Common Dialog Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Features

Common Dialog is a library that provides various high-level features common to the applications. Each feature has the system to dialog with end users and operates independently. The following dialog systems are available for this SDK release.

- Message Dialog
- Save Data Dialog
- Network Check Dialog
- IME Dialog
- Photo Import Dialog
- Photo Review Dialog
- Store Checkout Dialog
- NP Friend List Dialog
- NP Friend List2 Dialog
- NP Profile Dialog
- NP Message Dialog
- Trophy Setup Dialog
- Camera Import Dialog
- Facebook® Permission Dialog
- Cross-Controller Dialog
- "near" Dialog utility
- Tw Dialog
- Invitation Dialog
- Game Custom Data Dialog
- Video Import Dialog

Individual document is prepared for each dialog. For explanations of the usage procedure and the specifications, refer to the document.

Only the specifications common to all dialogs are described in this document.

Files

The following files are required in order to use Common Dialog at least.

File Name	Description
common_dialog.h	Header file
libSceCommonDialog_stub.a	Stub library file

Additional header files and stub library files will be required according to the dialog to be used.

Embedding into a Program

Include common_dialog.h in the source program.

The Common Dialog library can be linked in the PRX format only. Link libSceCommonDialog_stub.a to use Common Dialog. The PRX module is stored in storage managed by the system software and is automatically loaded when the process starts up.

Reference Materials

Refer to the following documents to use the Common Dialog library.

On common functions for Common Dialog

- Common Dialog Reference

On Message Dialog

- Message Dialog Overview

On Save Data Dialog

- Save Data User's Guide
- Save Data Dialog Overview

On Network Check Dialog

- Network Overview

On IME Dialog

- IME Overview

On Photo Import Dialog

- Photo Import Dialog Overview

On Photo Review Dialog

- Photo Review Dialog Overview

On Store Checkout Dialog

- Store Checkout Dialog Overview

On NP Friend List Dialog

- NP Friend List Dialog Overview

On NP Friend List2 Dialog

- NP Friend List2 Dialog Overview

On NP Profile Dialog

- NP Profile Dialog Overview

On NP Message Dialog

- NP Message Overview

On Trophy Setup Dialog

- NP Trophy Library Overview

On Camera Import Dialog

- Camera Import Dialog Overview

On Facebook Permission Dialog

- NP SNS Facebook Library Overview

On Cross-Controller Dialog

- Cross-Controller Dialog Overview

On "near" Dialog Utility

- near Dialog Utility Overview

On Tw Dialog

- Tw Dialog Overview

On Invitation Dialog

- InvitationDialog Library Overview

On Game Custom Data Dialog

- GameCustomDataDialog Library Overview

On Video Import Dialog

- Video Import Dialog Overview

2 Usage Procedure

Operation Status and Transition

Common Dialog has the following three operation statuses.

NONE (SCE_COMMON_DIALOG_STATUS_NONE)

This means the initial state. Common Dialog not in use is in this state.

RUNNING (SCE_COMMON_DIALOG_STATUS_RUNNING)

Common Dialog will be in this state during the execution.

FINISHED (SCE_COMMON_DIALOG_STATUS_FINISHED)

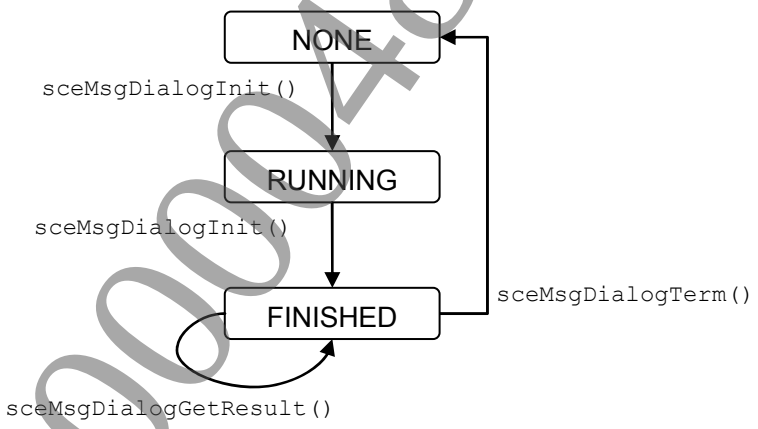
After completing the processing of Common Dialog, the state will transit to "FINISHED". The GUI of Common Dialog has disappeared from the screen. The resource can be released after the execution result of Common Dialog has been obtained by the application.

`SceCommonDialogStatus` is defined as an enumerated type for describing these states.

Each dialog has a function for obtaining the state without exception. In the case of Message Dialog, for example, the function is `sceMsgDialogGetStatus()`.

The following diagram shows the state transition of Common Dialog taking Message Dialog as an example.

Figure 1 The state transition of Common Dialog

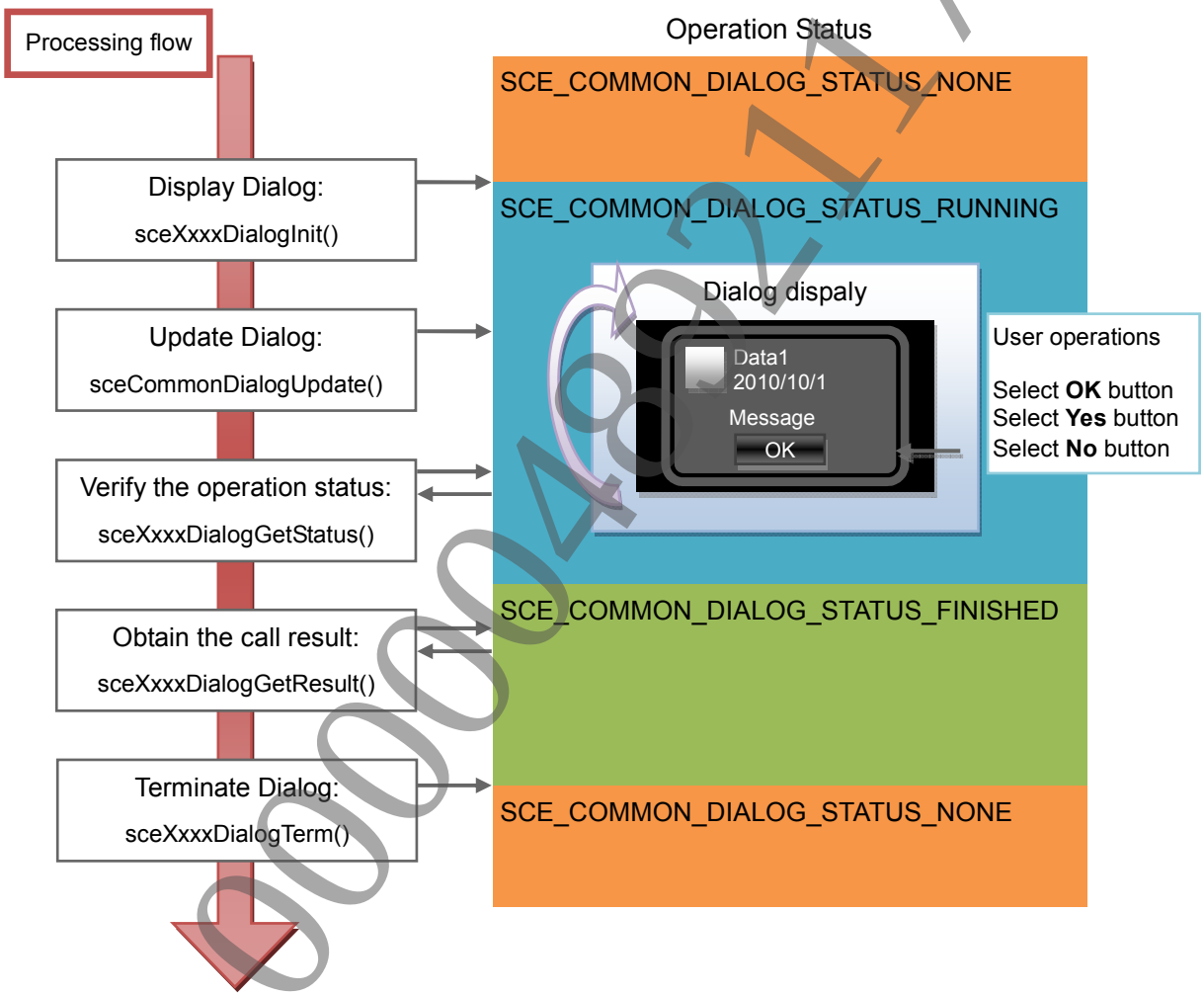


Basic Processing Procedure

The basic procedure for using Common Dialog is explained in this section. The processing procedure described below is common to all dialogs, while only Message Dialog is covered here as an example.

(1) Initialize libgxm
(2) Set the parameters
(3) Display Dialog
(4) Call the function for updating by each rendering frame
(5) Verify the operation status and obtain the call result
(6) Terminate Dialog

Figure 2 Basic Processing Procedure



(1) Initialize libgxm

Before using Common Dialog, call `sceGxmInitialize()` to initialize libgxm. If libgxm is not initialized, Common Dialog will fail to start.

(2) Set the parameters

Set the call parameter structure of Common Dialog. The behavior after starting up Common Dialog can be customized. Common Dialog generally provides functions that initialize parameter structures. For example, with Message Dialog, the parameter structure `SceMsgDialogParam` must be initialized using the function `sceMsgDialogParamInit()`. After using initialization functions to initialize parameter structures, set the required parameters.

In the following example, `SceMsgDialogParam` of Message Dialog is set and is customized to display the character string `"Hello, World!"` and an OK button.

```
SceMsgDialogParam param;
sceMsgDialogParamInit(&param);

SceMsgDialogUserMessageParam userMsgParam;
memset(&userMsgParam, 0x0, sizeof(userMsgParam));

param.mode = SCE_MSG_DIALOG_MODE_USER_MSG;
param.userMsgParam = &userMsgParam;
param.userMsgParam->msg = "Hello, World!";
param.userMsgParam->buttonType = SCE_MSG_DIALOG_BUTTON_TYPE_OK;
```

(3) Display Dialog

Call the function for starting up Common Dialog. In the case of Message Dialog, the function is `sceMsgDialogInit()`.

```
sceMsgDialogInit(&param);
```

After `sceMsgDialogInit()` is called completely, screen display of Message Dialog will start. operation A transition to `SCE_COMMON_DIALOG_STATUS_RUNNING` will occur for the operation status.

(4) Call the function for updating by each rendering frame

When the Common Dialog operation status is `SCE_COMMON_DIALOG_STATUS_RUNNING`, `sceCommonDialogUpdate()` must be called for every rendering frame. This function updates Common Dialog. If `sceCommonDialogUpdate()` is not called for every rendering frame, there is a possibility that the internal processing will not be performed correctly without the dialog being displayed.

```
SceCommonDialogUpdateParam updateParam;

updateParam.renderTarget.colorFormat      = SCE_GXM_COLOR_FORMAT_A8B8G8R8;
updateParam.renderTarget.surfaceType      = SCE_GXM_COLOR_SURFACE_LINEAR;
updateParam.renderTarget.colorSurfaceData = g_Dbuf[g_Dback];
updateParam.renderTarget.depthSurfaceData = NULL;
updateParam.renderTarget.width            = DISPLAY_WIDTH;
updateParam.renderTarget.height           = DISPLAY_HEIGHT;
updateParam.renderTarget.strideInPixels   = DISPLAY_STRIDE;

updateParam.displaySyncObject = g_Dsync[g_Dback];

sceCommonDialogUpdate(&updateParam);
```

The GUI of Common Dialog is directly rendered on the color surface of the application by `sceCommonDialogUpdate()`. Therefore, the information of the color surface to be used by the rendering frame must be given to this function as an augment.

It is possible to give depth surface to the argument of `sceCommonDialogUpdate()`. However, in this release of the SDK there are no dialogs that actually use depth surface. Specify NULL.

(5) Verify the operation status and obtain the call result

Call a function for obtaining the operation status to check the Common Dialog status, and if the result is that a transition to `SCE_COMMON_DIALOG_STATUS_FINISHED` has occurred for the operation status, obtain the result with a function that obtains call results. In the case of Message Dialog, the each operation can be performed by calling `sceMsgDialogGetStatus()` and `sceMsgDialogGetResult()` respectively.

```
SceCommonDialogStatus status = sceMsgDialogGetStatus();

if (status == SCE_COMMON_DIALOG_STATUS_FINISHED) {
        SceMsgDialogResult result;
        memset(&result, 0x0, sizeof(result));
        sceMsgDialogGetResult(&result);
```

It is recommended to verify the state of Common Dialog by each rendering frame.

(6) Terminate Dialog

After completing the use of Common Dialog, call the termination function to release the resource. In the case of Message Dialog, the function is `sceMsgDialogTerm()`.

```
sceMsgDialogTerm();
```

Functions

Among the functions provided by the Common Dialog library, the function which is common to all dialogs is shown below.

Function	Description
`sceCommonDialogGetWorkerThreadId()`	Gets worker thread identifier (thread ID) used by Common Dialog
`sceCommonDialogIsRunning()`	Gets information on whether or not Common Dialog is running
`sceCommonDialogUpdate()`	Updates Common Dialog

How to Flip the Frame Buffer

Common Dialog is directly rendered on the color surface of the application by `sceCommonDialogUpdate()`, but the rendering is not completed at the time of returning from this function. This function only issues commands for the GPU.

To correctly render Common Dialog, the frame buffer must be flipped after waiting for completion of the rendering of Common Dialog using display queue.

(1) Order the GPU to render the application

Set `SceGxmSyncObject` created using `sceGxmSyncObjectCreate()` for the *`fragmentSyncObject`* argument of `sceGxmBeginScene()` and call `sceGxmBeginScene()` to render the screen of the application.

Although the rendering performed by GPU is not completed at the time of returning from `sceGxmEndScene()`, go to the next step regardless of the situation.

(2) Order the GPU to render Common Dialog

Call `sceCommonDialogUpdate()` and specify `SceGxmSyncObject`, which is the same as used for the procedure (1) for the augment.

(3) Add to the display queue

Call `sceGxmDisplayQueueAddEntry()` and add the display queue of libgxm. Specify `SceGxmSyncObject`, which is the same as used for the procedures (1) and (2) for the augment *`newBuffer`*.

(4) Flip the frame buffer within the callback function of the display queue

Upon completing the rendering of the application and Common Dialog, the callback function of the display queue is called. Then flip the frame buffer by using `sceDisplaySetFrameBuf()`.

Note
Even if the frame buffer is flipped without waiting for completion of the rendering of Common Dialog, the rendering will appear normal in the majority of cases, but problems may occur, such as screenshots not getting captured correctly or displays getting corrupted owing to GPU overload. Therefore, be sure to flip the frame buffer in the manner described above.

Note
For details on the display queue, refer to the "Synchronization with Display" chapter of the "libgxm Overview" document.

3 Detailed Specifications

Language and Enter Button Assignment

The Common Dialog language will be the language set in **Language -> System Language** of the Settings application or will be displayed according to the language settings of the Sony Entertainment Network account. The Enter button assignment for Common Dialog is specified with Param File Editor. For the specification method, refer to the "Param File Editor User's Guide" document.

Button Control

Common Dialog is compatible with button control.

> **Note**
> Button control can be enabled by turning the following setting to On in the Settings application.
> **System > Control with Buttons on This System**

4 Notes

libgxm Initialization

Common Dialog is rendered using libgxm, therefore libgxm must be initialized before using Common Dialog. If a function that starts Common Dialog is called before initializing libgxm, the `SCE_COMMON_DIALOG_ERROR_GXM_IS_UNINITIALIZED` error will return.

Mutually Exclusive Use of libime or Common Dialog

Common Dialog and libime cannot be used simultaneously. While libime is being used, the `SCE_COMMON_DIALOG_ERROR_IME_IN_USE` error occurs if opening Common Dialog is tried such as by calling `sceMsgDialogInit()`.

Mutually Exclusive Use of Input Devices

While Common Dialog is running, the application cannot receive the input data by a user from hardware buttons and the touch panels. If hardware buttons are being pressed when Common Dialog terminates, button input information cannot be received until all buttons have been released.

Prohibition of Calling an API in libgxm Display-queue Callback

Any Common Dialog APIs cannot be called in the callback function of libgxm display queue. All the APIs will return the `SCE_COMMON_DIALOG_ERROR_ILLEGAL_CALLER_THREAD` error.

Supported Color Surface

Common Dialog supports limited configurations of color surfaces to render to. The surface must be configured as a frame buffer which the display service can output. Please refer to the "Display Service Overview" document for details of the configuration.

Delays in Reflecting Info bar Changes

While using Common Dialog, even if the application changes info bar status using the application manager's APIs, this will not be reflected. The effects will appear after Common Dialog is terminated

Dimmer

If necessary, the application may request Common Dialog to fill the screen area that is not occupied by its UI with a specified color. Concretely, this is specified by using `SceCommonDialogParam`'s member *`dimmer`*. However, the only colors that can actually be specified are opaque black (r,g,b,a)=(0,0,0,1) and transparent (r,g,b,a)=(0,0,0,0).

If dimmer color is not specified, nothing will be rendered in the screen area that is not occupied by the UI of Common Dialog (same as transparent). Whatever had been rendered by the application before Common Dialog will remain displayed.