# near Compliant Application Development Process Overview



PlayStation®Vita Programmer Tool Runtime Library Release 3.50

# Table of Contents

# 1 Introduction

## Structure of this Document

The "near" system enables the use of the various services in games, such as, the distribution of item data among users and the search of other users who are near the owner.

This document gives an overview of the development process of applications using the "near" system in the following order.

- Overview of the "near" Utility/"near" Dialog Utility
  Chapter 2 describes the functions provided to game programs by the "near" utility and the "near" Dialog utility.
- Overview of the "near" Game SDK
  The "near" game SDK only includes client-level functions that are required for the development of processing needed in game development. Therefore, no function is provided to connect with the server, and communication with the server by the "near" application and the "near" Dialog utility is emulated on the Development Kit (DevKit) and Testing Kit (TestKit). The "near" game service operations on the retail unit and the "near" game service on the DevKit/TestKit are described in Chapter 3.
- Checking/Testing on a DevKit/TestKit
  For the purpose of improving quality, methods for checking the operation of game programs using the "near" utility/"near" Dialog utility are described in chapter 4.
- Implementation
  Chapter 5 provides an example of implementation envisaging use of the "near" game service in a game program.
- FAQ
  Frequently asked questions are answered in Chapter 6.

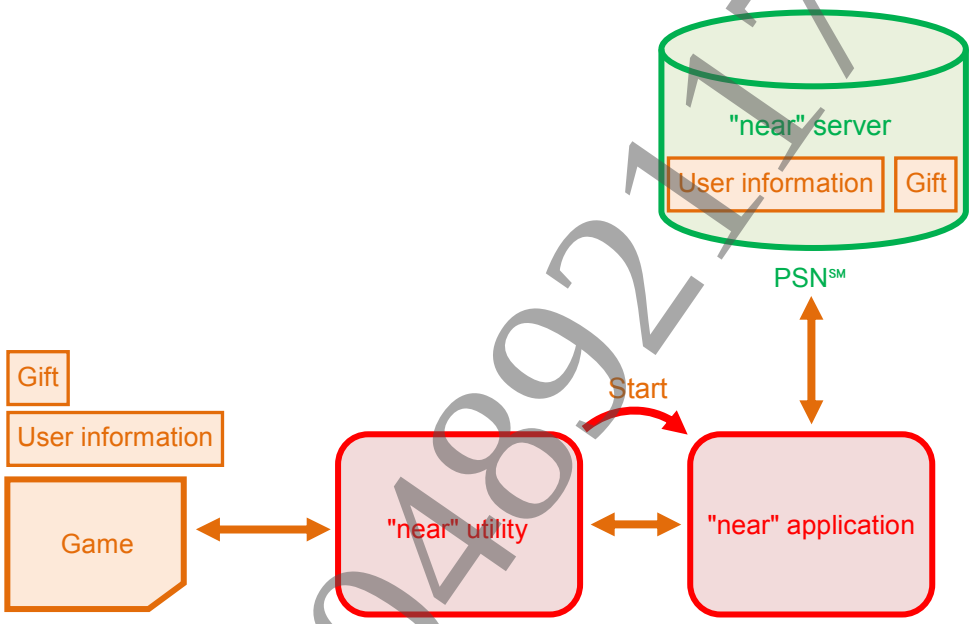
# 2 Overview of the "near" Utility/"near" Dialog Utility

This chapter provides an overview of the roles served by the "near" utility and "near" Dialog utility in the "near" system.

## Role of the "near" Utility

The "near" utility reads/writes data from/to the "near" memeory area managed by the system software based on game program requests. It also starts the "near" application given the request from a game. (Figure 1) For details on the "near" utility, refer to the "near Utility Overview" document.

**Figure 1 Role of the "near" Utility in the "near" System**



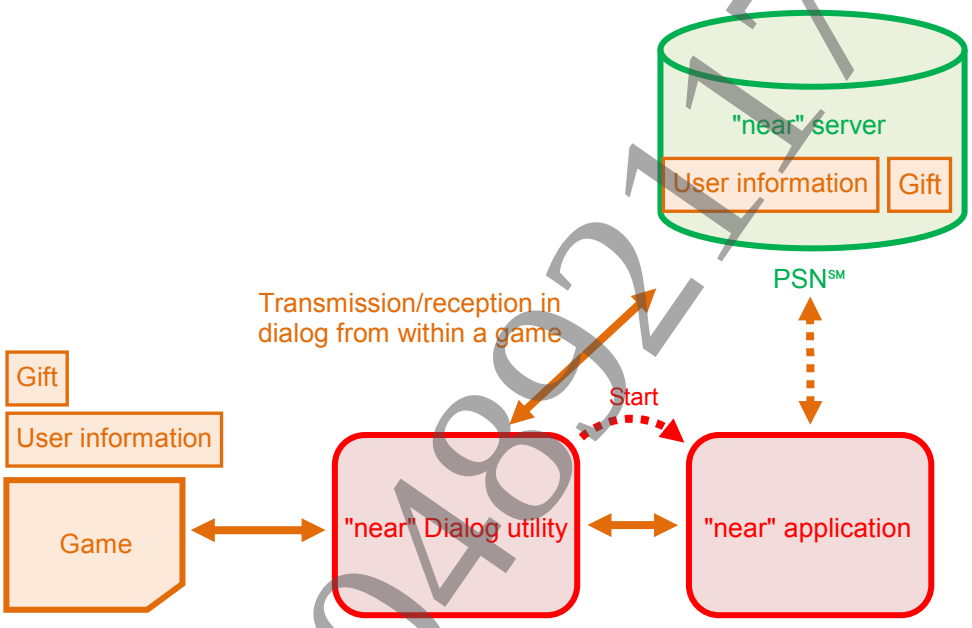
## Role of the "near" Dialog Utility

The "near" Dialog utility reads/writes data from/to the "near" memeory area managed by the system software based on game program requests.

Moreover, based on a game program request, it displays dialogs within a game to prompt the user to send information from PlayStation®Vita to the "near" server, and also to receive information from the "near" server.

The conditions for data to be exchanged between PlayStation®Vita and the "near" server by the "near" Dialog utility conform to the conditions to send/receive data within the "near" application. (Figure 2) For details on the "near" Dialog utility, refer to the "near Dialog Utility Overview" document.

**Figure 2 Role of the "near" Dialog Utility in the "near" System**




## Comparison of the Functions of the "near" Utility and "near" Dialog Utility

**Table 1 "near" Utility/"near" Dialog Utility Function Comparison**

| Feature | "near" Utility | "near" Dialog Utility |
|---|---|---|
| Setting gifts for distribution | Yes | Yes |
| Obtaining information of the discovered gift | Yes | Yes |
| Obtaining information of nearby users | Yes | Yes |
| Obtaining the usage status of the "near" application | Yes | Yes |
| Starting the "near" application | Yes | Yes |
| Updating "near" with displaying a dialog on the game screen | - | Yes |
| Receiving gifts with displaying a dialog on the game screen | - | Yes |

## Simultaneous Use of the "near" Utility and the "near" Dialog Utility

The "near" game service does not presume the simultaneous use of the "near" utility and the "near" Dialog utility. Do not program your application in such a way that, for example, a value obtained using the "near" utility must be passed to the "near" Dialog utility.

# 3 Overview of the "near" Game SDK

On the DevKit/TestKit, it is possible to verify operations emulating the "near" game service. In this chapter you will find an explanation of the operations that can be verified on the DevKit/TestKit by comparing them with PlayStation®Vita retail unit operations.

## Operation on the PlayStation®Vita Retail Unit

The "near" application and game programs using the "near" utility/"near" Dialog utility on the PlayStation®Vita retail units are shown in Figure 3 and Figure 4.

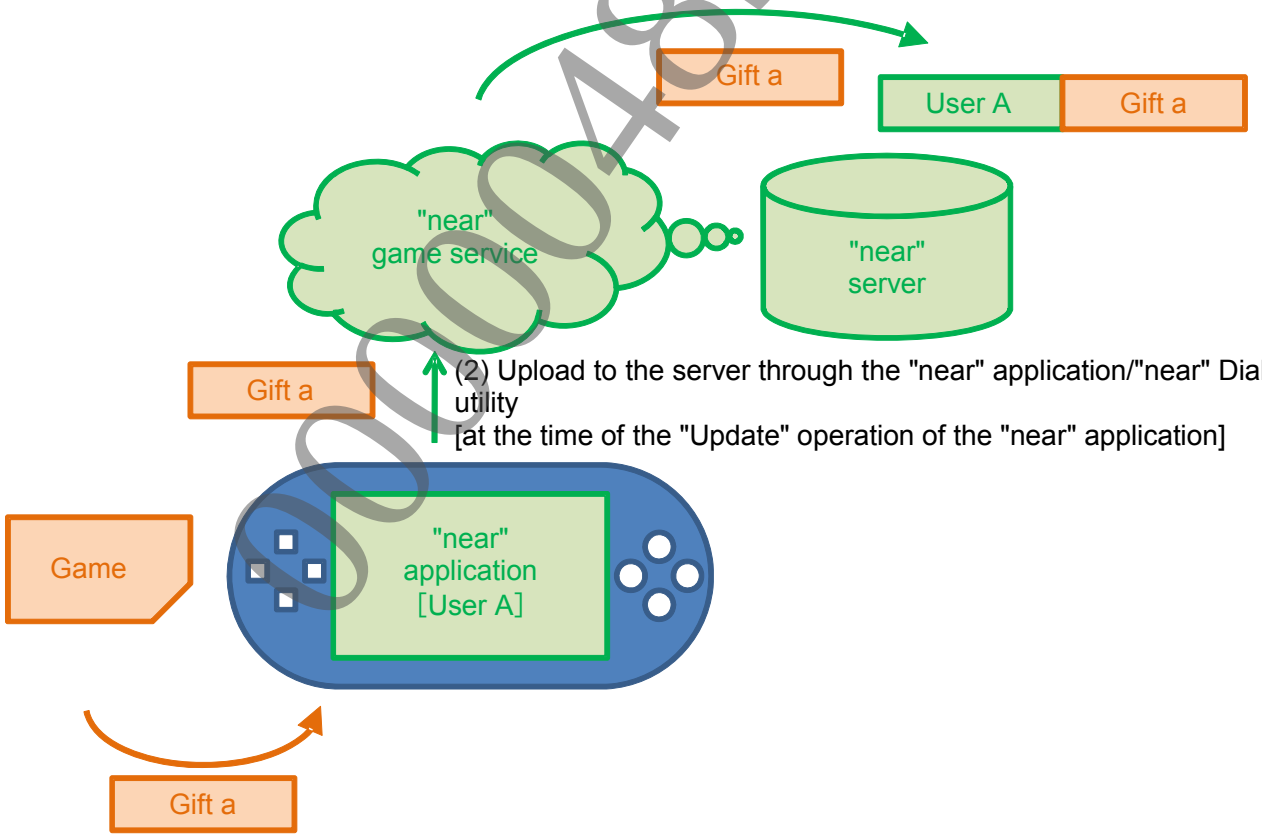
### (1) The Upload of Gift Data (Operation by User A on the Retail Unit)

Operation by User A on the retail unit to upload gift data to the "near" server is explained with reference to Figure 3 below.

(1) Gift data is registered to the retail unit using the "near" utility/"near" Dialog utility.
At this point in time, the "near" application/"near" Dialog utility becomes capable of uploading to the "near" server.

(2) On the "near" application/"near" Dialog utility, gift data is uploaded to the "near" server
Gift data is uploaded to the "near" server by the "Update" operation on the "near" application/"near" Dialog utility. Gifts uploaded with the "near" application/"near" Dialog utility will be selected by game title. For the selecting conditions, refer to the "near System Overview" document.

**Figure 3 User A Uploading Gift Data from the Retail Unit**




(1) Gift data is registered using the "near" utility/"near" Dialog utility .

### (2) The Download of Gift Data (Operations by User B on the Retail Unit)

Operation by User B on the retail unit to download gift data from the "near" server is explained with reference to Figure 4 below.

(1) The "near" application/"near" Dialog utility downloads the Condition part of a gift from the "near" server.
The Condition part of gift data distributed by a nearby user and friend is downloaded during the "Update" operation of the "near" application/"near" Dialog utility.

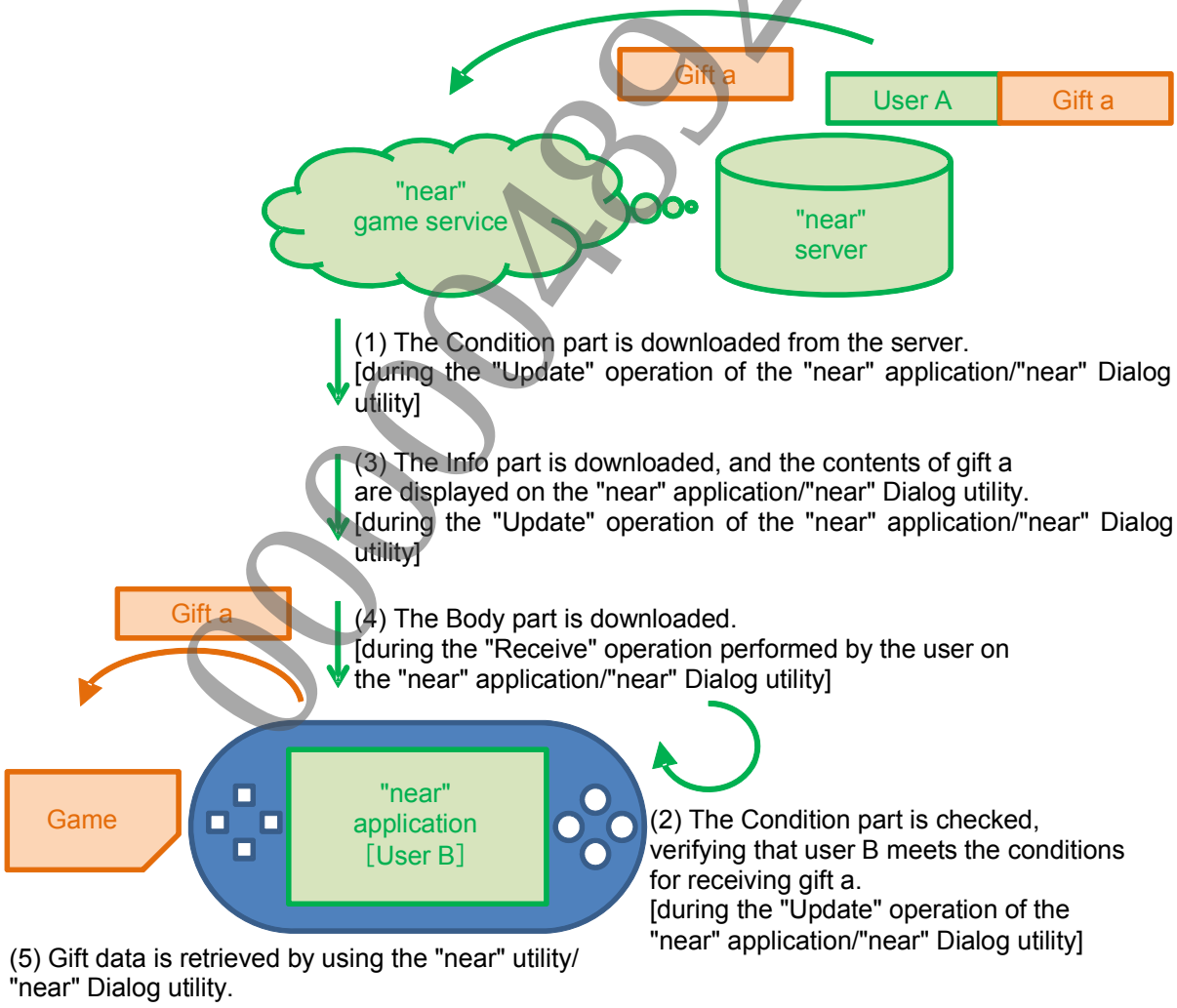
(2) The "near" application/"near" Dialog utility evaluates the Condition part of gift a
The Condition part is evaluated after step (1) in order to verify whether User B meets the receive conditions.

(3) The "near" application/"near" Dialog utility downloads the Info part of gift a.
The Info part is subsequently downloaded, provided that receive conditions are met in step (2).

(4) The "near" application/"near" Dialog utility downloads the Body part of gift a
The Body part of the gift is downloaded through User B's performance of the "Receive" operation in the "near" application/"near" Dialog utility.

(5) The game reads the Body part of the gift via the "near" utility/"near" Dialog utility and uses it in the game
The Body part of the gift is obtained from within the game via the "near" utility/"near" Dialog utility and used in the game.

**Figure 4 User B Downloading Gift Data from User A to the Retail Unit**



## Operation on the DevKit/TestKit

Operation of the "near" application and game programs using the "near" utility/"near" Dialog utility on DevKit depends on whether the **Release Check Mode** of the DevKit is set to **Release Mode** or **Development Mode**. For details on the **Release Check Mode** and the setting procedure, refer to the "Development Kit Neighborhood Settings Guide" document. Operation on TestKit is the same as on DevKit set to **Release Mode**.

When starting up the "near" application/"near" Dialog utility on DevKit/TestKit, pay attention to the following points. A memory card is required in order to start up the "near" application/"near" Dialog utility on DevKit in **Release Mode**/TestKit. On DevKit in **Development Mode**, connection to host0: is required.

Also, on DevKit/TestKit, when `SceNpCommunicationId` is input in the "near" utility/"near" Dialog utility to perform initialization, `SceNpCommunicationId` can be displayed within the game as a system message for confirmation. To do so, set **Settings -> ★Debug Settings -> PSN℠ -> NP Debug** to **ON**.

### Operation on DevKit (Development Mode)

The development host computer functions as the "near" server, which holds the gift data and transfers data among users when the **Release Check Mode** on the DevKit is set to **Development Mode**. Gift data is sent/received by writing/reading the data to/from the directory of the development host computer. Both Figure 5 and Figure 6 show this operation.

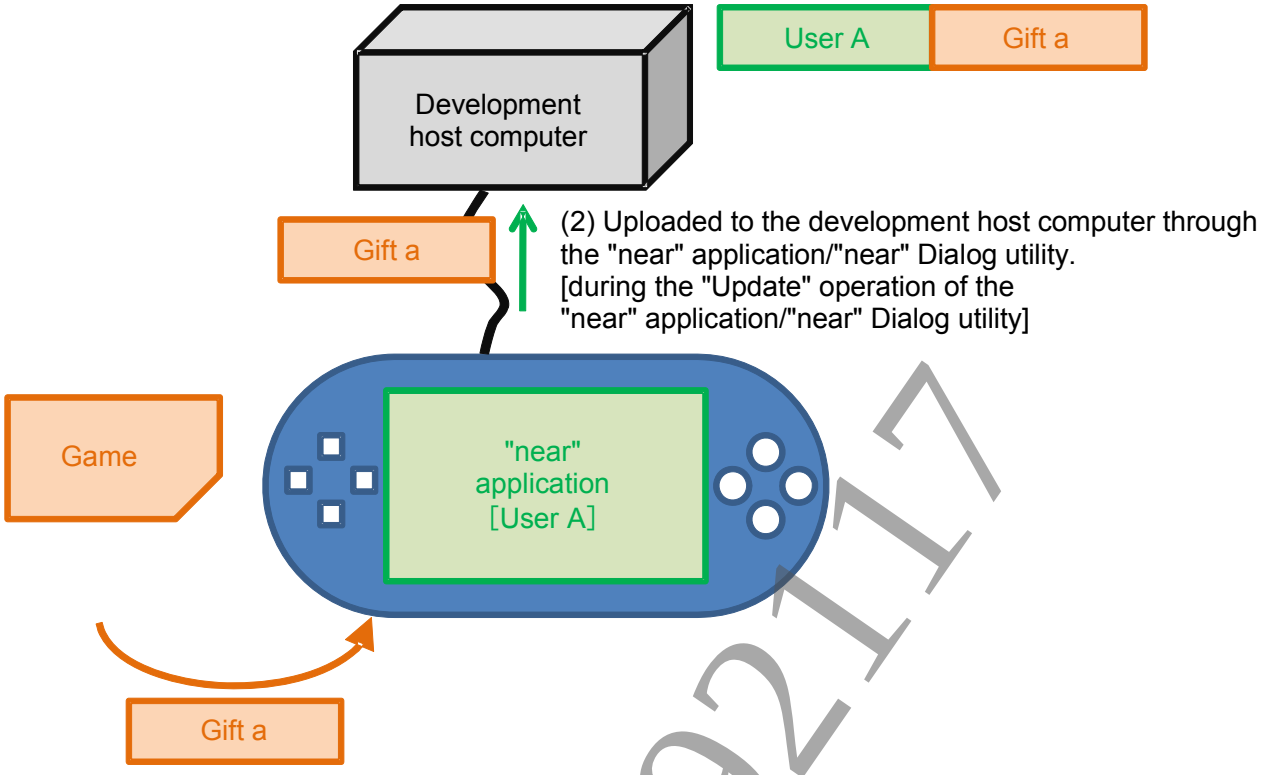
#### (1) The Upload of Gift Data (Operation by User A on DevKit (Development Mode))

Operation on DevKit (**Development Mode**) to be carried out by User A in order to upload gift data to the development host computer (functioning as the "near server) is explained with reference to Figure 5 below.

(1) Gift data is registered on the DevKit (**Development Mode**) by using the "near" utility/"near" Dialog utility
The "near" application/"near" Dialog utility becomes capable of uploading to the development host computer.

(2) Gift data is uploaded to the development host computer on the "near" application/"near" Dialog utility
Gift data is uploaded to the development host computer through the "Update" operation performed from the "near" application/"near" Dialog utility. Gifts uploaded from the "near" application/"near" Dialog utility will be selected by game title. For the selecting conditions, refer to the "near System Overview" document.

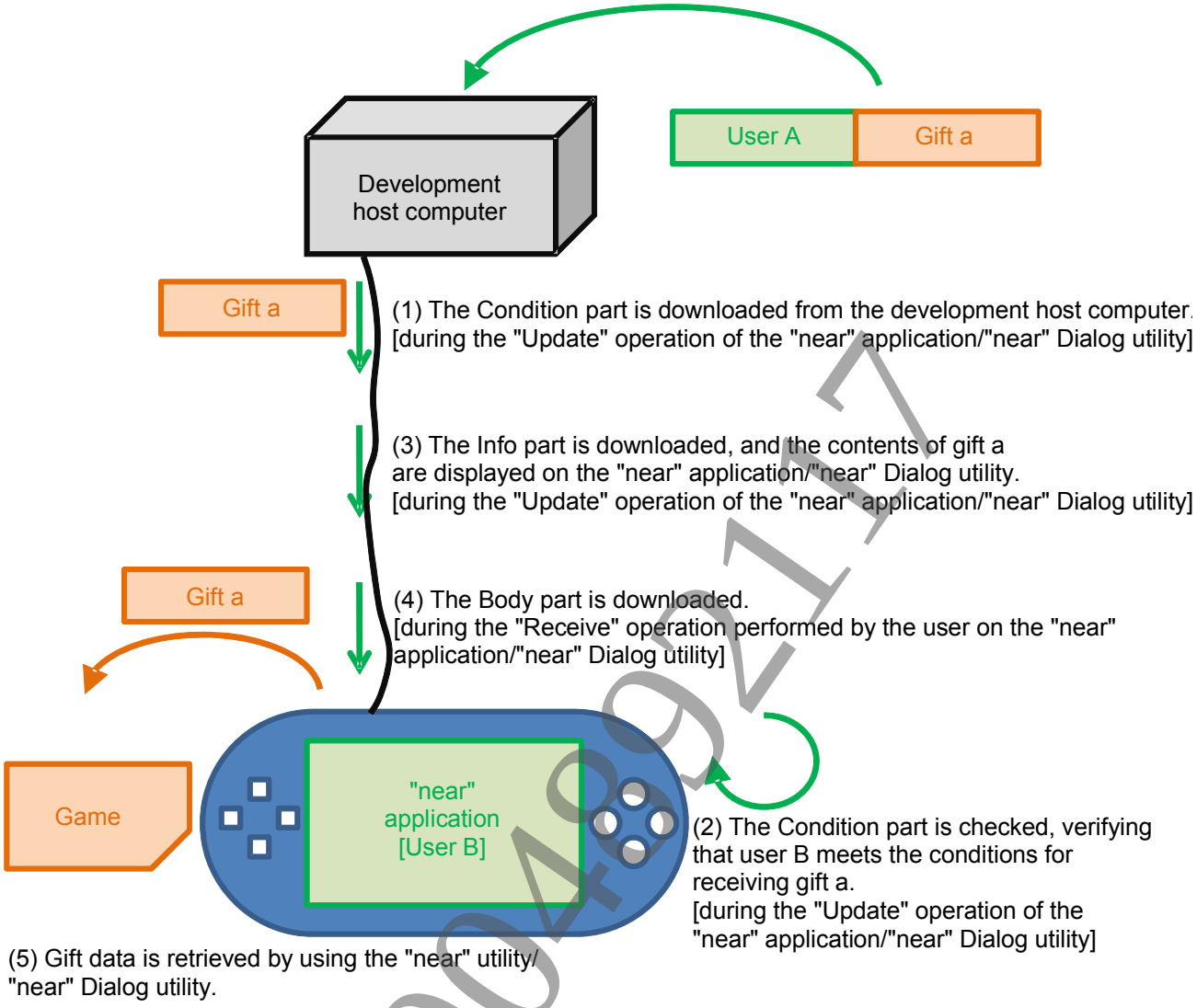
**Figure 5 User A Uploading Gift Data from the DevKit (Development Mode)**




### (2) The Download of Gift Data (Operation by User B on DevKit (Development Mode))

Operation on DevKit (**Development Mode**) to be carried out by User B in order to download gift data to the development host computer (functioning as the "near server) is explained with reference to Figure 6 below.

**Figure 6 User B Downloading Gift Data from User A on DevKit (Development Mode)**




(1) The "near" application/"near" Dialog utility downloads the Condition part of gift data from the development host computer
The Condition part of gift data uploaded by development host computer with another account is downloaded during the "Update" operation of the "near" application/"near" Dialog utility.

(2) The "near" application/"near" Dialog utility evaluates the Condition part of gift data
The Condition part is evaluated after step (1) in order to verify whether User B meets the receive conditions. Some part of the Condition part is not evaluated on DevKit (**Development Mode**). For details, refer to the "near Utility Overview" and "near Dialog Utility Overview" documents.

(3) The "near" application/"near" Dialog utility downloads the Info part of gift data from the development host computer
The Info part is subsequently downloaded from the development host computer, provided that receive conditions are met in step (2).

(4) The "near" application/"near" Dialog utility downloads the Body part of gift a from the development host computer
The Body part of gift data is downloaded from the development host computer by the User B's performing "Receive" operation in the "near" application/"near" Dialog utility.

(5) The game title reads the Body part of gift data via the "near" utility/"near" Dialog utility, and uses it in the game
The Body part is obtained from within the game title via the "near" utility/"near" Dialog utility, and used in the game.

### Operation on DevKit (Release Mode)/TestKit

The memory card functions as the "near" server, which holds the gift data and transfers the data among users when the **Release Check Mode** on the DevKit is set to **Release Mode** or on the TestKit. Gift data is sent/received by writing/reading the data to/from the memory card. Both Figure 7 and Figure 8 show this operation. Since the development host computer is not required, use it for the purpose of QA, etc.

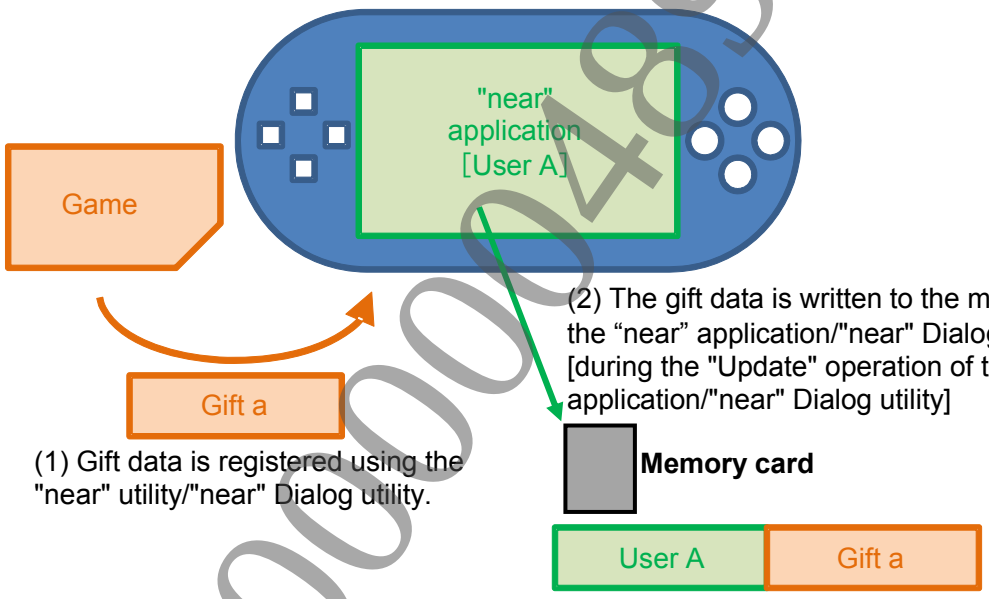
### (1) The Upload of Gift Data (Operations by User A on DevKit (Release Mode)/TestKit)

Operation on DevKit (**Release Mode**)/TestKit to be carried out by User A in order to upload gift data to the memory card (functioning as the "near server) is explained with reference to Figure 7 below.

(1) Gift data is registered to DevKit (**Release Mode**)/TestKit by the "near" utility/"near" Dialog utility
The "near" application/"near" Dialog utility becomes able to write data to the memory card.

(2) The gift data is written to the memory card through the "near" application/"near" Dialog utility
The gift data is written to the memory card during the "Update" operation of the "near" application/"near" Dialog utility. The gifts written to the memory card through the "near" application are selected by a game title. For the selecting conditions, refer to the "near System Overview" document.

**Figure 7 User A Uploading Gift Data from the DevKit (Release Mode)/TestKit**




### (2) The Download of Gift Data (Operations by User B on DevKit (Release Mode)/TestKit)

Operation on DevKit (**Release Mode**)/TestKit to be carried out by User B in order to download gift data from the memory card (functioning as the "near server) is explained with reference to Figure 8 below.

(1) The "near" application/"near" Dialog utility reads the Condition part of gift data from the memory card
The Condition part of a gift written to the memory card by another account is read during the "Update" operation of the "near" application/"near" Dialog utility.

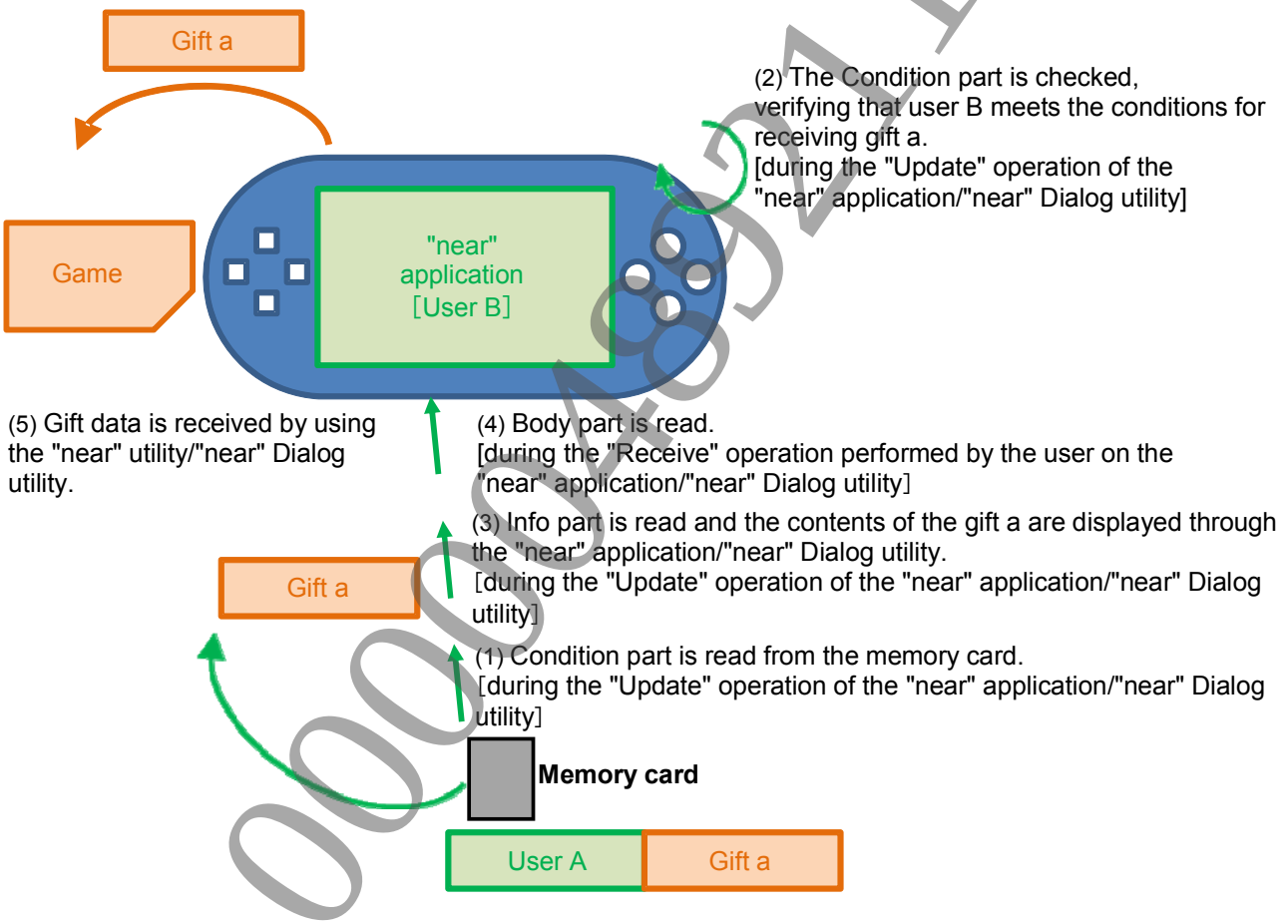
(2) The "near" application/"near" Dialog utility evaluates the Condition part of gift data.
The Condition part is evaluated after step (1) in order to verify whether User B meets the receive conditions. Some of the Condition part is not evaluated on DevKit (**Release Mode**)/TestKit. For details, refer to the "near Utility Overview" and "near Dialog Utility Overview" documents.

(3) The "near" application/"near" Dialog utility reads the Info part of gift data from the memory card
The Info part is subsequently read from the memory card, provided that receive conditions are met in step (2).

(4) The "near" application/"near" Dialog utility reads the Body part of gift data from the memory card
The Body part of gift data is read from the memory card by User B's performing "Receive" operation in the "near" application/"near" Dialog utility.

(5) The game title reads the Body part of gift data via the "near" utility/"near" Dialog utility and uses it in the game
The Body part of gift data is obtained from within the game title via the "near" utility/"near" Dialog utility and used in the game.

**Figure 8 User B Downloading Gift Data from User A to DevKit (Release Mode)/TestKit**



# 4 Checking/Testing on a DevKit/TestKit

The "near" application/"near" Dialog utility on DevKit/TestKit cannot be connected to the "near" server. However, with regards to behavior when various data is exchanged between the game program and the "near" utility/"near" Dialog utility, it is possible to achieve the same behavior as on retail units by setting the DevKit's **Release Check Mode** to **Release Mode** or using TestKit. Also, system messages for debugs that are displayed onscreen enable checking of whether intended parameters are sent to the "near" application via the "near" utility/"near" Dialog utility.

This chapter describes how to set up DevKit/TestKit for achieving an operation environment equivalent to that of retail units in order to improve the quality of game programs.

## How to Check Game Program Operation in an Operation Environment Equivalent to that of Retail Units Using DevKit

When the DevKit's **Release Check Mode** is set to **Release Mode**, boot parameters will be changed and the program will be executed in an environment equivalent to that of retail units. For this reason, set DevKit's **Release Check Mode** to **Release Mode** in order to check the operation of game programs in an operation environment equivalent to that of retail units using DevKit. For details, refer to the "System Software Overview" document.

## Checking the "near" Utility/"near" Dialog Utility in General

### `SceNpCommunicationId` and *`version`*

When the following settings are made on the DevKit/TestKit, the values specified in the `SceNpCommunicationId` and *`version`* arguments of `sceNearInitialize()` used when initializing the "near" utility/"near" Dialog utility will be displayed onscreen. Check that the correct `SceNpCommunicationId` and *`version`* are being used.

(1) Start up the Settings application and select ★**Debug Settings**

(2) Select **PSN℠**

(3) Set **NP Debug** to **ON**

## Checking the Transfer of Gifts

### Checking the Sending/Receiving of Gifts on DevKit (Release Mode)/TestKit

(1) Create multiple Sony Entertainment Network accounts for a single DevKit (**Release Mode**)/TestKit
In order to check the sending/receiving of gifts, create at least two Sony Entertainment Network accounts (receiving and sending side). For details on creating Sony Entertainment Network accounts, refer to the "PSN℠ Overview" document.

(2) Switch the Sony Entertainment Network account to the sending-side account

(3) Register the gifts to be sent within the game program

(4) Start up the "near" application, tap the **"near" Update** button and run with the **Share Online ID** setting. Alternatively, check both the **Share Online ID** and **Do not ask my permission again** of **Sharing Settings** in **Settings** of the "near" application and use the "near" Dialog utility to carry out "Update" operation.
Gift data matching the sending-side account will be written to the memory card.

(5) Switch the Sony Entertainment Network account to the receiving-side account
Perform subsequent operations while signed in to PSN℠ with the receiving-side account.

(6) Start up the "near" application, tap the **"near" Update** button and run with the **Share Online ID** setting. Alternatively, check both the **Share Online ID** and **Do not ask my permission again** of **Sharing Settings** in **Settings** of the "near" application and use the "near" Dialog utility within the game to carry out "Update" operation.
Gifts data written out to the memory card in the step (4) will be referenced. The conditions for the receiving the gifts will be also referenced, and gifts will be discovered by the receiving-side account in accordance with the receive conditions.

(7) From the "Discoveries" screen of the "near" application, open the "Game Goods Details" screen and tap the **Download** button to start receive processing. In addition, use the "near" Dialog utility from within the game program to receive items.

(8) Start the game from the "near" application and check whether the gifts received from the "near" application can be used. In addition, use the "near" Dialog utility from within the game to check whether the gifts received from the "near" application can be used.

### Checking the Registered Gifts to Be Sent

After performing (3) of "Checking the Sending/Receiving of Gifts on DevKit (Release Mode)/TestKit", check the following with the sending-side account:

- By tapping the user's own avatar on the "Discoveries" screen in the "near" application, an icon will be displayed next to "Play History", indicating that there are registered gifts to be sent. Tap the icon to display the contents of gifts, and then check whether the item name character strings, item details character strings and item images are displayed correctly.

### Checking the Contents of Sent/Received Gifts

After performing "Checking the Sending/Receiving of Gifts on DevKit (Release Mode)/TestKit", check the following with the receiving-side account:

- Check that the item name character strings, item detail character strings and item images on the "Discoveries" screen and "Game Goods Details" screen in the "near" application are displayed correctly.
- Use the received data in the game and check that it can be used correctly.

### Checking the Receive Conditions of Sent/Received Gifts

On DevKit/TestKit, the following receive conditions function in the same manner as on retail units:

- Whether a gift can only be received by users who own the title or can be received by users who do not own the title
(In the "near" game service, even if a title has been installed, that title is not deemed to be owned until it is started up and initialization processing using the "near" utility/"near" Dialog utility in the title is performed.)
- Whether the same gift can or cannot be received from multiple users
- Validity period after the gift is discovered
- Relationship between the gift sender and the receiving users (attributes)
- Probability of gift discovery

Since the following receive condition does not function on the DevKit/TestKit in the same manner as on retail units, it is necessary to check whether it is registered correctly using a system message displayed on the game screen. (Figure 9)

- Distance (meters) between the gift sender and receiving users

### Checking the Receive Conditions on DevKit/TestKit

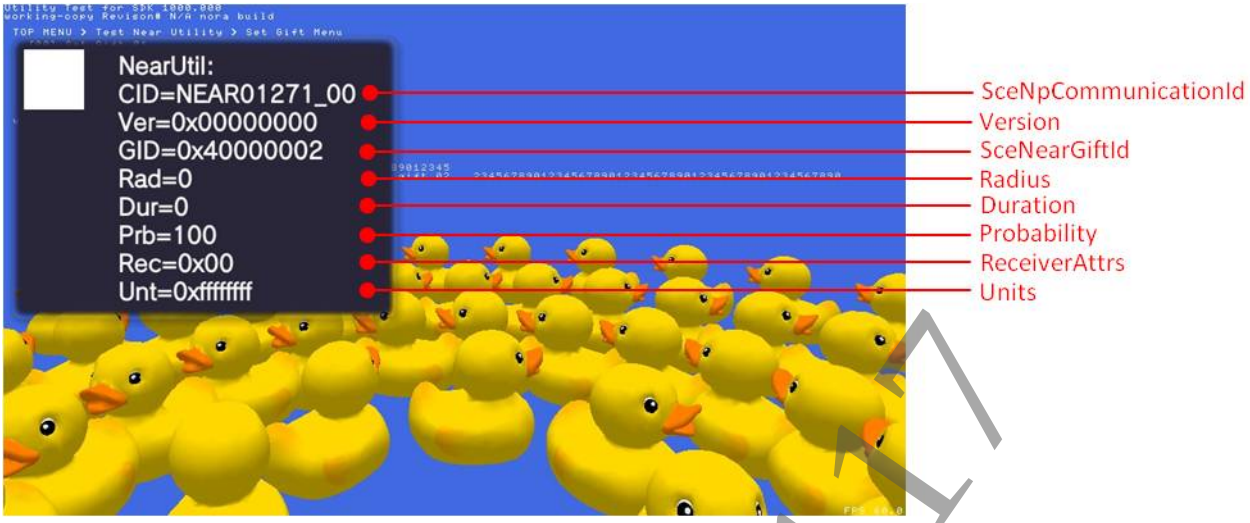
To display the receive conditions as a system message, make settings as follows:

(1) Start up the Settings application and select ★**Debug Settings**

(2) Select **PSN℠**

(3) Set **NP Debug** to **ON**

**Figure 9 Displaying Conditions to Receive a Gift**




### Placing Gift Data to the Development Host Computer

When using the above-described method, a procedure for switching the gift sending user and the gift receiving user on one DevKit/TestKit is required. A method for conveniently placing already transmitted gifts distributed from multiple users on the development host computer is provided. Using this method, up to 100 gift data items can be placed; so use this method as needed.

(1) Create Sony Entertainment Network accounts to be used as gift senders for the test

(2) Set the **Release Check Mode** of DevKit to **Development Mode**

(3) Reboot the system software

(4) In host0:gift_import/, create a file in a format like that shown in Figure 10 (file name: gift_import.txt). Use the account created in step (1) as the gift-sending user

(5) Place thumbnail images and gift data files of gifts specified in gift_import.txt in host0:

(6) Input and execute the following command in the command line of the development host computer:

```
psp2ctrl cp ./gift_import/* ux0:data/near/gift_import/
```

In this way, the gift_import.txt file, the thumbnail image files, and the gift data files will be written onto the memory card.
Also, refer to the "System Software Overview" document on how to place data on the memory card

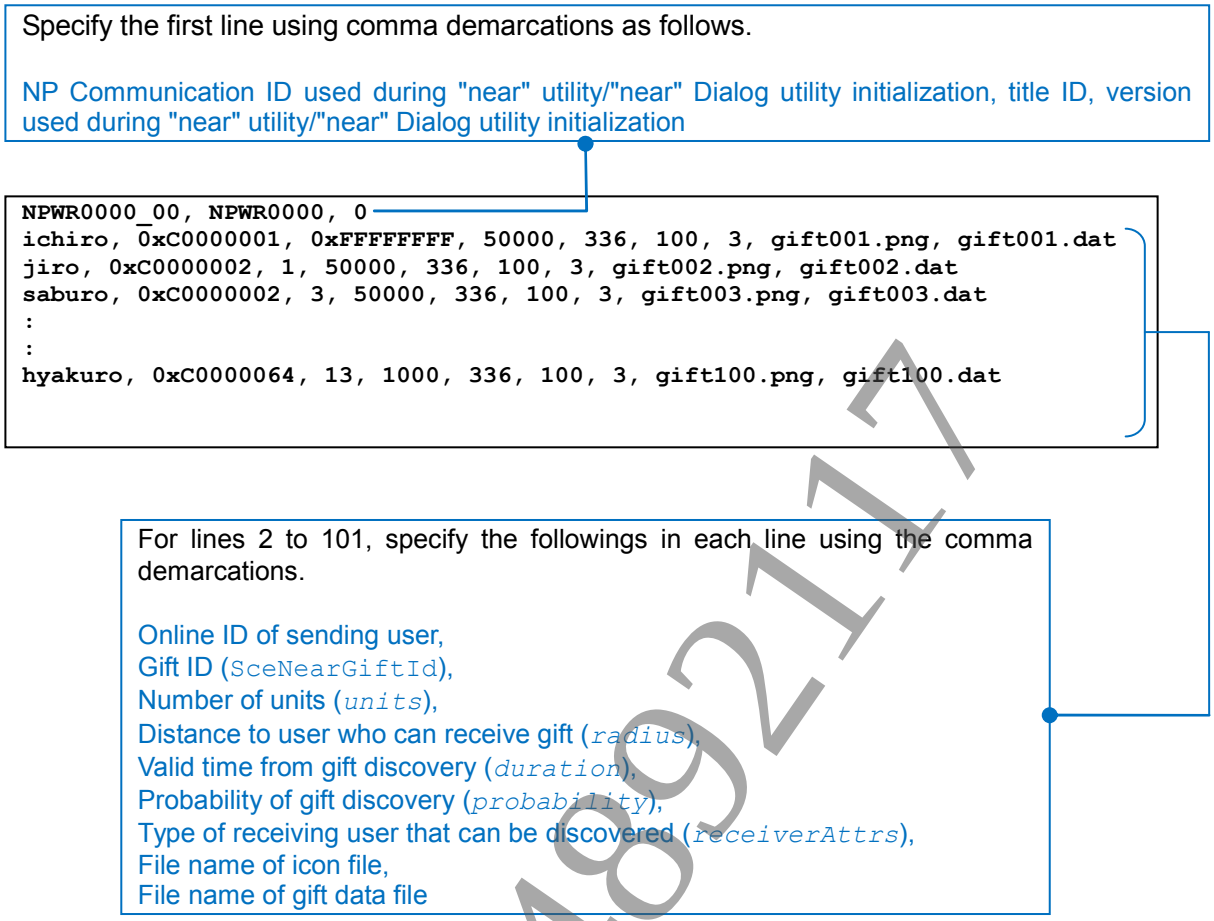
(7) Set the DevKit's **Release Check Mode** to **Release Mode** or insert the memory card used in (6) to the TestKit

(8) Reboot the system software

(9) Switch to the account of the gift-receiving user and start up the "near" application. At this time, ux0:data/near/gift_import/gift_import.txt is read by the "near" application, and the thumbnail image files and the gift data files of the gift(s) referenced from gift_import.txt are read and are placed on the development host computer.
By reflecting it on the "near" application, ux0:data/near/gift_import/gift_import.txt file is deleted automatically.

(10) Tap the **"near" Update** button of the "near" application and check whether the gifts can be correctly received. Alternatively, use the "near" Dialog utility from within the game program to "Update" operation. Note that only up to 10 gifts can be received at one location. To receive more than 10 gifts, the **"near" Update** button must be tapped multiple times.

**Figure 10 gift_import.txt Writing Example**



## How to Delete Gifts in the "near" Application

It is possible to delete gift information that is saved in the "near" memory area of a DevKit/TestKit and restart testing from the gift discovery stage.

### How to Delete "Discovered" and "Received" State Gifts

A red **Delete** button will be displayed in the lower right of the "Game Goods Details" screen on DevKit/TestKit. Press this button and the displayed gift will be deleted, and the same gift can be discovered again.

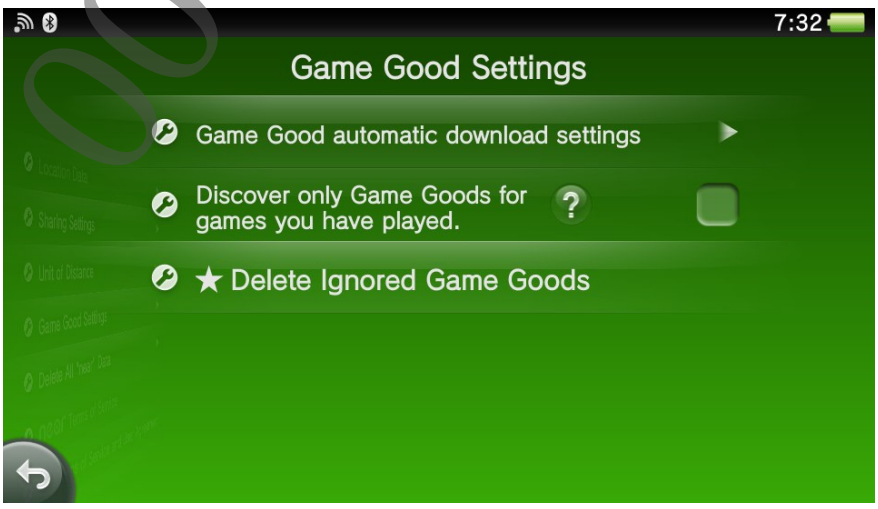
**Figure 11 Game Goods Details Screen (DevKit/TestKit)**


### How to Delete "Ignored" State Gifts

"Ignored" state gifts are still saved on the memory area of the "near" application, but they will not be displayed on the "Discoveries" screen of the "near" application. Therefore, they cannot be individually deleted from the "Game Goods Details" screen.

By selecting **Settings -> Game Goods Settings** on DevKit/TestKit, the **★Delete Ignored Game Goods** menu item will be displayed for deleting "Ignored" state gifts only. By selecting this menu item, all of the "Ignored" state gift information that is not displayed in the "near" application will be deleted from the "near" memory area, and the same gifts can be discovered again.

**Figure 12 Game Goods Settings Screen (Menu Item for Deleting "Ignored" State Gift Information)**



## Checking Nearby User Obtainment

### Checking Nearby User Obtainment on DevKit/TestKit

(1) Create multiple Sony Entertainment Network accounts for a single DevKit/TestKit

(2) Switch the Sony Entertainment Network account to an account other than the account for which you wish to obtain nearby users
Perform subsequent operations while signed in to PSN℠ with the account other than the account for which you wish to obtain nearby users.

(3) Register gifts to be sent within the game program

(4) Start up the "near" application, tap the **"near" Update** button and run with the **Share Online ID** setting. Alternatively, check both the **Share Online ID** and **Do not ask my permission again** of **Sharing Settings** in **Settings** of the "near" application and use the "near" Dialog utility within the game to carry out "Update" operation.
Gift data matching the sending-side account will be written onto the memory card.

(5) Switch the Sony Entertainment Network account to the account for which you wish to obtain nearby users
Perform subsequent operations while signed in to PSN℠ with the account for which you wish to obtain nearby users.

(6) Start up the "near" application, tap the **"near" Update** button and run with the **Share Online ID** setting. Alternatively, check both the **Share Online ID** and **Do not ask my permission again** of **Sharing Settings** in **Settings** of the "near" application and use the "near" Dialog utility within the game to carry out "Update" operation.
Users that have written out gifts to be sent onto the memory card with other accounts will be referenced. Receive conditions will not be referenced, and nearby users will be discovered.

(7) Start up the game and check that nearby users can be obtained correctly

## Obtaining the Usage Status of the "near" Application

### Checking the Usage Status of the "near" Application on DevKit/TestKit

(1) Set the DevKit's **Release Check Mode** to **Development Mode**

(2) Reboot the system software

(3) In host0:, create a file in a format like that shown in Figure 13 (file name: near_status.txt)

(4) Input and execute the following command in the command line of the development host computer:
```
psp2ctrl cp ./near_status.txt ux0:data/near_status.txt
```
In this way, the near_status.txt file will be written onto the memory card.
Also, refer to the "System Software Overview" document on how to place data on the memory card.

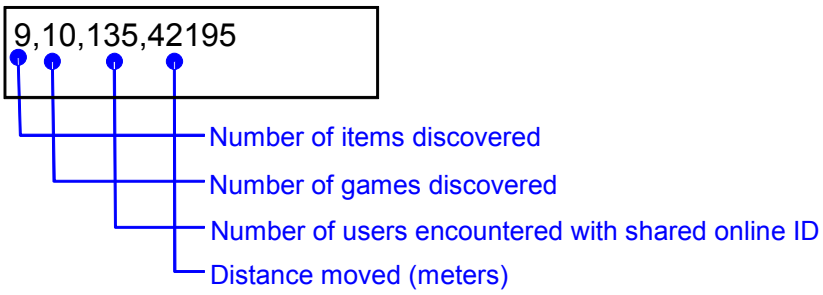
(5) Set the DevKit's **Release Check Mode** to **Release Mode** or insert the memory card used in (4) to the TestKit

(6) Reboot the system software

(7) Start up the "near" application. At this time, ux0:data/near_status.txt will be read by the "near" application
By reflecting it on the "near" application, the ux0:data/near_status.txt file is deleted automatically.

(8) Start up the game, then check whether the usage status of the "near" application is obtained correctly, and whether it is successfully reflected in the game

**Figure 13 near_status.txt Writing Example**




## Obtaining a List of Nearby Users

Below is an explanation of a procedure for recording nearby users on the "near" application in order to check that the game program is capable of obtaining lists of nearby users saved on the "near" application.

(1) Set the DevKit's **Release Check Mode** to **Development Mode**

(2) Reboot the system software

(3) In host0:, create a text file named near_neighbors.txt shown in Figure 14

(4) Input and execute the following command in the command line of the development host computer:
`psp2ctrl cp ./near_neighbors.txt ux0:data/near_neighbors.txt`
In this way, the near_neighbors.txt file will be written onto the memory card.

(5) Set the DevKit's **Release Check Mode** to **Release Mode** or insert the memory card used in (4) to the TestKit

(6) Reboot the system software

(7) Sign in to PSN℠ from **Settings -> PSN℠**

(8) Start up the "near" application
Start up the "near" application. At this time, ux0:data/near_neighbors.txt will be read by the "near" application. Then, the nearby users are recorded into the "near" application as the new nearby users discovered with the "near" application. Upon termination of the processing, ux0:data/near_neighbors.txt is deleted.

(9) Start up the game, then check whether the nearby users stored in the "near" application are obtained correctly, and whether they are successfully reflected in the game.

### Details on the Processing during the Startup of the "near" Application

Below are the details on the processing of the above-described procedure (8) by which the "near" application reads near_neighbors.txt and precautions on the processing.

- Upon startup, the "near" application reads near_neighbors.txt that is stored on the memory card
- The online IDs (`SceNpOnlineId`) written in near_neighbors.txt are converted to the valid NP IDs by the NP Lookup library. Therefore, connection with PSN℠ must be established
- In the case that a network error occurs, such as, when the user cannot sign in to PSN℠, conversion to NP IDs cannot be performed, and thus the registration of nearby users after the error occurrence will be terminated without being processed
- When an online ID cannot be converted to a valid NP ID (`SceNpId`) through the conversion processing of the NP Lookup library, the online ID will not be registered as a nearby user
- The online IDs that failed to be converted to NP IDs will be written to the ux0:data/near_neighbors_failed.txt file

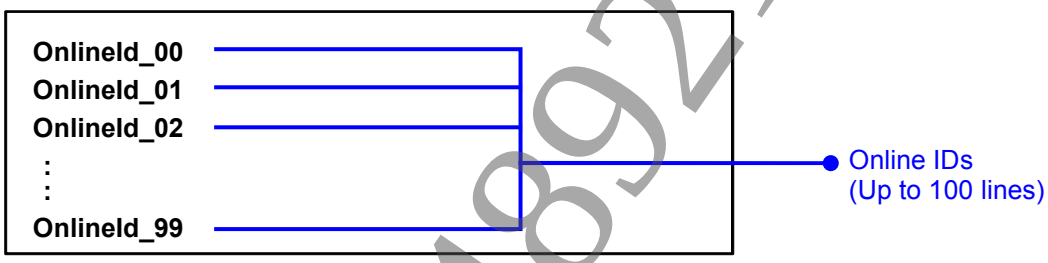
- There is a possibility that the number of nearby users that can be obtained with `sceNearGetMyStatus()` is larger than that of `sceNearGetNeighbors()`, `sceNearGetRecentNeighbors()` and `sceNearGetNewNeighbors()`, because the online IDs written in the text file are directly converted to the NP IDs. This phenomenon never occurs when executing an actual game.

### near_neighbors.txt File Format

Below is a description of the format that is used when writing online IDs to near_neighbors.txt.

- One line consists of one online ID
- Either CR+LF or LF can be used as the line break code
- Up to 100 online IDs (i.e. 100 lines) can be written
  Since only up to the 100th online ID can be read, online IDs below the 100th line will be ignored
- The maximum file size is 2 KB
  Data of up to 2 KB will be read
- Signed-up online IDs are used
  The online IDs that have not been signed up yet will not be registered to the "near" application as nearby users

**Figure 14 near_neighbors.txt Writing Example**




## Notes

- With a DevKit/TestKit, the "near" server is not used for gift registration/discovery. For this reason, you will not be able to verify all behavior occurring when the "near" server is actually used.
  There are the following limitations:
  - Specifying the location for gift distribution
    With a DevKit/TestKit, gift distribution location and discovery location are the same
  - Specifying the radius where gifts can be received
    With a DevKit/TestKit, the radius within which gifts can be discovered is never left
- DevKit/TestKit does not use the "near" server for gift registration/discovery; instead, it allows you to export the files to the development host computer or a memory card and to verify simulated operation.
  Compatibility is not guaranteed when the SDK version is different for files exported to the development host computer/memory card.

# 5 Implementation

This chapter provides an example of how to implement the "near" game service in a game. Depending on the game's scenario and exposition, the example of implementation provided in this chapter may not be applicable as it is. Refer to this when using the "near" game service in a game.

## Distribution of Items or Raw Materials

This example of implementation entails the distribution of items and raw materials that offer a high added value when used in the game.

### Data Stored in the Body Part of Gift Data

ID or flag that expresses an item or material, or the data itself that expresses an item or material

### Implementation Method

(1) Sending Side
Store the ID, flag or data itself that expresses an item or material assigned in a game as a gift data in the Body part of the gift data using the "near" utility/"near" Dialog utility in the game program. Allocate `SceNearGiftId` by the granularity of the item or material in the game.

(2) Receiving Side
Read the ID, flag or data itself that expresses an item or material stored in the gift data from the Body part of the gift data using the "near" utility/"near" Dialog utility in the game program, and is reflected within the game.

### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated when storing an item or material in a game program.

- [Flag that enables even users who do not own the game to be discovered (=0x80000000)]
  To make the game itself serve an advertising role: Enable (HIGH)
  To prevent in-game information from being disclosed: Disable (LOW)
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Items, such as medicinal herbs, of which several can be owned without causing a problem: Enable (HIGH).
  Items, such as weapons and protective gears, of which ownership of one is sufficient: Disable (LOW).

**`SceNearGiftCondition.`*`radius`***

Specify the distance according to purpose, within the game program.

- If the game is given the role of advertising itself: 0 m (use the critical distance determined on the server).
- To increase the desired value of that item or material: Set an appropriate finite value.

**`SceNearGiftCondition.`*`duration`***

Determine the expiration date suitable for an item or material in the game program. (For example, in the case of an item or material with the attribute of "freshness," set a shorter expiration date.)

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000)

**SceNearGiftCondition.*probability***

Determine the discovery probability according to the item or material in the game program. (The discovery probability can be adjusted during operation.)

***units*(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())**

Determine whether or not to set a unit limit according to the item or material in the game program. (This can be used taking into consideration the game element.)

## Exchange of Items

This implements the exchange of items A and B that one owns in the game, and items C and D that another person owns in the same game.

### Data Stored in the Body Part of Gift Data

- Data of A and B, or the flags to unlock them
- As the condition for using the data of A and B, the data of C and D must be sent back
- "Request ID" assigned in a game on the sending side

### Implementation Method

(1) Sending Side
In the game program, store the following three data items in the Body part of the gift data using the "near" utility/"near" Dialog utility.

- Data of A and B, or the flags to unlock them
- Usage condition that only the person that passes the data of C and D can use A and B
- "request ID" assigned for one exchange request, while being incremented at each request in the game program

Allocate one `SceNearGiftId` for one "request ID" in the game program.

Save the "request IDs" of the unrealized requests to the save data, etc., in the game program.

(2) Receiving Side
Using the "near" utility/"near" Dialog utility in the game program, read the data of A and B (or the flags to unlock them), usage condition that the data of C and D are to be submitted to the sending side and "request ID" stored in the Body part of the gift data.
The receiving side compares the combination of the sender's online ID and "request ID" with the combination of the already processed online ID and "request ID" saved in the game program of the receiving side. If the gift data is that of the sender with the same online ID, and the processed "request ID" is the same or greater than the "request ID" received this time, the "request" received this time is ignored.
If the received "request ID" is greater than the processed "request ID", the game program will proceed to the processing for using the "request" received this time.
If the data of C and D in the game program of the receiving side can be sent, game boot messages (custom data attached) is sent to the sending side user, with the data of C and D (or the flags to unlock them) and the "request ID" sent from the sending side as custom data.
Upon the completion of the send processing, enable the data of A and B to be used in the game program.
The online ID and "request ID" of the sender of the processed request are saved using save data, etc., in the game program.

(3) Sending Side
The sending side opens the game boot messages (custom data attached) sent from the receiving side user in the game program, and upon finding out that the request of the "request ID" stored in the custom data has been realized, it makes the use of the data of C and D possible. The realized "request ID" is deleted from the list of unrealized "request IDs" saved in the save data, etc.

### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated per "exchange request" in the game program.

- [Flag that enables even users who do not own the title to be discovered (=0x80000000)]
  To make the game itself serve an advertising role: Enable (HIGH)
  To prevent in-game information from being disclosed: Disable (LOW)
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Enable (HIGH).

**`SceNearGiftCondition.`*`radius`***

0 m (use the critical distance determined on the server)

**`SceNearGiftCondition.`*`duration`***

Determine the expiration date suitable for the item or material in the game program. (For example, in the case of an item or material with the attribute of "freshness," set a shorter expiration date.)

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000).

**`SceNearGiftCondition.`*`probability`***

Set to 100%.

***`units`*`(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())`**

0xFFFFFFFF (In this assumed usage example, no restriction can be placed on the number of units.)

## Collection of Numerical Values

"Things that can be received only from other persons" and "things that the more you have, the happier you are," - such as, power that can be used by the character operated by the game user - are distributed as gifts.

This implementation example assumes that, in return for one's own power being used by other users, the power of other users can be gained as well.

### Data Stored in the Body Part of Gift Data

- Parameter that expresses the power in the game of the sender's side
- "Power ID" assigned in the game of the sender's side

**Implementation Method**

(1) Sending Side

In order to gain power from other users in the game program, the "power collection" action is invoked in the game of the sending side.

Store the following two data items in the Body part of the gift data using the "near" utility/"near" Dialog utility in the game program.

- Power parameter of the character the sender is operating
- "Power ID" that is assigned while being incremented for each power parameter instance in the game program

Allocate one `SceNearGiftId` for one "power ID" in the game program.

At the same time gift registration completes, one of the following is carried out within the game program.

- The game program uses the "near" utility to start the "near" application and prompt a "near" update
- The game program uses the Message Dialog library to prompt a "near" update

Carry out one of the following so that the user receives power from other users while sending his/her own power parameter.

- The user taps the **"near" Update** button from within the "near" application
- After the game program receives agreement from the user using the Message Dialog Library to update "near", the game program uses the "near" Dialog utility to update "near"

(2) Receiving Side

Following reception of another user's gift in the "near" application, the power parameter and "power ID" of the sending side stored in the gift data are read using the "near" utility/"near" Dialog utility in the game program.

The receiving side compares the combination of the sender's online ID and "power ID" with the combination of the already processed online ID and "power ID" saved in the game program of the receiving side. If the gift data is that of the sender with the same online ID, and the processed "power ID" is the same or greater than the "power ID" received this time, the power received this time is ignored.

If the received "power ID" is greater than the processed "power ID", the game program will proceed to the processing for using the power received this time.

The received power parameters that can be used are reflected to the character of the receiving user.

The online ID and "power ID" of the sender of the processed power are saved using save data, etc., in the game program.

#### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated per "power collection" action in the game program.

- [Flag that enables even users who do not own the title to be discovered (=0x80000000)]
  Disable (LOW).
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Enable (HIGH).

**`SceNearGiftCondition.`*`radius`***

0 m (use the critical distance determined on the server)

**`SceNearGiftCondition.`*`duration`***

Set between 0 hour and 24 hours. (The shorter the duration is set, the more persons power must be received from simultaneously; this is used to set the difficulty level.)

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000)

**`SceNearGiftCondition.`*`probability`***

Set to 100%.

***`units`*`(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())`**

Set 0xFFFFFFFF (unlimited).

## Unlocking of Scenario

This example of implementation entails the transmission and reception of game progress status.

The sending side distributes his/her stage progress status in the game to those around.

For stages that the receiving side has not yet unlocked, the receiving side unlocks the stages beaten by the user who sent the gift so that they can be played.

#### Data Stored in the Body Part of Gift Data

- Status data that expresses the beating status of each stage in the game (set of flags)
- "Status ID" assigned through incrementing in the game

#### Implementation Method

(1) Sending Side
In the game program, store the following two data items in the Body part of the gift data, using the "near" utility/"near" Dialog utility.

- Status data that expresses the status of the stages beaten by the sending side user in the game
- "Status ID" that is assigned while being incremented for each stage progress status data in the game

Allocate one `SceNearGiftId` for one "status ID" in the game program.

(2) Receiving Side
The status data that expresses the status of the stages beaten by the sender and "status ID" of the sending side stored in the Body part of the gift data are read using the "near" utility/"near" Dialog utility in the game program.
The receiving side compares the combination of the sender's online ID and "status ID" with the combination of the already processed online ID and "status ID" saved in the game program of the receiving side.
If the gift data is that of the sender with the same online ID, and the processed "status ID" is the same or greater than the "status ID" received this time, the "status" received this time is ignored.
If the received "status ID" is greater than the processed "status ID", the game program will proceed to the processing for using the "status" received this time.
If a stage that is still not unlocked in the receiving side game program is beaten in the sending side game program, the corresponding stage is unlocked.
The online ID and "status ID" of the sender of the processed status are saved using save data, etc., in the game program.

### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated each time a stage beating status is assigned to the gift data.

- [Flag that enables even users who do not own the title to be discovered (=0x80000000)]
  To make the game itself serve an advertising role: Enable (HIGH)
  If the stage beating unit is too small, causing inconvenience to users: Disable (LOW)
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Enable (HIGH).

**`SceNearGiftCondition.`*`radius`***

0 m (use the critical distance determined on the server)

**`SceNearGiftCondition.`*`duration`***

Set 0 hour (longest validity period determined by the "near" system).

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000)

**`SceNearGiftCondition.`*`probability`***

Set to 100%.

***`units`*`(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())`**

Set 0xFFFFFFFF (unlimited).

## Distribution of Challenge Letters and Invitation Letters

This example of implementation entails inviting to play together other users who are online or connected on an ad hoc basis.

It is assumed that to play with the recruited users, the sending side and the receiving side must both be online, or else be connected on an ad hoc basis.

If distributing challenge letters to opponents in a race game, the ghost data is stored in the gift data and distributed.

In online RPG, events to be realized by organizing a party, or one's own fighting capability is stored in the gift data and distributed.

### Data Stored in the Body Part of Gift Data

- Profile data that indicates the game progress status of the users defined in the game
- "Profile ID" assigned through incrementing in the game

### Implementation Method

(1) Sending Side
In the game program, store the following two data items in the Body part of the gift data, using the "near" utility/"near" Dialog utility.

- Profile data that indicates the progress status in the game
- "Profile ID" that is assigned while being incremented in the game program each time the profile data is updated

Allocate one `SceNearGiftId` to one "profile ID" in the game program and register the gift data to send. The game program will update "near" with the "near" Dialog utility after it subsequently receives agreement from the user via the Message Dialog library to update "near". Alternatively, the game program will use the "near" utility to start the "near" application, and the user will tap the **"near" Update** button within the "near" application to update.

When "near" is updated, the user receives the profile data of other users while sending his/her own profile data.

(2) Receiving Side
Using the "near" utility/"near" Dialog utility in the game program, the profile data and "profile ID" stored in the gift data are read.
The receiving side compares the combination of the sender's online ID and "profile ID" with the combination of the already processed online ID and "profile ID" saved in the game program of the receiving side.
If the gift data is that of the sender with the same online ID, and the processed "profile ID" is the same or greater than the "profile ID" received this time, the "profile" received this time is ignored.
If the received "profile ID" is greater than the processed "profile ID", the game program will proceed to the processing for using the "profile" received this time.
Whether to play with the user who sent the gift is confirmed while presenting the received usable user profile data to the receiving side user. Once the receiving user decides to play, he/she sends a message indicating either "I accept the invitation" or "I accept the challenge," and the respective users are prompted to contact each other about the date and time for playing together.
The online ID and "profile ID" of the sender of the processed profile are saved using save data, etc., in the game program.

### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated each time a profile data is assigned to the gift data.

- [Flag that enables even users who do not own the title to be discovered (=0x80000000)]
  To make the game itself serve an advertising role: Enable (HIGH)
  If the profile data sending frequency is too high, causing inconvenience to users: Disable (LOW).
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Enable (HIGH).

**`SceNearGiftCondition.`*`radius`***

0m (use the critical distance determined on the server)

**`SceNearGiftCondition.`*`duration`***

Set between 0 hour and 24 hours. (The shorter the duration is set, the more accurately the latest in-game status is relayed.)

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000)

**`SceNearGiftCondition.`*`probability`***

Set to 100%.

***`units`*`(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())`**

Set 0xFFFFFFFF (unlimited).

## Distribution of the Alter Ego of a Pet or One's Own Character

This example of implementation entails sending an alter ego to another player's game to work on behalf on one's character.

When one's alter ego comes back from work, the payment earned becomes one's payment.

### Data Stored in the Body Part of Gift Data

- "Alter ego data" that indicates the capability of the alter ego of the character defined in the game
- "Alter ego ID" assigned through incrementing in the game

### Implementation Method

(1) Sending Side
In the game program, store the following two data items in the Body part of the gift data, using the "near" utility/"near" Dialog utility.

- "Alter ego data" of one's own character
- "Alter ego ID" that is assigned while being incremented each time the alter ego data is updated in the game program

Allocate one `SceNearGiftId` to one "alter ego ID" in the game program and register the gift data to send. The game program will update "near" with the "near" Dialog utility after it subsequently receives agreement from the user via the Message Dialog library to update "near". Alternatively, the game program will use the "near" utility to start the "near" application, and the user will tap the **"near" Update** button within the "near" application to update.

When "near" is updated, the user receives the alter ego data of other users while sending his/her own "alter ego data"

(2) Receiving Side
Using the "near" utility/"near" Dialog utility in the game program, the "alter ego data" and "alter ego ID" stored in the gift data are read.
The receiving side compares the combination of the sender's online ID and "alter ego ID" with the combination of the already processed online ID and "alter ego ID" saved in the game program of the receiving side.
If the gift data is that of the sender with the same online ID, and the processed "alter ego ID" is the same or greater than the "alter ego ID" received this time, the "alter ego" received this time is ignored.
If the received "alter ego ID" is greater than the processed "alter ego ID", the game program will proceed to the processing for using the "alter ego" received this time.
Using the received usable "alter ego data", the user performs some work on that alter ego in the game program on the receiving side.
Once the "alter ego data" performs work in the receiving side's game, InGame data that indicates that "Work was accomplished" is sent to the user who sent the gift using game boot messages (custom data attached).
The online ID and "alter ego ID" of the sender of the processed "alter ego data" are saved using save data, etc., in the game program.

(3) Sending Side
The sending side opens the game boot messages (custom data attached) sent from the receiving side user in the game program, and upon being informed that "Work was accomplished," this is reflected in the sending side's own game.

### Method to Use the Variables in the "near" Utility/"near" Dialog utility

**`SceNearGiftId`**

One `SceNearGiftId` is allocated per "alter ego ID" in the game program.

- [Flag that enables even users who do not own the title to be discovered (=0x80000000)]
  To make the game itself serve an advertising role: Enable (HIGH)
  If the alter ego's sending unit is too small, causing inconvenience to users: Disable (LOW).
- [Flag that enables the same item to be discovered multiple times if from different users (=0x40000000)]
  Enable (HIGH).

**`SceNearGiftCondition.`*`radius`***

0 m (use the critical distance determined on the server)

**`SceNearGiftCondition.`*`duration`***

0 hour (longest validity period determined by the "near" system).

**`SceNearGiftCondition.`*`receiverAttrs.playerRelation`***

Both nearby users and friends can be discovered (0x00000003 or 0x00000000)

**`SceNearGiftCondition.`*`probability`***

Set to 100%.

***`units`*`(sceNearSetGift() / sceNearSetGift2() / sceNearUtilitySetGift() / sceNearUtilitySetGift2())`**

Set 0xFFFFFFFF (unlimited).

# 6 FAQ

**(1) Given that gifts registered for distribution can be stored in the system for only 50 titles, how can one keep track of the gifts registered for distribution that are deleted? Also, how should one respond when gifts are deleted?**

Check by performing obtainment using `sceNearGetGift()`/`sceNearUtilityGetGift()` or `sceNearGetGiftStatus()`/`sceNearUtilityGetGiftStatus()`. When gifts are deleted, we recommend performing registration for distribution again on the game program side. To this end, we recommend storing duplicates of gift data registered for distribution.

**(2) Gifts cannot be sent/received using the "near" application started up from Visual Studio for debugging purpose.**

"near" sends and receives gifts of the user's 5 "Most recently played titles".

For development environments that use a version earlier than 1.800, in order to include an application in this 5 "Most recently played titles", it is first required to package the relevant application. Then, the packaged application must be installed on DevKit and started up from the home screen of DevKit. When being started up from Visual Studio for debugging purpose, such application does not satisfy the above conditions; therefore, it is not recognized as the 5 "Most recently played titles".

For development environments that use a version equal to or later than 1.800, it is not required to package the relevant application and install it on DevKit. An application started up in debugger mode from Visual Studio will be recognized as one of the 5 "Most recently played titles" and gift sending/receiving can be performed from the "near" application/"near" Dialog utility.

For details, refer to the "Operating Notes for Sample Program" section of the "near Utility Overview" and "near Dialog Utility Overview" documents.

**(3) Can SceNearGiftId be referenced on the receiving side?**

No. If behavior of both a game program and "near" application depends on the `SceNearGiftId` issued by a game program, handling of malfunctions may be difficult. Therefore, specifications are such that the game program side and the “near” application are forced to reference different parameters.

`SceNearGiftId` is the information set for the "near" application to identify a gift, and the game program on the receiving side cannot reference the `SceNearGiftId`. If information for identifying a gift is required in the game program on the receiving side, define such information for the gift data.

**(4) How will the game title sold in multiple areas behave when a different title ID is used by each area while a common NP Communication ID is shared?**

In this version, a game title is designed to behave using one NP Communication ID together with one title ID. However, it is possible to share a common NP Communication ID among the different titles if such request is submitted. The limitations currently applied are as follows.

- When starting up the game from the "Game Goods Details" screen in "near" application:
  When the title ID of the installed game is the same as that of a game that has sent the gift, the installed game can be started up.
  When the title ID of the installed game is different from that of a game that has sent the gift, the installed game can be started up from the "near" application screen if "near" utility/"near" Dialog utility is initialized by the game having a different title ID by using the same NP Communication ID.
- If multiple games register a gift to be sent using the same NP Communication ID, the previous registered gift will be overwritten with the following registered gifts.

**(5) When attempting to perform update with "near" application of DevKit/TestKit, a message saying "Unable to obtain location data." is displayed and sending/receiving gifts between accounts cannot be performed. How should one respond in such a case?**

In some cases, location information may not be obtained depending on the surrounding environment. It is possible to have the "near" application read the location information by making use of a GPS emulation file when location information cannot be obtained. For details, refer to the "liblocation Overview" document.