Application Utility Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Library Overview

Purpose and Features

Application utility library is a utility that provides various features for game content (a subset of a file which contains applications such as save data and additional contents). The following are examples of the features that the application utility library provides.

- **Application event related features**
- **Save data related features**
- **Additional contents related features**
- **Background download related features**
- **Content data mount related features**
- **System parameter obtaining features**
- **Application parameter obtaining features**
- **Safe memory related features**
- **PlayStation®Store related features**
- **Internet Browser related features**
- **Save data for PSP™ (PlayStation®Portable) related features**

Files

The following files are required in order to use the application utility library.

File Name	Description
apputil.h	Header file
apputil/apputil_psp.h	Header file (for save data for PSP™ related features)
apputil/apputil_ext.h	Header file (for extended features)
libSceAppUtil_stub.a	Stub library file
libSceAppUtil_stub_weak.a	weak import stub library file
libSceAppUtilExt_stub.a	Stub library file (for extended features)
libSceAppUtilExt_stub_weak.a	weak import stub library file (for extended features)

When including apputil.h, various header files will be automatically included as well.

The application utility library can be linked in the PRX format only.

Link libSceAppUtil_stub.a or libSceAppUtil_stub_weak.a statically to use the application utility library. For details on the differences between the two stub library files, refer to the "Module (PRX) Management" chapter in the "Kernel Overview" document.

The PRX module is stored in storage managed by system software and the application utility module is loaded automatically when the application is booted up.

In addition, usage of some extended features requires libSceAppUtilExt_stub.a or libSceAppUtilExt_stub_weak.a to be statically linked and the PRX modules for extended features to be explicitly loaded after application boot.

Used Resources

The application utility library uses the following system resources.

Resource	Description
Footprint	Approx. 44 KiB when PRX is loaded 64 KiB when PRX for extended features is loaded
Processor time	Negligible
IO (Storage)	IO occurs when using save data related features. It depends on the number of files and the file size.

Sample Programs

Refer to the following files as sample programs that use the application utility library.

sample_code/system/api_savedata/fixed_basic/

This sample program uses the save data feature provided by the application utility library and performs basic and fixed save/load operation, along with API of Save Data Dialog.

For details on Save Data Dialog, refer to "Save Data Dialog Overview".

sample_code/system/api_savedata/list_basic/

This sample program uses the save data feature provided by the application utility. It performs list type save/load operations along with APIs of Save Data Dialog.

For details on Save Data Dialog, refer to "Save Data Dialog Overview".

sample_code/system/api_savedata/transfer/

This sample program uses the save data feature provided by the application utility. Save data of another title is read by using the feature in combination with APIs of Save Data Dialog.

For details on Save Data Dialog, refer to the "Save Data Dialog Overview" document.

sample_code/system/api_savedata/psp_basic/

This sample program uses save data for PSP™ related features provided by the application utility. APIs of Save Data Dialog are used in combination with the feature to load save data for PSP™.

For details on Save Data Dialog, refer to the "Save Data Dialog Overview" document.

sample_code/system/api_np_message_dialog/fixed_basic/

This sample program uses the application event related features provided by the application utility. It sends and receives messages with the invitation data and the messages with game data attached through PSN℠. For details on NP Message Dialog, refer to the "NP Message Overview" document.

sample_code/system/api_drm/

This sample program uses the additional contents related features, application parameter obtaining features and background download related feature provided by the application utility. It introduces implementation methods, such as, installing additional content from within the application, accessing the installed additional content, determining whether the content is upgradable or not, and mounting additional content of another title.

sample_code/network/api_near_util/

This is a sample program using the application event related features provided by the application utility. It shows the various usage methods of the "near" utility.

For details on the "near" utility, refer to the "near Utility Overview" document.

sample_code/system/api_webbrowser/

This is a sample program using the Internet Browser related features provided by the application utility. It shows the usage methods of the various features provided by the Internet Browser.

sample_code/system/api_livearea/application_bootup/

This is a sample program using the application event related features provided by the application utility. It describes how to start up an application from LiveArea™.

For details on LiveArea™, refer to the "LiveArea™ Specifications" document.

sample_code/system/api_screenshot/

This sample program uses application event related features provided by the application utility. It describes how to use screenshot capture notification events.

For details on the screenshot library, refer to the "Screenshot Library Overview" document.

sample_code/network/api_np/np_activity/

This sample program uses application event related features provided by the application utility. It describes how to use application startup from an activity.

For details on activities, refer to the "Activity System Overview" document.

sample_code/network/api_teleport/

This sample program uses application event related features provided by the application utility. The notification event handling method for when the application is started using the Teleport feature will be shown.

For details on Teleport feature, refer to the "Teleport Library Overview" document.

sample_code/system/api_invitation_dialog/

This sample program uses application event related features provided by the application utility. It indicates how to use the invitation dialog for sending/receiving invitations to session using **PSN℠**. For details on the invitation dialog, refer to the "Invitation Dialog Library Overview" document.

sample_code/system/api_game_custom_data_dialog/

This sample program uses application event related features provided by the application utility. It indicates how to use the game custom data dialog for sending/receiving game custom data using **PSN℠**. For details on the game custom data dialog, refer to the "GameCustomDataDialog Library Overview" document.

sample_code/system/api_videoimport_dialog/

This sample program uses the video mount feature in the content data mount related features provided by the application utility. It indicates how to obtain video file paths, etc. using the video import dialog. For details on the video import dialog, refer to the "Video Import Dialog Overview" document.

Reference Materials

Refer also to the following documents to use the save data related features of the application utility library.

- Application Development Process Overview
- Save Data User's Guide
- Save Data Dialog Overview

Functions

The following are the functions provided by the application utility library.

For details on these functions, refer to "Application Utility Reference".

For library

Function	Description
`sceAppUtilInit()`	Initializes library
`sceAppUtilShutdown()`	Terminates library

For application event

Function	Description
`sceAppUtilReceiveAppEvent()`	Receives the application event
`sceAppUtilAppEventParseNpInviteMessage()`	Parses the invitation message event parameters
`sceAppUtilAppEventParseNpAppDataMessage()`	Parses the event parameters of the message with game data attached
`sceAppUtilAppEventParseNearGift()`	Parses the "near" gift event parameters
`sceAppUtilAppEventParseLiveArea()`	Parses event parameters of application startup from LiveArea™
`sceAppUtilAppEventParseScreenShotNotification()`	Parses event parameters of a screenshot capture notification
`sceAppUtilAppEventParseNpActivity()`	Parses application startup event parameters from activity
`sceAppUtilAppEventParseTeleport()`	Parses application startup event parameters from the Teleport library
`sceAppUtilAppEventParseSessionInvitation()`	Parse session invitation event parameters
`sceAppUtilAppEventParseGameCustomData()`	Parse game custom data event parameters

For save data

Function	Description
sceAppUtilSaveDataSlotCreate()	Creates save data slot
sceAppUtilSaveDataSlotDelete()	Deletes save data slot
sceAppUtilSaveDataSlotSetParam()	Sets the save data slot parameters
sceAppUtilSaveDataSlotGetParam()	Gets the save data slot parameters
sceAppUtilSaveDataSlotSearch()	Searches save data slots
sceAppUtilSaveDataDataSave()	Saves the save data
sceAppUtilSaveDataDataRemove()	Deletes save data
sceAppUtilSaveDataGetQuota()	Gets maximum capacity and currently used space of save data
sceAppUtilSaveDataMount()	Mounts the save data directory
sceAppUtilSaveDataUmount()	Unmounts the save data directory

For background download

Function	Description
sceAppUtilBgdlGetStatus()	Gets background download list status

For additional contents

Function	Description
sceAppUtilDrmOpen()	Opens additional contents
sceAppUtilDrmClose()	Closes additional contents
sceAppUtilAddcontMount()	Mounts the additional content root directory
sceAppUtilAddcontUmount()	Unmounts the additional content root directory

For content data mount

Function	Description
sceAppUtilPhotoMount()	Mounts the photo data directory
sceAppUtilPhotoUmount()	Unmounts the photo data directory
sceAppUtilMusicMount()	Mounts the music data directory
sceAppUtilMusicUmount()	Unmounts the music data directory
sceAppUtilExtVideoMount()	Mounts the video data directory (loading the module for extended features is required)
sceAppUtilExtVideoUmount()	Unmounts the video data directory (loading the module for extended features is required)

For system parameter obtaining feature

Function	Description
sceAppUtilSystemParamGetInt()	Gets system parameters (integer values)
sceAppUtilSystemParamGetString()	Gets system parameters (character strings)

For application parameter obtaining feature

Function	Description
sceAppUtilAppParamGetInt()	Gets application parameters (integer values)

For safe memory

Function	Description
sceAppUtilSaveSafeMemory()	Saves data in a buffer into the safe memory
sceAppUtilLoadSafeMemory()	Reads the data saved in the safe memory into a buffer

For PlayStation®Store

Function	Description
sceAppUtilStoreBrowse()	Starts up the Title Store application

For Internet Browser

Function	Description
`sceAppUtilLaunchWebBrowser()`	Starts up the Internet Browser application
`sceAppUtilAddCookieWebBrowser()`	Writes out cookie data to be read by the Internet Browser
`sceAppUtilResetCookieWebBrowser()`	Initializes cookie data to add

For Save Data for PSP™

Function	Description
`sceAppUtilPspSaveDataGetDirNameList()`	Gets a list of save data for PSP™
`sceAppUtilPspSaveDataLoad()`	Loads save data for PSP™

2 Basic Usage Procedure

Basic processing procedure of the application utility library is described below.

(1) Load the module

The application utility library is provided as the PRX module.

The application utility module is loaded automatically when the application is started up, and is unloaded automatically when the application is terminated.

(2) Initialize the library

In order to use the application utility library, the library needs to be initialized by calling the `sceAppUtilInit()`. Set appropriate parameters for the `SceAppUtilInitParam` structure and call the function only once when the application is started up. Also, specify the `SceAppUtilBootParam` structure in the second argument as structure for obtaining boot-up parameters. All of the following processing of the `sceAppUtilXxx()` return errors if the library is not initialized with the `sceAppUtilInit()`.

```
/* Set the parameters to be passed to initialization function sceAppUtilInit()
*/
SceAppUtilInitParam initParam;
SceAppUtilBootParam bootParam;
memset(&initParam, 0, sizeof(SceAppUtilInitParam));
memset(&bootParam, 0, sizeof(SceAppUtilBootParam));

/* Initialize the library */
ret = sceAppUtilInit( &initParam, &bootParam );
```

(3) Use the library

Execute arbitrary processing including application event feature, save data feature, mount related features, system parameter obtaining feature, etc. using `sceAppUtilXxx()`.

(4) Terminate the library

The library is automatically terminated when the application is terminated.

(5) Unload the module

When the application is terminated, the application utility module will be unloaded automatically.

3 Provided Features

The features provided by the application utility library are listed below.

Application Event Related Features

Below is an explanation of the application event related features provided by the application utility library. It is possible to parse the content of the received application event by using these features together with the application status obtaining API `sceAppMgrGetAppState()` provided by the application manager.

When receiving the application event, the application will obtain the type and content of the application event through `sceAppUtilReceiveAppEvent()` first. Then, the application will properly parse the content of the received application event by calling the corresponding `sceAppUtilAppEventParseXxx()` based on the received application event notification type, and return the result by storing it into the structure passed in the second argument of the same API.

For example, the procedure for parsing the parameters when an invitation message event is received is as follows:

```
/* Obtain the application status */
SceAppMgrAppState appState;
memset(&appState, 0, sizeof(SceAppMgrAppState));

ret = sceAppMgrGetAppState(&appState);

/* When obtainment of the status completes successfully and an application event
is received */
if( (ret==SCE_OK) && (appState.appEventNum > 0))
{
    /* receive an application event */
    SceAppUtilAppEventParam eventParam;
    memset(&eventParam, 0, sizeof(SceAppUtilAppEventParam));

    ret = sceAppUtilReceiveAppEvent( &eventParam );

    /* When reception of the event completes successfully and
       the event notification type is the invitation message event */
    if( (ret==SCE_OK) &&
        (eventParam.type == SCE_APPUTIL_APPEVENT_TYPE_NP_INVITE_MESSAGE) )
    {
        SceAppUtilNpInviteMessageParam param;
        memset(&param, 0, sizeof(SceAppUtilNpInviteMessageParam));

        /* Parse the invitation message parameters and then store the result */
        ret = sceAppUtilAppEventParseNpInviteMessage( &eventParam, &param );
    }
}
```

When using application event related features, refer to the "Application Manager Overview" and "Application Manager Reference" documents.

Save Data Related Features

For explanations of the save data related features provided by the application utility library, refer to "Save Data User's Guide".

Additional Contents Related Features

Below is an explanation of the additional contents related features provided by the application utility library. In order to enable access to additional contents from the application, it is necessary to use the function for opening additional contents provided by the application utility library.

Enable access to additional contents by specifying the directory name and mount point of additional contents and calling the opening function.

When additional contents supported by that application are installed, a directory having as its name the label part of the contents ID of the additional contents package will be created below "addcont0:". "addcont0:" will be mounted automatically when the application is started up. In order to access each additional contents directory installed, opening/closing must be performed for each additional contents directory.

For example, if the name of the label part of the contents ID of the additional contents package is "0000111122223333", a directory named "addcont0:0000111122223333" will be created by installing these additional contents. Proceed as follows to access these additional contents:

```
/* Set additional contents directory */
SceAppUtilDrmAddcontId id;
memset(&id, 0, sizeof(SceAppUtilDrmAddcontId));
strncpy((char*)&id.data, "0000111122223333",
        sizeof(SceAppUtilDrmAddcontId) );

/* Open additional contents */
ret = sceAppUtilDrmOpen( &id, NULL );

/* Access additional contents */
fd = sceIoOpen( "addcont0:0000111122223333/data.dat", SCE_O_RDONLY, 0 );
```

The number of additional contents that can be opened simultaneously is 16. When the necessary processing for additional contents is terminated, call sceAppUtilDrmClose().

```
/* Close additional contents */
ret = sceAppUtilDrmClose( &id, NULL );
```

Moreover, additional content with an arbitrary product code can be loaded by mounting another title's additional content root directory as "addcont1:" and specifying this as the second argument of sceAppUtilDrmOpen() or sceAppUtilDrmClose(). However, the product code of that additional content must be set to param.sfo of the application, and the passcode set to that additional content's package must be known. For details regarding to the mounting of another title's additional content, refer to the "Application Development Process Overview" document.

```
/* Specify additional content to mount */
SceAppUtilTitleId titleId;
memset(&titleId, 0, sizeof(titleId));
strncpy((char*)titleId.data, "ABCD00001", sizeof(titleId)-1 );

/* Specify passcode of additional content (note the NULL terminator is excluded)
*/
SceAppUtilPassCode passCode;
memset(&passCode, 0, sizeof(passCode));
strncpy((char*)passCode.data, "AAAABBBBCCCCDDDDEEEEFFFFGGGGHHHH",
sizeof(passCode) );

/* Mount the additional content root directory */
ret = sceAppUtilAddcontMount(&titleId, &passCode);

/* Open additional content of another title */
SceAppUtilMountPoint mountPoint;
memset( &mountPoint, 0, sizeof(mountPoint) );
strncpy( (char*)mountPoint.data, "addcont1:", sizeof(mountPoint)-1 );
```

```
ret = sceAppUtilDrmOpen( &id, &mountPoint );
```

Only one additional content root directory can be mounted at a time. Once the necessary processing for additional content completes, call `sceAppUtilAddcontUmount()`.

```
/* Close additional content */
ret = sceAppUtilDrmClose( &id, &mountPoint );

/* Unmount additional content root directory */
ret = sceAppUtilAddcontUmount();
```

Background Download Related Features

This feature obtains the status of the background download list. It allows obtaining the number of contents that can be applied to that application on the background download list. Using this feature, it is possible to know the timing for calling the feature to apply the additional contents within the application that are offered from Store Checkout Dialog.

```
SceAppUtilBgdlStatus bgdlStat;
memset( &bgdlStat, 0, sizeof(bgdlStat) );
bgdlStat.type = SCE_APPUTIL_BGDL_STATUS_TYPE_ADDCONT;

/* Get background download list status */
ret = sceAppUtilBgdlGetStatus( &bgdlStat );
if( ret = 0 && bgdlStat.addcontNumReady > 0 ) {
        /* Apply add contents by StoreCheckoutDialog */
}
```

Content Data Mount Related Features

Below is an explanation of content data mount related features provided by the application utility library.

There is a restriction where the maximum number of virtual drives that can be mounted with each mount feature is limited to two. When accessing three or more types of content data, appropriately unmount a virtual drive that is no longer needed and then mount the next virtual drive.

Photo Mount Feature

In order to use this feature, call `sceAppUtilPhotoMount()`/`sceAppUtilPhotoUmount()`.

When an application calls `sceAppUtilPhotoMount()`, the photo data directory will be mounted as a "`photo0:`" virtual drive, and read access will be possible for photo data files.

After the necessary processing is complete, call `sceAppUtilPhotoUmount()` and unmount "`photo0:`". It is not possible to use the virtual drive "`photo0:`" after unmounting.

```
/* Mount the photo data directory */
ret = sceAppUtilPhotoMount();
...
/* File I/O processing using the virtual drive "photo0:" is enabled */
...
/* Unmount the photo data directory */
ret = sceAppUtilPhotoUmount();
```

During the time that an application performs read access to a photo data file using the virtual drive, Photos application or other applications may add or delete a photo data file.

It is possible to continue the read processing even when the open file is deleted. However, the file is actually deleted when it is closed, so opening the file will fail from the next time.

Do not search a file by opening the "photo0:" directory. Make sure to obtain the path to the photo data through Photo Import Dialog library as needed.

Regarding the photo data directory mount/unmount feature and the details on Photo Import Dialog library, also refer to the "Photo Import Dialog Overview" and "Photo Import Dialog Reference" documents.

Music Mount Feature

In order to use this feature, call `sceAppUtilMusicMount()/sceAppUtilMusicUmount()`.

When an application calls `sceAppUtilMusicMount()`, the music data directory will be mounted as a "`music0:`" virtual drive, and read access will be possible for music data files.

After the necessary processing is complete, call `sceAppUtilMusicUmount()` and unmount "`music0:`". It is not possible to use the virtual drive "`music0:`" after unmounting.

```
/* Mount the music data directory */
ret = sceAppUtilMusicMount();
...
/* File I/O processing using the virtual drive "music0:" is enabled */
...
/* Unmount the music data directory */
ret = sceAppUtilMusicUmount();
```

During the time that an application performs read access to a music data file using the virtual drive, Music application or other applications may add or delete a music data file.

It is possible to continue the read processing even when the open file is deleted. However, the file is actually deleted when it is closed, so opening the file will fail from the next time.

In this case, the latest file list can be obtained by searching music data files again, starting from the top directory of "music0:".

Likewise, with regard to a file added by another application, it is also possible to obtain the latest file list by searching music data files again, starting from the top directory of "music0:".

Note that directories under music0:/system/ are used by the system and access to such directories is prohibited.

Regarding the audio output from the BGM port, also refer to the "BGM Port Control System Call Overview" document.

Video Mount Feature

In order to use this feature, `sceAppUtilExtVideoMount()/sceAppUtilExtVideoUmount()` must be called with the application utility library module for extended feature loaded.

When an application calls `sceAppUtilExtVideoMount()`, the video data directory will be mounted as a "`video0:`" virtual drive, and read access will be possible for video data files.

After the necessary processing is complete, call `sceAppUtilExtVideoUmount()` and unmount "`video0:`". It is not possible to use the virtual drive "`video0:`" after unmounting.

```
/* Load the additional library module */
ret = sceSysmoduleLoadModule( SCE_SYSMODULE_APPUTIL_EXT );

/* Mount the video data directory */
ret = sceAppUtilExtVideoMount();
...
/* File I/O processing using the virtual drive "video0:" is enabled */
...
/* Unmount the video data directory */
ret = sceAppUtilExtVideoUmount();
```

```
/* Unload the additional library module */
ret = sceSysmoduleUnloadModule( SCE_SYSMODULE_APPUTIL_EXT );
```

During the time that an application performs read access to a video data file using the virtual drive, Videos application or other applications may add or delete a video data file.

It is possible to continue the read processing even when the open file is deleted. However, the file is actually deleted when it is closed, so opening the file will fail from the next time.

Do not search a file by opening the "`video0:`" directory. Make sure to obtain the path to the video data through the video import dialog library as needed.

Regarding the video data directory mount/unmount feature and the details on the video import dialog library, also refer to the "Video Import Dialog Overview" and "Video Import Dialog Reference" documents.

System Parameter Obtaining Features

Below is an explanation of the system parameter obtaining feature provided by the application utility library.

It is possible to obtain system parameters through `sceAppUtilSystemParamGetInt()`/`sceAppUtilSystemParamGetString()`, the APIs for obtaining the system parameters set to PlayStation®Vita.

For the obtainable system parameters, refer to the "Application Utility Reference" document.

For the various definitions of the system parameters, also refer to system_param.h.

For example, the procedure for obtaining the language settings currently set to PlayStation®Vita is as follows:

```
/* Obtain the language settings set to PlayStation(R)Vita */
SceInt32 language;
ret = sceAppUtilSystemParamGetInt( SCE_SYSTEM_PARAM_ID_LANG, &language );
```

Likewise, the procedure for obtaining the user name currently set to PlayStation®Vita is as follows:

```
/* Obtain the user name set to PlayStation(R)Vita */
SceChar8 buf[SCE_SYSTEM_PARAM_USER_NAME_MAXSIZE];
ret = sceAppUtilSystemParamGetString(
                    SCE_SYSTEM_PARAM_ID_USER_NAME,
                    &buf,
                    SCE_SYSTEM_PARAM_USER_NAME_MAXSIZE );
```

Application Parameter Obtaining Features

Below is an explanation of the feature for obtaining application parameters provided by the application utility library. Application parameters can be obtained by using the API `sceAppUtilAppParamGetInt()`, which obtains the application parameters set in each application.

For the obtainable application parameters, refer to the "Application Utility Reference" document.

For example, proceed as follows to obtain the SKU flag set in the application:

```
/* Obtain SKU flag */
SceInt32 skuflag;
ret = sceAppUtilAppParamGetInt( SCE_APPUTIL_APPPARAM_ID_SKU_FLAG, &skuflag );
```

Safe Memory Related Features

Below is an explanation of the safe memory related features provided by the application utility library.

Safe memory is a nonvolatile memory area applications can use. The size of the safe memory is defined with `SCE_APPUTIL_SAFEMEMORY_MEMORY_SIZE`.

The procedure for saving data into the safe memory is as follows.

```
SceChar8 buf[1024];
SceOff offset = 0;

/* Set data content a user wants to save */
buf = ...

/* Save the data into the safe memory */
ret = sceAppUtilSaveSafeMemory( buf, sizeof(buf), offset );
```

Similarly to the above, the procedure for reading the safe memory is as follows.

```
SceChar8 buf[1024];
SceOff offset = 0;

/* Read data from the safe memory */
ret = sceAppUtilLoadSafeMemory( buf, sizeof(buf), offset );

/* Use the content read from the safe memory */
buf = ...
```

For details on the safe memory features, also refer to the chapter "Safe Memory" in the "Save Data User's Guide" document.

PlayStation®Store Related Features

Below is an explanation of the PlayStation®Store related features provided by the application utility library. It is possible through these features to start up the Title Store application from a game, to browse and purchase the products displayed in the Title Store of PlayStation®Store and redeem the promotion code. Specify the categories or products to be displayed at the startup of the store application.

To start up the Title Store application with the products specified, call `sceAppUtilStoreBrowse()` as in the following procedures.

```
SceAppUtilStoreBrowseParam param;
param.type = SCE_APPUTIL_STORE_BROWSE_TYPE_PRODUCT2;
param.id = "IV0002-NPXS00004_00-0000111122223333";
ret = sceAppUtilStoreBrowse( param );
```

To start up the Title Store application with the categories specified, call `sceAppUtilStoreBrowse()` as in the following procedures.

```
SceAppUtilStoreBrowseParam param;
param.type = SCE_APPUTIL_STORE_BROWSE_TYPE_CATEGORY2;
param.id = "IV0002-NPXS00004_00";
ret = sceAppUtilStoreBrowse( param );
```

In order to display the promotion code input UI and redeem a promotion code, call `sceAppUtilStoreBrowse()` with the following procedure:

```
SceAppUtilStoreBrowseParam param;
param.type = SCE_APPUTIL_STORE_BROWSE_TYPE_PRODUCT_CODE2;
param.id = NULL;
ret = sceAppUtilStoreBrowse( param );
```

Call `sceAppUtilStoreBrowse()` with the following procedure to redeem the specified promotion code.

```
SceAppUtilStoreBrowseParam param;
param.type = SCE_APPUTIL_STORE_BROWSE_TYPE_PRODUCT_CODE2;
param.id = "ABCD-1234-EFGH";
ret = sceAppUtilStoreBrowse( param );
```

While the Title Store application is running, the game process will be suspended. When the Title Store application is terminated, the game process will be resumed. If additional contents are being accessed when the Title Store application is called, that call fails.

When a product is acquired through purchase or redemption of a promotion code, the store application and the Title Store application will issue a system event. It is possible to know whether any products have been acquired by monitoring system events. For details, refer to the "Application Manager Overview" and "Application Manager Reference" documents.

Internet Browser Related Features

Below is an explanation of the Internet Browser related features provided by the application utility library.

Startup Feature

Starting up the Internet Browser application from a game with these features allows you to use the webpage display feature and Internet search feature provided by the Internet Browser.

The basic usage procedure consists of setting the operation mode in *`launchMode`* of the `SceAppUtilWebBrowserParam` structure, giving the corresponding character string to the *`wbstr`* member and calling `sceAppUtilLaunchWebBrowser()`. In this way, the Internet Browser application will be started up in the type that has been set.

Specify the startup type by combining the startup type (`SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_*`) and the startup command type (`SCE_APPUTIL_WEBBROWSER_LAUNCH_CMD_*`) of the Internet Browser application.

Specifying Startup Type

The Internet Browser can be started up by specifying one of the following startup types for `sceAppUtilLaunchWebBrowser()`.

Normal Internet Browser

This Internet Browser is started up when the startup type is set to `SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_NORMAL`. This is the same Internet Browser as that started up by tapping the browser icon on the home screen. All features provided by an Internet Browser can be used with this type.

Modal Internet Browser

This Internet Browser is started up when the startup type is set to `SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_MODAL`. Unlike the normal Internet Browser, the behavior of this Internet Browser is such that it is closely linked with the calling application. When the modal Internet Browser is terminated, it returns to the calling application. Some features provided by the normal Internet Browser such as Internet search feature, bookmark feature and other setting features cannot be used with this Internet Browser.

Type	Internet Search	Bookmark	Window Management	Browser Settings
Normal Internet Browser	Yes	Yes	Yes	Yes
Modal Internet Browser	-	-	-	-

Specifying Startup Command Type

A specific feature can be executed when the Internet Browser application is started up by specifying one of the following startup command types for `sceAppUtilLaunchWebBrowser()`.

Webpage Display Feature (SCE_APPUTIL_WEBBROWSER_LAUNCH_CMD_OPENURL)

Call `sceAppUtilLaunchWebBrowser()` with the procedure shown below to start up the Internet Browser application specifying the webpage display feature.

The *`wbstr`* member is interpreted as the URL.

```
#define SAMPLE_URL "http://www.scedev.net"
SceAppUtilWebBrowserParam param;
param.wbstr = SAMPLE_URL;

param.wbstrLength = sizeof(SAMPLE_URL) - 1;
param.launchMode = SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_NORMAL |
                   SCE_APPUTIL_WEBBROWSER_LAUNCH_CMD_OPENURL;
ret = sceAppUtilLaunchWebBrowser(param);
```

Internet Search Feature (SCE_APPUTIL_WEBBROWSER_LAUNCH_CMD_SEARCH)

Call `sceAppUtilLaunchWebBrowser()` with the procedure shown below to start up the Internet Browser application specifying the Internet search feature.

The *`wbstr`* member is interpreted as the search keyword. This keyword's character code is UTF-8. In order to specify multiple search keywords, specify as a character string divided by space characters (space=0x20 or tab=0x09).

```
#define SAMPLE_SEARCH_STRING "PS Vita BROWSER"
SceAppUtilWebBrowserParam param;
param.wbstr = SAMPLE_SEARCH_STRING;
param.wbstrLength = sizeof(SAMPLE_SEARCH_STRING) - 1;
param.launchMode = SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_NORMAL |
                   SCE_APPUTIL_WEBBROWSER_LAUNCH_CMD_SEARCH;
ret = sceAppUtilLaunchWebBrowser(param);
```

Startup and Termination with SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_NORMAL

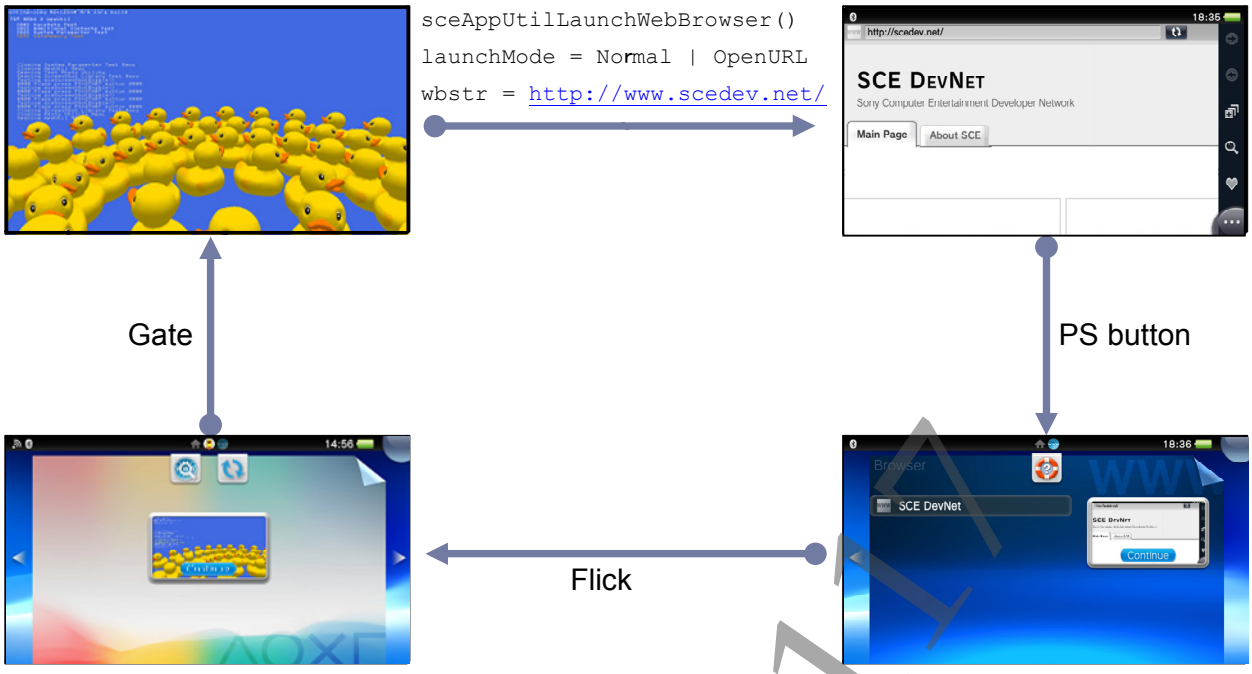
While the Internet Browser application is running, the game process will be suspended. In order to return to the game process, the user must be made to perform the following procedure:

(1) Press the PS button to return to the LiveArea™ of the Internet Browser application.

(2) Go to the game application's LiveArea™ by flicking.

(3) Tap the Gate of the LiveArea™ moved to in (2).

This sequence of operations is shown below:



Startup and Termination with SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_MODAL

While the Internet Browser application is running, the game process will be suspended. In order to return to the game process, the user must be made to perform the following procedure (A) or (B):

(A) Using PS button

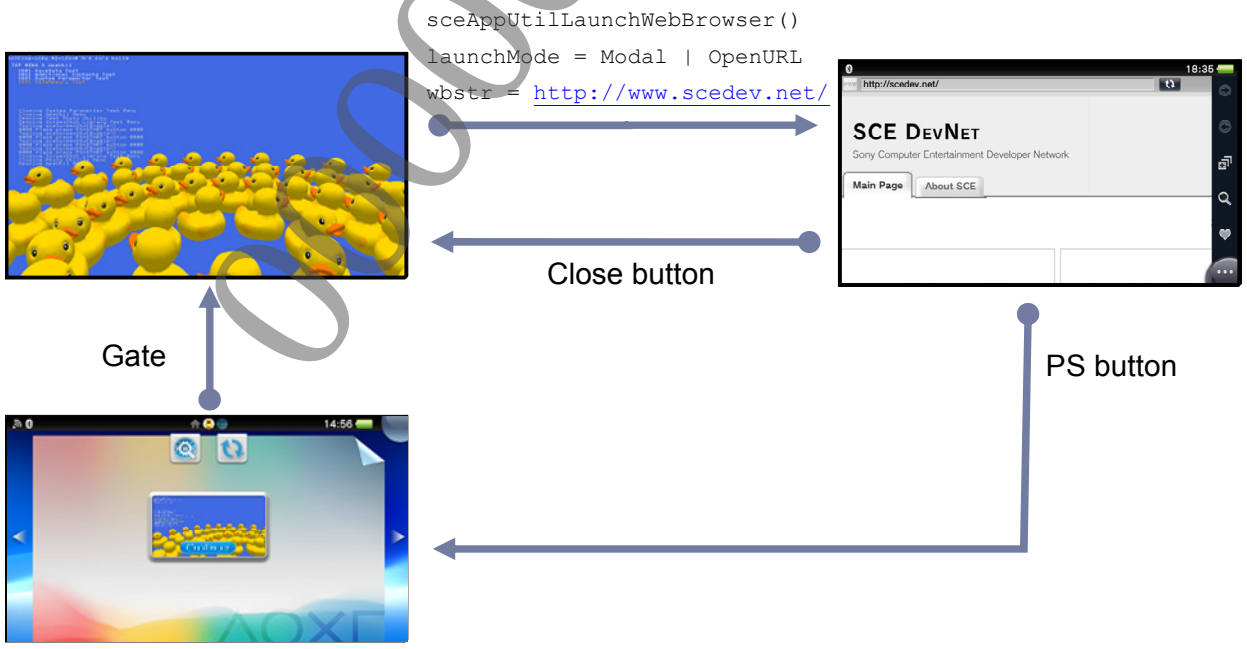
(1) Press the PS button to return to the LiveArea™ of the game process.

(2) Tap the gate of the LiveArea™ moved to in (1).

(B) Using Close button

(1) Return to the game application by tapping the Close button.

This sequence of operations is shown below:



Add Cookie Feature

Use this feature to write out a cookie from a game and to have it read by the Internet Browser after it starts up. A maximum of ten cookies can be written at once.

Initialization

`sceAppUtilResetCookieWebBrowser()` deletes all the cookies written by `sceAppUtilAddCookieWebBrowser()`.

Adding Processing

Call `sceAppUtilAddCookieWebBrowser()` as follows to write out cookies to be read by the Internet Browser.

Subsequently call `sceAppUtilLaunchWebBrowser()` to start up the Internet Browser and to read contents.

```
#define SAMPLE_URL "http://www.scedev.net"
#define SAMPLE_COOKIE_NAME "key"
#define SAMPLE_COOKIE_VALUE "value"

SceAppUtilWebBrowserAddCookieParam param;

/* 0-clear */
memset( &param, 0, sizeof(param) );

/* Write out cookies for the Internet Browser */

param.wbstr = SAMPLE_URL;
param.wbstrName = SAMPLE_COOKIE_NAME;
param.wbstrValue = SAMPLE_COOKIE_VALUE;

SceInt32 ret = sceAppUtilAddCookieWebBrowser( param );
```

Cookie Format

Cookies used by this system are based on the specifications of Netscape Communications (note: explanation of path differs) and RFC 6265.

A cookie set by `sceAppUtilAddCookieWebBrowser()` pseudo-executes "`Set-Cookie: NAME=VALUE; expires=DATE; domain=DOMAIN; path=PATH; secure`" (set by the response from the server) to realize an arbitrary cookie-saving processing.

The following is an explanation of the Set-Cookie header and member specifications of `SceAppUtilWebBrowserAddCookieParam`.

NAME=VALUE (Required)

- *`wbstrName`*
  Specify the name of the cookie. Use a character string excluding a semicolon, comma, and space. To specify a character string including the above prohibited characters, perform a URL-encode (for example) to convert to a character string that does not include the prohibited characters (there are no specific encode rules).
- *`wbstrValue`*
  Specify the value of the cookie.
  When a value is not specified, operation will be the same as when an empty character string is specified.

expires=DATE

- *wbstrExpires*
  Set the expiration date of the cookie. The expiration date is expressed with the following two methods. The latter is a format shown by a JavaScript Date object.
  Wdy, DD-Mon-YYYY HH:MM:SS GMT
  Wdy, DD Mon YYYY HH:MM:SS GMT
  The time zone must be GMT.
  When a value is not specified for `expires`, operation will be the same as when 0 is set and the cookie will be recognized as a session cookie.

Abbreviation	Meaning	Usable character strings
Wdy	Day of the week	Monday/Mon, Tuesday/Tue, Wednesday/Wed, Thursday/Thu, Friday/Fri, Saturday/Sat, Sunday/Sun
DD	Day	One to two digit number
Mon	Month	Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec
YYYY	Year	A four digit number
HH	Hour	A two digit number
MM	Minute	A two digit number
SS	Second	A two digit number

domain=DOMAIN

- *wbstrDomain*
  Specify the domain where the cookie will be valid. Domain match will be checked by a backward match with FQDN specified by this parameter. For example, when "`domain=.foo.com`" is specified, a match will be determined with a host named "`bar.foo.com`".
  Note that the domain specification must start with a period (.). When the domain is specified without a period, as in "`domain=foo.com`", a match will be determined with a host named "`bar.foofoo.com`".
  If a domain is not specified, the domain of the URL specified in *wbstr* will be set. Moreover, if a backward match cannot be determined between the domain specified in *wbstrDomain* and the domain specified in *wbstr*, the cookie will not be registered.
  In addition, do not specify a value that only consists of a public suffix - such as, "`.com`" or "`.co.jp`". For a list of public suffixes, refer to the following URL.
  http://publicsuffix.org/
  (The above reference destination has been confirmed as of February 18, 2015. Note that pages may have been subsequently moved or its contents modified.)

path=PATH

- *wbstrPath*
  Specify the path by which the cookie will be valid.
  If the path is not specified, the path of the URL specified in *wbstr* will be set.

secure

- *isSecure*
  Specify as a cookie that can only be used in HTTPS.
  If this is not specified, the cookie will be handled as *isSecure*=`false` and the specified cookie will be sent in both HTTP and HTTPS communications.

Save data for PSP™ Related Features

For save data for PSP™ related features provided by the application utility library, refer to the "Save Data User's Guide" document.