and title bar display functions.

Figure 1 IME Dialog Library Example

libime

libime is a library for communicating with IME. By performing communication with IME using event handlers and APIs, the application is able to draw the caret and character string that are being edited. This enables text input with high affinity to the application.

Figure 2 libime Example



Related Files

The following files are necessary to use the IME Dialog library.

File Name	Description
ime_dialog.h	Header file
libime.h	
libSceCommonDialog_stub.a	Stub library file

The following files are necessary to use libime.

File Name	Description
libime.h	Header file
libSceIme_stub.a	Stub library file
libSceIme_stub_weak.a	weak import stub library file

Sample Programs

Sample programs using IME are as follows.

sample_code/input_output_devices/api_ime_dialog/basic/

This is a basic sample program using the IME Dialog library.

sample_code/input_output_devices/api_libime/basic/

This is a basic sample program using libime.

Reference Documents

Please refer to the following document for information on common limitations, specifications, etc. in the Common Dialog library.

- Common Dialog Overview

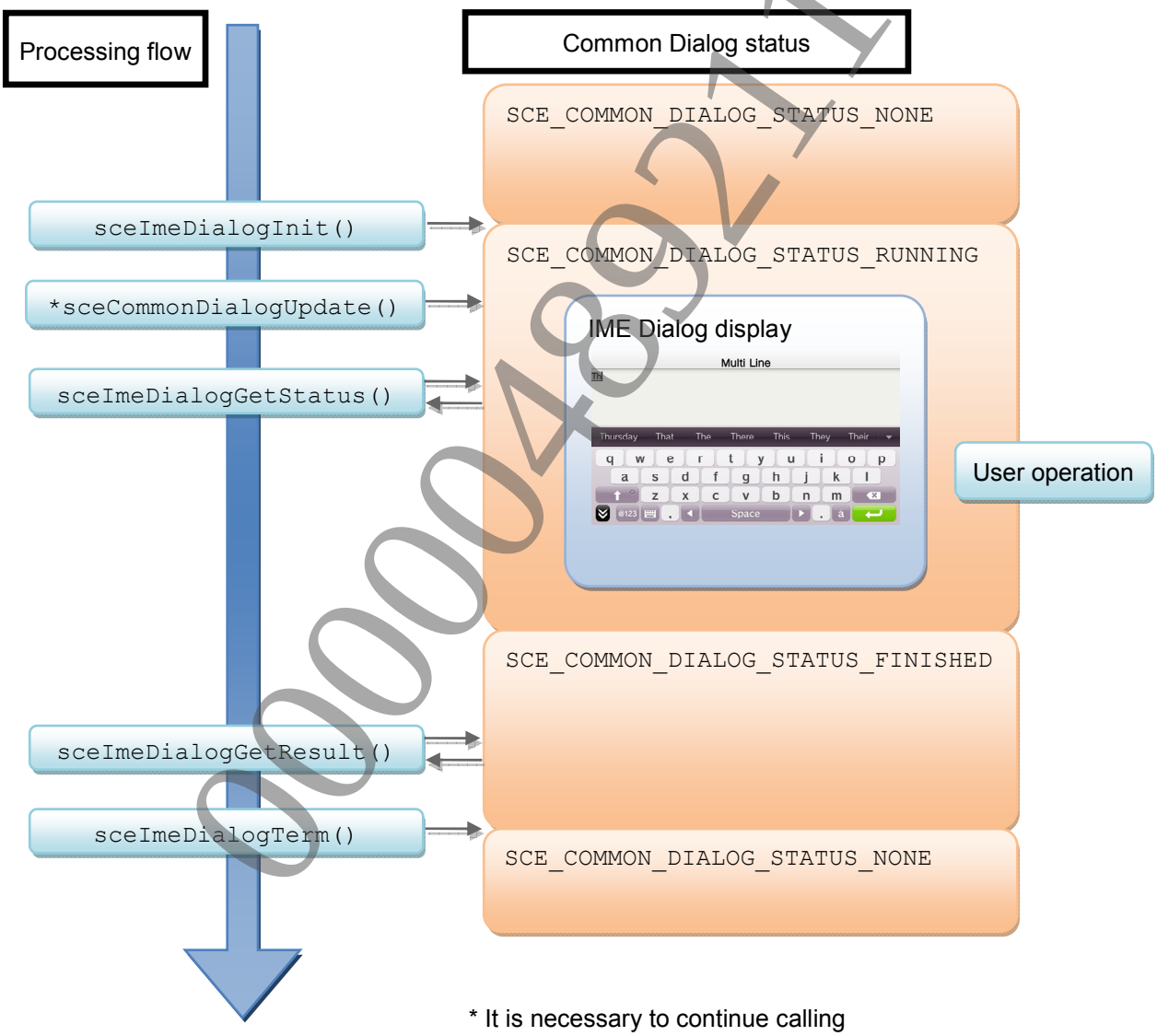
2 IME Dialog Library Usage Procedure

Basic Processing Procedure

This section explains the basic procedure for calling the IME Dialog library. The following is an overview of the processing flow.

(1) Set the parameters in the `SceImeDialogParam` type variable.
(2) Initialize the library.
(3) Wait for the response from the dialog.
(4) Retrieve call results.
(5) Terminate.

Figure 3 Basic Processing Procedure



* It is necessary to continue calling `sceCommonDialogUpdate()` at every frame during `Status≠SCE_COMMON_DIALOG_STATUS_NONE`.

Procedure for Calling IME Dialog

First, prepare an `SceImeDialogParam` type variable, and set the necessary parameters after performing initialization with `sceImeDialogParamInit()`.

(1) Library Initialization

Perform initialization by calling `sceImeDialogInit()`. Specify the `SceImeDialogParam` type variables set above as arguments.

(2) Retrieving Operation Status

Perform polling at every frame for the operation status of IME Dialog by calling `sceImeDialogGetStatus()`.

(3) Retrieving Call Results

When IME Dialog is closed, causing operation status to change to `SCE_COMMON_DIALOG_STATUS_FINISHED`, it is possible to retrieve results with `sceImeDialogGetResult()`. The input character string is stored in *`inputTextBuffer`* set with `SceImeDialogParam`.

(4) Termination

After retrieving the results, perform termination processing by calling `sceImeDialogTerm()`. By so doing, the resources allocated at the time of initialization will be freed.

Process Abort

If urgently interrupting Message Dialog display from the application, when the application is terminated, etc., call `sceImeDialogAbort()`. The dialog will be terminated immediately, and operation status will change directly to `SCE_COMMON_DIALOG_STATUS_FINISHED` without transition to operation substatus.

Main APIs Used in Basic Processes

API	Description
`SceImeDialogParam`	Parameter structure such as buffer, mode settings, etc.
`sceImeDialogParamInit()`	Initializes parameter structure.
`sceImeDialogInit()`	Initializes library.
`sceImeDialogTerm()`	Terminates library.
`sceImeDialogGetStatus()`	Retrieves operation status.
`sceImeDialogGetResult()`	Retrieves call results.

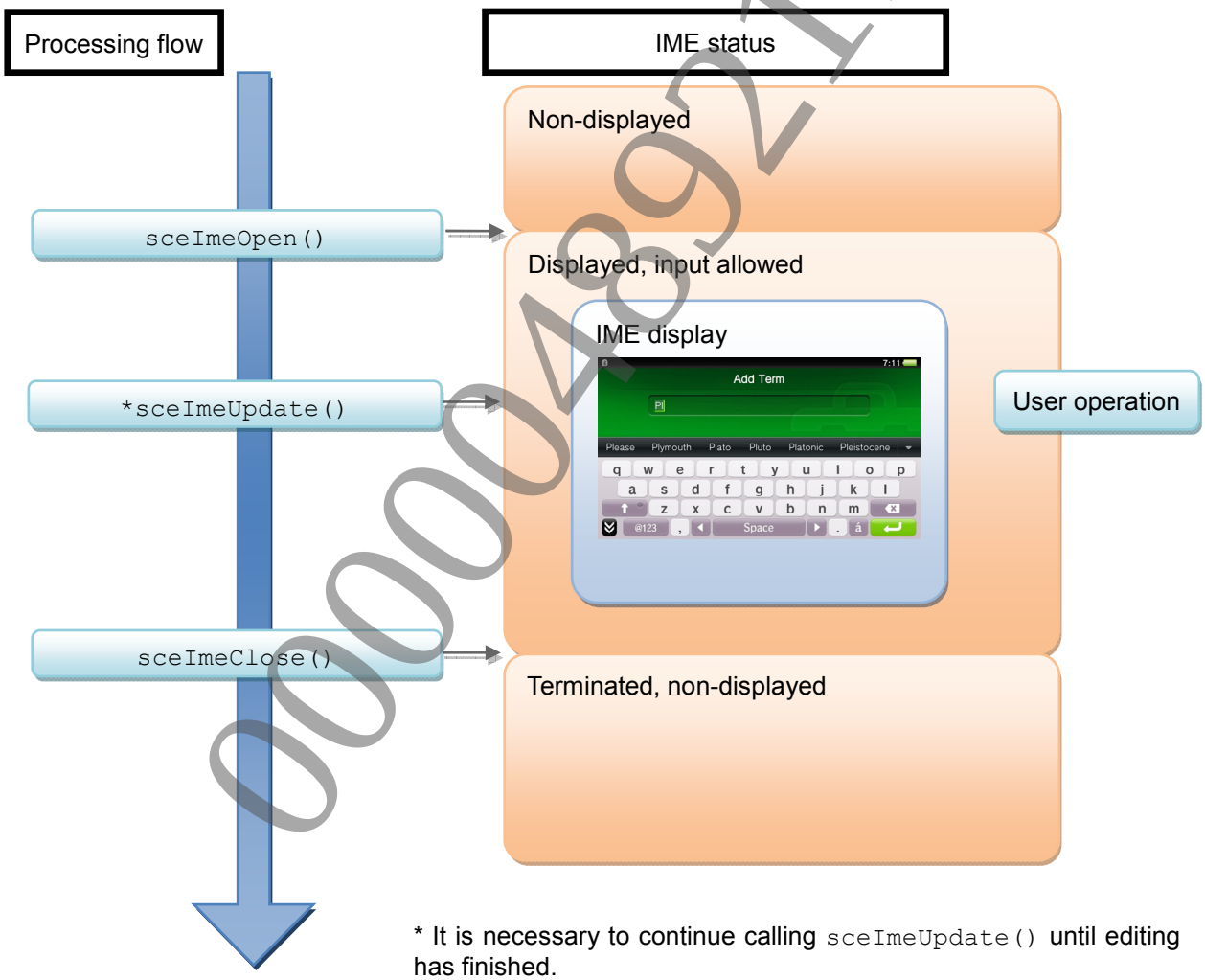
3 libime Usage Procedure

Basic Processing Procedure

This section explains the basic procedure for calling libime. The following is an overview of the processing flow.

(1) Load the libime module.
(2) Set the parameters in `SceImeParam` type variables.
(3) Start libime.
(4) Check IME events and perform the necessary processes.
(5) Notify status to IME based on the results of user operations and event processing.
(6) Repeat (4) to (5).
(7) Terminate libime in accordance with event content, user operations, etc.

Figure 4 Basic Processing Procedure



* It is necessary to continue calling `sceImeUpdate()` until editing has finished.

Procedure for Calling libime

(1) Loading the Module

The libime module is loaded with the API of libsysmodule. Specify `SCE_SYSMODULE_IME` and call `sceSysmoduleLoadModule`.

First, prepare an `SceImeParam` type variable and set the necessary parameters after performing initialization with `sceImeParamInit()`.

(2) Setting the Parameters

Set the IME configuration to be used in `SceImeParam`.

(3) Starting libime

Use the `SceImeParam` type variables set above as arguments and call `sceImeOpen()`.

(4) IME Event Processing

Call `sceImeUpdate()` at every frame and check for the presence of IME events. If IME events are present, the handler set in the `SceImeParam` type initialization parameter is called.

(5) Status Notification to IME

The current status must be notified to IME in accordance with processing of IME events and user operations.

If preedit position has been changed, its geometrical information must be notified with `sceImeSetPreeditGeometry()`.

If the caret's position has been changed, its geometrical information must be notified with `sceImeSetCaret()`. Also, if the caret's position has changed by additional operations other than IME events - for example, if the caret's position has been changed by the user's touch gesture - the caret's position within the character string must be notified to IME with `sceImeSetCaret()`.

(6) libime Termination

If, based on the user's operation of IME, etc., text input is no longer necessary, close libime by calling `sceImeClose()`. The input text is stored in the buffer set with the initialization parameters.

Main APIs Used in Basic Processes

API	Description
`SceImeParam`	Parameter structure of buffer, mode settings, etc.
`sceImeParamInit()`	Initializes parameter structure.
`sceImeOpen()`	Starts libime.
`sceImeClose()`	Terminates libime.
`sceImeUpdate()`	Updates libime and checks for the presence of events.
`sceImeSetCaret()`	Sets caret parameters.
`sceImeSetPreeditGeometry()`	Sets geometrical information of preedit.

About IME Event Processing

Event information notified from IME is stored in the `SceImeEvent` structure. The structure is composed of the *`id`* indicating the type of event, and the union *`param`*, where the information corresponding to the type of event is stored. Below are listed and explained event types and the members of the union that are used for each type.

IME Event List

id	*param*	Description
SCE_IME_EVENT_OPEN	*rect*	IME has been opened.
SCE_IME_EVENT_UPDATE_TEXT	*text*	Edited character string has been updated.
SCE_IME_EVENT_UPDATE_CARET	*caretIndex*	Caret position has been updated.
SCE_IME_EVENT_CHANGE_SIZE	*rect*	IME area has been changed.
SCE_IME_EVENT_PRESS_CLOSE	None	Close button has been pushed.
SCE_IME_EVENT_PRESS_ENTER	None	Enter button has been pushed.

Below is a detailed description of each event.

SCE_IME_EVENT_OPEN

This event is issued immediately before IME is loaded in the memory and input is allowed. *`rect`* stores the information of the rectangular area where IME is displayed. Adjust the area where the text being edited is drawn to the area used by IME.

SCE_IME_EVENT_UPDATE_TEXT

This event is issued when the text being edited is updated. *`text`* stores the information related to the text being edited and the caret. The character string being edited is stored in the *`str`* member of *`text`*. Preedits from *`preeditIndex`* to *`preeditLength`* length must be placed inside a semi-transparent rectangle when drawn, or similar means must be used so that the user may understand that they are preedits. Also, when the caret's position is changed, `sceImeSetCaret()` must be called to notify the caret's geometrical information to IME.

`editIndex` stores the position where the text has been edited before this event occurs. *`editLengthChange`* stores the changes of the text length made through the edit operation. For example, the value 3 will be stored in *`editLengthChange`* if three characters are added upon finalizing the preedits, and similarly, the value -1 will be stored when one character is deleted using backspace.

SCE_IME_EVENT_UPDATE_CARET

This event is issued when the caret's position is updated. The caret's position in the text is stored in *`caretIndex`*. `sceImeSetCaret()` must be called to notify the caret's geometrical information to IME.

SCE_IME_EVENT_CHANGE_SIZE

This event is issued when the rectangular area where IME is displayed is changed. Updated information of the rectangular area is stored in *`rect`*. Adjust the area where the text being edited is drawn to the area used by IME.

SCE_IME_EVENT_PRESS_CLOSE

This event is issued when the close button is pushed. Use the text last retrieved with `SCE_IME_EVENT_UPDATE_TEXT` as the result and immediately terminate libime by calling `sceImeClose()`.

SCE_IME_EVENT_PRESS_ENTER

This event is issued when the enter button is pushed. Perform the next action or ignore after calling `sceImeClose()`, as appropriate.

Pseudocode Sample

```
// Setting of IME parameters

ImeParam param;
sceImeParamInit(&param);

// omitted
param.handler = &OnImeEvent;
// omitted

// Starting IME
ret = sceImeOpen(&param);
if (ret < 0)
    ERROR;

// omitted
void MainLoop()
{
     // omitted
     while (isAlive)
     {
         // omitted
         ret = sceImeUpdate();
         if (ret < 0)
             ERROR;
         // omitted
     }
     // omitted
}

// Definition of event handler
void OnImeEvent(void *arg, const SceImeEvent *e)
{
    switch(e->id)
    {
        case SCE_IME_EVENT_OPEN:
            SetCaretVisible(true);
            AdjustTextBoxPosition(e->param.rect);
            break;

        case SCE_IME_EVENT_CHANGE_SIZE:
            AdjustTextBoxPosition(e->param.rect);
            break;

        case SCE_IME_EVENT_UPDATE_TEXT:
            CopyToDisplayText(e->param.text.str);
            SetPreeditArea(e->param.text.preeditIndex,
                e->param.text.preeditLength);
            SetCaret(e->param.text.caretIndex);
            break;

        case SCE_IME_EVENT_UPDATE_CARET:
            SetCaret(e->param.caretIndex);
            break;

        case SCE_IME_EVENT_PRESS_ENTER:
            SetEditing(false);
            sceImeClose();
            DoNextAction();
```

```
            break;

        case SCE_IME_EVENT_PRESS_CLOSE:
            EndEdit();
            sceImeClose();
            break;
    }
}

void SetCaret(SceUInt32 caretIndex)
{
    // omitted
    SceImeCaret caret;
    caret.index = caretIndex;
    GetCaretGeometryFromIndex(&caret, caretIndex);
    sceImeSetCaret(&caret);
    // omitted
}

void OnTouch()
{
     // omitted
     if (IsTextBoxTouched())
     {
          caretIndex = GetCaretIndexFromTouchPoint();
          SetCaret(caretIndex);
     }
     // omitted
}

void SetPreeditArea(SceUInt32 index, SceUInt32 len)
{
    // omitted
    SceImePreeditGeometry preedit;
    GetPreeditGeometryFromIndex(&preedit, index);
    sceImeSetPreeditGeometry(&preedit);
    // omitted
}
```

4 Specification Details

Startup Parameter

Functions provided by IME can be set through parameters that are given at the time of startup. Some parameters are shared by the IME Dialog library and libime. The settings of the shared parameters are explained below.

Input Language

IME has a function allowing it to handle multiple languages simultaneously. The input languages to be used are set by the user through the system menu, but the application can limit or specify those input languages.

Settings related to input language are performed with the members *supportedLanguages* and *languagesForced* of the initialization structure.

supportedLanguages specifies the languages that can be used from IME through logical OR of SCE_IME_LANGUAGE_* defined by the header. Setting to 0 causes all languages to be specified. If *languagesForced* is set to SCE_FALSE, the languages that can actually be used from IME will be the set intersection of the languages set in *supportedLanguages* and those set from the system menu. If the set intersection is empty, input is enabled for all languages set in *supportedLanguages*.

If *languagesForced* has been set to SCE_TRUE, it will be possible to use all languages set in *supportedLanguages*, irrespective of system settings.

Input Type

By setting an input type suitable for the type of text to be input, it is possible to change the UI provided by IME. This is set with the member *type* of the initialization structure. Currently, the following types are available. Additions will be provided in future updates.

type	Description
SCE_IME_TYPE_DEFAULT	Setting for inputting regular texts.
SCE_IME_TYPE_BASIC_LATIN	Setting for inputting character strings that only contain alphanumeric characters and symbols, such as user name, password etc.
SCE_IME_TYPE_NUMBER	Setting for inputting numbers
SCE_IME_TYPE_EXTENDED_NUMBER	Setting for inputting integers and floating points
SCE_IME_TYPE_URL	Setting for inputting URL
SCE_IME_TYPE_MAIL	Setting for inputting an email address

Input Options

A parameter to modify IME settings that cannot be set with input type. It is set in the member *option* of the initialization structure. For details, refer to the following.

- The "SceImeParam" section in the "libime Reference" document

Input Character Set

The character set that can be input with IME is limited by input types and input languages. Inputtable character sets for each input type and input language are described below.

SCE_IME_TYPE_DEFAULT

Inputtable character sets for each language are as follows.

Language	Inputtable Character Set
Danish	Latin Extended-A
German	Latin Extended-A
English (United States)	Latin Extended-A
Spanish	Latin Extended-A
French	Latin Extended-A
Italian	Latin Extended-A
Dutch	Latin Extended-A
Norwegian	Latin Extended-A
Polish	Latin Extended-A
Portuguese (Portugal)	Latin Extended-A
Russian	Cyrillic
Finnish	Latin Extended-A
Swedish	Latin Extended-A
Japanese	CP932
Korean	Hangul Compatibility Jamo, Hangul Syllables
Chinese (simplified)	GB18030
Chinese (traditional)	GB18030
Portuguese (Brazil)	Latin Extended-A
English (United Kingdom)	Latin Extended-A
Turkish	Latin Extended-A

In addition to the above, the following character sets can be input for all languages:

Latin, Latin-1 Supplement, U+2018, U+2019, U+201A, U+201C, U+201D, U+201E, U+2039, U+203A, U+20A9, U+20AC, U+2116

There is a possibility that the inputtable characters may increase with the future enhancement. Therefore, such as in the case where a character that is not included in the font set is input, applications are required to handle the situation by, for example, displaying a font representing the invalid character so as not to disrupt the application progress.

SCE_IME_TYPE_BASIC_LATIN

The inputtable characters are from U+0020 to U+007E.

SCE_IME_TYPE_NUMBER

The inputtable characters are from U+0030 to U+0039.

SCE_IME_TYPE_EXTENDED_NUMBER

The inputtable characters are from U+0030 to U+0039 and from U+002C to U+002E.

SCE_IME_TYPE_URL

Characters that can be entered are the same as `SCE_IME_TYPE_DEFAULT`.

SCE_IME_TYPE_MAIL

Characters that can be entered are the same as `SCE_IME_TYPE_BASIC_LATIN`.

5 Precautions

- libime cannot be used at the same time as the Common Dialog library.
- Text input during use of libime is limited to input via the touch panel.