NP Matching 2 Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Purpose and Features

The NP Matching 2 library supports applications that implement network features, such as online game play and chats. Applications can use this library to utilize the NP Matching 2 system that is provided via PSN℠.

The NP Matching 2 library provides features for creating a session (collective name for lobby and room) among users who are connected to the network and for enabling messaging and the sharing information among these users. In addition, it also provides features to search for created sessions and support the establishment of P2P connections.

More specifically, the NP Matching 2 library provides the following features.

- Room search
- Dynamic room creation
- In-room chat
  A vulgarity filter (filter to conceal inappropriate language) can be enabled for this feature.
- Data exchange between users in a room
- Room entrance restriction
- Support for team play
- Static lobby setup
- In-lobby chat
  A vulgarity filter (filter to conceal inappropriate language) can be enabled for this feature.
- User search using the NP ID
- P2P connection establishment between room members

Embedding into Program

Include np.h in the source program. Various header files will be automatically included as well.

Load the PRX module in the program, as follows.

```
if ( sceSysmoduleLoadModule(SCE_SYSMODULE_NP_MATCHING2) != SCE_OK ) {
    // Error handling
}
```

Upon building the program, link libSceNpMatching2_stub.a.

Sample Programs

The following files are provided as sample programs that use the NP Matching 2 library for reference purposes.

sample_code/network/api_np/np_matching2/simple/

This program exemplifies the basic usage of the NP Matching 2 library, enabling the developer to easily understand the basic flow from initialization to room creation.

sample_code/network/api_np/np_matching2/room_sample/

This sample shows the usage procedure of the room feature.

sample_code/network/api_np/np_matching2/slot_reservation_sample/

This sample shows the implementation method for room slot reservation and invitation.

sample_code/network/api_np/np_matching2/signaling_rudp_sample/

This sample shows the usage procedure of the signaling feature and librudp.

sample_code/network/api_np/np_matching2/game_sample/

This sample shows how to implement the NP Matching 2 library, signaling feature and librudp into a game.

sample_code/network/api_np/np_matching2/lobby_sample/

This sample shows the lobby feature of the NP Matching 2 library.

Reference Materials

Refer to the following document for an overview of the PSN℠ functionalities.

- PSN℠ Overview

Refer to the following documents regarding the NP library, which is commonly required when using the PSN℠ functionalities.

- NP Library Overview
- NP Library Reference

Refer to the following document regarding Network Check Dialog for switching service states of the NP library.

- Network Overview

2 Configuration and Terms

This chapter explains the configuration and terms of the NP Matching 2 library.

Context

A context is positioned as the upper-most concept in the usage scheme of the NP Matching 2 library. An application that uses the NP Matching 2 library must first create a context, through which various features of the NP Matching 2 library can be accessed.

The NP Communication ID and NP communication passphrase - obtained when registering to use the NP and NP Matching 2 system - are required for creating a context. A 16-bit context ID will be assigned to a created context. This context ID will subsequently be used by the application to identify the context.

Event Callback Functions

In the NP Matching 2 library, execution results - when executing matching processing that entails access to the server, and events that occur within a session - such as a user joining or leaving, or receiving data, are notified as events in the callback functions that are registered by the application. There are 7 types of events, and their corresponding callback functions, as explained below.

For details on each event and callback function, refer to the "NP Matching 2 Library Reference" document.

Request Events and the Request Callback Function

Of the features that are provided by the NP Matching 2 library, those that access the server are called request functions. An event that indicates the completion of a request function is called a request event. A request event is notified to the request callback function.

The application must register a default request callback function before executing a request function, or specify a request callback function every time a request function is executed.

The macro used to represent request events and the type definition for the request callback function are as follows.

Request Event

```
SCE_NP_MATCHING2_REQUEST_EVENT_*
```

Request Callback Function

```
typedef void (*SceNpMatching2RequestCallback)(
    SceNpMatching2ContextId ctxId,
    SceNpMatching2RequestId reqId,
    SceNpMatching2Event event,
    int errorCode,
    const void *data,
    void *arg
    );
```

Room Events and the Room Event Callback Function

An event that occurs while the user is in a room is called a room event. A room event is notified to the room event callback function.

A room event can occur at any time while the user is joined into a room. The application must register the room event callback function before the user joins a room.

Events that can be notified as room events while the user is joined into a room are as follows.

- A new member joined the room
- A room member left the room
- A room member was kicked out from the room
- The room was destroyed
- The room owner changed
- Internal room data was updated
- Internal room member data was updated
- The signaling option parameter was updated

The macro used to represent room events and the type definition for the room event callback function are as follows.

Room Event

```
SCE_NP_MATCHING2_ROOM_EVENT_*
```

Room Event Callback Function

```
typedef void (*SceNpMatching2RoomEventCallback)(
    SceNpMatching2ContextId ctxId,
    SceNpMatching2RoomId roomId,
    SceNpMatching2Event event,
    const void *data,
    void *arg
    );
```

Room Message Events and the Room Message Callback Function

An event indicating that a room chat message or room message has been received while the user is joined into a room is called a room message event. A room message event is notified to the room message callback function.

A room message event can occur at any time while the user is joined into a room. The application must register the room message callback function before the user joins a room.

The macro used to represent room message events and the type definition for the room message callback function are as follows.

Room Message Event

```
SCE_NP_MATCHING2_ROOM_MSG_EVENT_*
```

Room Message Callback Function

```
typedef void (*SceNpMatching2RoomMessageCallback)(
    SceNpMatching2ContextId ctxId,
    SceNpMatching2RoomId roomId,
    SceNpMatching2RoomMemberId srcMemberId,
    SceNpMatching2Event event,
    const void *data,
    void *arg
    );
```

Lobby Events and the Lobby Event Callback Function

An event that occurs while the user is joined into a lobby is called a lobby event. A lobby event is notified to the lobby event callback function.

A lobby event can occur at any time while the user is joined into a lobby. The application must register the lobby event callback function before the user joins a lobby.

Events that can be notified as lobby events while the user is joined into a lobby are as follows.

- A new member joined the lobby
- A lobby member left the lobby
- The lobby was destroyed
- The internal lobby member data was updated

The macro used to represent lobby events and the type definition for the lobby event callback function are as follows.

Lobby Event

```
SCE_NP_MATCHING2_LOBBY_EVENT_*
```

Lobby Event Callback Function

```
typedef void (*SceNpMatching2LobbyEventCallback)(
   SceNpMatching2ContextId ctxId,
   SceNpMatching2LobbyId lobbyId,
   SceNpMatching2Event event,
   const void *data,
   void *arg
   );
```

Lobby Message Events and the Lobby Message Callback

An event indicating that a lobby chat message has been received while the user is joined into a lobby is called a lobby message event. A lobby message event is notified to the lobby message callback function.

A lobby message event can occur at any time while the user is joined into a lobby. The application must register the lobby message callback function before the user joins a lobby.

The macro used to represent lobby message events and the type definition for the lobby message callback function are as follows.

Lobby Message Event

```
SCE_NP_MATCHING2_LOBBY_MSG_EVENT_*
```

Lobby Message Callback Function

```
typedef void (*SceNpMatching2LobbyMessageCallback)(
   SceNpMatching2ContextId ctxId,
   SceNpMatching2LobbyId lobbyId,
   SceNpMatching2LobbyMemberId srcMemberId,
   SceNpMatching2Event event,
   const void *data,
   void *arg
   );
```

Signaling Events and the Signaling Callback Function

When a room has its signaling option parameter set, processing will be executed for that room that will allow P2P connections to be established between room members. When these connections are established or broken, a corresponding event called the signaling event is notified to the signaling callback function.

A signaling event can occur at any time while the user is joined into a room to which the signaling option parameter has been set. The application must register the signaling callback function before the user joins a room or before the signaling option parameter is set.

The macro used to represent signaling events and the type definition for the signaling callback function are as follows.

Signaling Event

```
SCE_NP_MATCHING2_SIGNALING_EVENT_*
```

Signaling Callback Function

```
typedef void (*SceNpMatching2SignalingCallback)(
    SceNpMatching2ContextId ctxId,
    SceNpMatching2RoomId roomId,
    SceNpMatching2RoomMemberId peerMemberId,
    SceNpMatching2Event event,
    int errorCode,
    void *arg
    );
```

Context Events and the Context Callback Function

An event indicating a change in the context state is called a context event. A context event is notified to the context callback function.

A context event occurs when the processing to start or stop the context completes, and when the context being used becomes unavailable. A context event can occur at any time after the context has been created. The application must register the context callback function immediately after creating the context.

The macro used to represent context events and the type definition for the context callback function are as follows.

Context Event

```
SCE_NP_MATCHING2_CONTEXT_EVENT_*
```

Context Callback Function

```
typedef void (*SceNpMatching2ContextCallback)(
    SceNpMatching2ContextId ctxId,
    SceNpMatching2Event event,
    SceNpMatching2EventCause eventCause,
    int errorCode,
    void *arg
    );
```

Internal Thread

1 internal thread exists for the NP Matching 2 library. This thread creates data required for event notification to the application upon an event occurrence, and calls the appropriate callback function.

The internal thread of the NP Matching 2 library is created upon the library's initialization, and deleted upon its termination. The stack size and priority of the internal thread can also be set upon initialization.

Heap Area

The NP Matching 2 library has its own heap area.

Memory usage of the NP Matching 2 library largely depends on the application using it. Execute `sceNpMatching2GetMemoryInfo()` to obtain information on the memory area that is being used as the heap area. Upon developing your application, execute `sceNpMatching2GetMemoryInfo()` to check how much memory is being used by your application.

3 Operation Overview and Characteristic Usage

This chapter provides an overview of the library's operations, and an explanation of the usage procedure that is characteristic and common to, several of the processing.

Information and Attributes to Be Referenced (Internal or External to a Session)

Basic information and attributes of components belonging to the NP Matching 2 library can be separated into 2 types: those for referencing from within a session and those for referencing from outside a session (for more details, refer to the "NP Matching 2 System Overview" document).

In the NP Matching 2 library, structures representing the information and attributes of components belonging to the NP Matching 2 system - as well as the functions for accessing them - are separately defined for those that are referenced internally and those that are referenced externally. Naming rules apply according to this distinction. This is also true for macro definitions.

The names of elements representing information and attributes to be externally referenced from outside a session are defined as follows.

```
"(Name representing the information/attribute of a component)" + "External"
```

The names of elements representing information and attribute to be internally referenced from within a session are defined as follows.

```
"(Name representing information/attribute of a component)" + "Internal"
```

For example, information about a room to be referenced from outside the room is represented by the structure, `SceNpMatching2RoomDataExternal`. Information about a room to be referenced from inside the room is represented by the structure, `SceNpMatching2RoomDataInternal`. Functions for obtaining these information are `sceNpMatching2GetRoomDataExternalList()` and `sceNpMatching2GetRoomDataInternal()`, respectively.

Functions accessing session information or attributes for internal/external reference and the related macros are also defined by including the above component names.

Initial Attribute Values and Attribute Representation

The components of the NP Matching 2 system have different attribute types: flag attribute, integer attribute, and binary attribute. This section describes the initial values of each attribute - in other words, the values before the application sets them for each component - and the representation used for these attributes in the NP Matching 2 library.

Flag Attribute

Bit values are set to represent the setting of each component upon its creation. This initial value must be saved on the application-side if you ever wish to return a flag attribute value to its initial setting.

Integer Attribute

0 is stored for an integer attribute in its initial state. When initially obtaining the integer attribute value, the values of members of the `SceNpMatching2IntAttr` structure, which represents an integer attribute, will be as follows.

Structure Member	Value
`id`	Attribute ID representing the obtained integer attribute
`num`	0

To return the integer attribute to its initial setting from the application, return the members of the `SceNpMatching2IntAttr` structure to the same values as the above table.

Binary Attribute

A binary data of size 0 is stored for an attribute value in its initial state. When initially obtaining the binary attribute value, the members of the `SceNpMatching2BinAttr` structure, which represents a binary attribute, will be as follows.

Structure Member	Value
`id`	Attribute ID representing the obtained binary attribute
`ptr`	NULL
`size`	0

To return the binary attribute to its initial setting from the application, return the members of the `SceNpMatching2BinAttr` structure to the same values as the above table.

Request Functions

Of the features provided by the NP Matching 2 library, those that access the server are called request functions. Although there are various request functions, the basic usage is the same for all of them.

This section explains the procedure for calling a request function using an example of a pseudo request function, `sceNpMatching2`***`RequestFunc`***`()`.

The processing of a request function entails a network access. However, because request functions are asynchronous, control immediately returns when a request function is successfully called. 0 is returned when the call of a request function is successful; a negative value is returned upon error.

When the processing executed by the request function completes, a request event corresponding to the executed request function is notified to the request callback function. The names of macro definitions for request events corresponding to the request functions are defined according to the following rule.

```
SCE_NP_MATCHING2_REQUEST_EVENT_REQUEST_FUNC
```

The success/failure of the request function's processing can be determined by the value of the *`errorCode`* argument of the request callback function. When processing is successful, this value is 0. When processing fails, this value is a negative value.

When processing of the request function does not end within a certain period of time, the processing will be aborted and the `SCE_NP_MATCHING2_ERROR_REQUEST_TIMEOUT` time-out error will be notified to the request callback function.

The arguments of request functions are defined by a common format, as follows.

```
sceNpMatching2RequestFunc(
    //E Context ID
    const SceNpMatching2ContextId ctxId,
    //E Request parameters
    const SceNpMatching2RequestFuncRequest *reqParam,
    //E Request option parameters
    const SceNpMatching2RequestOptParam *optParam,
    //E Pointer to area for storing the request ID
    SceNpMatching2RequestId *assignedReqId
    );
```

Request ID

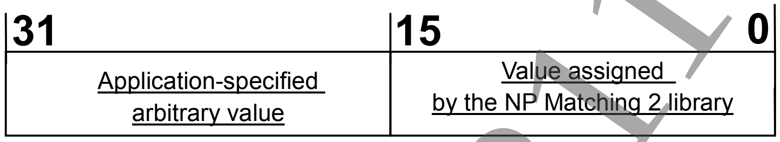
A 32-bit request ID is allocated to a request when it is created with the execution of a request function. The allocated request ID is copied to the area indicated by the request function's *`assignedReqId`* argument.

The request ID of the applicable request is also stored in the *`reqId`* argument of the request callback function. The application can compare the request ID stored in the *`assignedReqId`* argument of the request function, with the request ID of the *`reqId`* argument of the request callback function, to determine which request the notified request event corresponds to.

A request ID consists of a lower 16-bit value assigned by the NP Matching 2 library, and an upper 16-bit value arbitrarily specified by the application. Specification of the upper 16 bits by the application enables the application to set up its own rules when grouping the request IDs, such as, upon queuing for events.

To specify a value for the upper 16-bits of the request ID, specify the value as a request option parameter in the argument of the request function.

Figure 1 Request ID Structure


31
15
0
Application-specified arbitrary value
Value assigned by the NP Matching 2 library

Request Parameters

When executing a request function, specify request parameters unique to the request in the argument of the request function. Request parameters are represented as a structure with a name pursuant to the common naming rule. 1 request parameter structure is defined for 1 request function. A structure representing request parameters is defined as follows.

```
typedef struct SceNpMatching2RequestFuncRequest {
    . . . .
} SceNpMatching2RequestFuncRequest;
```

Event Data (Response Data)

Some request events corresponding to request functions are accompanied by event data. Event data accompanying a request event is called response data.

Response data is represented by a structure with a name pursuant to the common naming rule. 1 response data structure is defined for 1 request function. A structure representing response data is defined as follows.

```
typedef struct SceNpMatching2RequestFuncResponse {
    . . . .
} SceNpMatching2RequestFuncResponse;
```

Request Option Parameters

Request option parameters are represented by a structure indicating common parameters for all request functions. The specification of request option parameters is optional; it can be specified as necessary. A structure representing request option parameters is defined as follows.

```
typedef struct SceNpMatching2RequestOptParam {
    SceNpMatching2RequestCallback cbFunc;
    void *cbFuncArg;
    SceUInt32 timeout;
    SceUShort16 appReqId;
    SceUChar8 padding[2];
} SceNpMatching2RequestOptParam;
```

The `cbFunc` structure member represents the request callback function for notifying the completion of the executed request function with the specification of request option parameters, and `cbFuncArg` represents arbitrary data to be passed to the request callback function. The use of the request callback function specified here will be prioritized over the default request callback function registered to the context. For details, refer to the subsequent section "Specifying the Request Callback Function and the Priority Order".

The `timeout` structure member represents the time-out time for the processing to be executed by the request function. The timeout time specified here is prioritized over the default time-out time that is registered to the context. For details, refer to the subsequent section "Specifying the Time-out Time and the Priority Order".

The `appReqId` structure member represents the value for the upper 16 bits of the request ID.

Specifying the Request Callback Function and the Priority Order

Before executing a request function, the application must have a request callback function registered for receiving notification on the processing completion of a request. There are 2 ways to register the request callback function to the NP Matching 2 library, as follows.

- Register a default request callback function to the context
- Specify a request callback function upon executing the request function

`sceNpMatching2SetDefaultRequestOptParam()` registers a default request callback function to the context. This registered callback function will subsequently be used as the common request callback function for all request functions.

It is also possible to specify a request callback function exclusively for a request, by specifying the request option parameters upon executing a request function. The request callback function specified upon executing the request function will only be used to notify the request event of that request function's processing.

If a request callback function is not specified upon executing the request function, the default request callback function registered to the context will be used. The priority of the request callback function used is as shown in Table 1.

Specifying the Time-out Time and the Priority Order

Request processing to be carried out from a request function has a time-out time set to it. The default setting in the NP Matching 2 library is 10 seconds. The application can also set its own time-out time. There are 2 ways to make this setting, as follows.

- Register a default request time-out time to the context
- Specify a time-out value upon executing the request function

`sceNpMatching2SetDefaultRequestOptParam()` registers a default request time-out time to the context. This registered time-out time will subsequently be used for all request functions.

It is also possible to specify a time-out time exclusively for a request by specifying the request option parameters upon executing a request function. The time-out time specified upon executing the request function will only be applied to that request function's processing.

If a time-out time is not specified upon executing the request function, the default time-out time registered to the context will be used. If a default time-out time is also not registered to the context, the default time-out time of the NP Matching 2 library will be used. The priority order by which time-out values will be used are as shown in Table 1.

Table 1 Priority Order of Request Callback Function and Time-out Times

Priority	Request Callback Function and Arbitrary Data Passed to Argument *arg*	Time-out Time
1	Value specified in the request option parameter upon executing the request function	
2	Default value registered to the context using `sceNpMatching2SetDefaultRequestOptParam()`	
3	None	Default value of the NP Matching 2 library (10 seconds)

For details on each request function, refer to the "NP Matching 2 Library Reference" document.

Events and Event Data

The events of the NP Matching 2 library and their corresponding event data types are as follows.

Request Events

Event	Event Data Type
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_WORLD_INFO_LIST`	`SceNpMatching2GetWorldInfoListResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_ROOM_MEMBER_DATA_EXTERNAL_LIST`	`SceNpMatching2GetRoomMemberDataExternalListResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_SET_ROOM_DATA_EXTERNAL`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_ROOM_DATA_EXTERNAL_LIST`	`SceNpMatching2GetRoomDataExternalListResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_LOBBY_INFO_LIST`	`SceNpMatching2GetLobbyInfoListResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_SET_USER_INFO`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_USER_INFO_LIST`	`SceNpMatching2GetUserInfoListResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_CREATE_JOIN_ROOM`	`SceNpMatching2CreateJoinRoomResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_JOIN_ROOM`	`SceNpMatching2JoinRoomResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_ROOM`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_GRANT_ROOM_OWNER`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_KICKOUT_ROOM_MEMBER`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_SEARCH_ROOM`	`SceNpMatching2SearchRoomResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_SEND_ROOM_CHAT_MESSAGE`	`SceNpMatching2SendRoomChatMessageResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_SEND_ROOM_MESSAGE`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_SET_ROOM_DATA_INTERNAL`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_ROOM_DATA_INTERNAL`	`SceNpMatching2GetRoomDataInternalResponse`
`SCE_NP_MATCHING2_REQUEST_EVENT_SET_ROOM_MEMBER_DATA_INTERNAL`	No event data
`SCE_NP_MATCHING2_REQUEST_EVENT_GET_ROOM_MEMBER_DATA_INTERNAL`	`SceNpMatching2GetRoomMemberDataInternalResponse`

Event	Event Data Type
SCE_NP_MATCHING2_REQUEST_EVENT_SET_SIGNALING_OPT_PARAM	No event data
SCE_NP_MATCHING2_REQUEST_EVENT_JOIN_LOBBY	SceNpMatching2JoinLobbyResponse
SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_LOBBY	No event data
SCE_NP_MATCHING2_REQUEST_EVENT_SEND_LOBBY_CHAT_MESSAGE	SceNpMatching2SendLobbyChatMessageResponse
SCE_NP_MATCHING2_REQUEST_EVENT_SET_LOBBY_MEMBER_DATA_INTERNAL	No event data
SCE_NP_MATCHING2_REQUEST_EVENT_GET_LOBBY_MEMBER_DATA_INTERNAL	SceNpMatching2GetLobbyMemberDataInternalResponse
SCE_NP_MATCHING2_REQUEST_EVENT_GET_LOBBY_MEMBER_DATA_INTERNAL_LIST	SceNpMatching2GetLobbyMemberDataInternalListResponse

Room Events

Event	Event Data Type
SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_JOINED	SceNpMatching2RoomMemberUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT	SceNpMatching2RoomMemberUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_KICKEDOUT	SceNpMatching2RoomUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED	SceNpMatching2RoomUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_ROOM_OWNER_CHANGED	SceNpMatching2RoomOwnerUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_UPDATED_ROOM_DATA_INTERNALL	SceNpMatching2RoomDataInternalUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_UPDATED_ROOM_MEMBER_DATA_INTERNAL	SceNpMatching2RoomMemberDataInternalUpdateInfo
SCE_NP_MATCHING2_ROOM_EVENT_UPDATED_SIGNALING_OPT_PARAM	SceNpMatching2SignalingOptParamUpdateInfo

Room Message Events

Event	Event Data Type
SCE_NP_MATCHING2_ROOM_MSG_EVENT_CHAT_MESSAGE	SceNpMatching2RoomMessageInfo
SCE_NP_MATCHING2_ROOM_MSG_EVENT_MESSAGE	SceNpMatching2RoomMessageInfo

Lobby Events

Event	Event Data Type
SCE_NP_MATCHING2_LOBBY_EVENT_MEMBER_JOINED	SceNpMatching2LobbyMemberUpdateInfo
SCE_NP_MATCHING2_LOBBY_EVENT_MEMBER_LEFT	SceNpMatching2LobbyMemberUpdateInfo
SCE_NP_MATCHING2_LOBBY_EVENT_LOBBY_DESTROYED	SceNpMatching2LobbyUpdateInfo
SCE_NP_MATCHING2_LOBBY_EVENT_UPDATED_LOBBY_MEMBER_DATA_INTERNAL	SceNpMatching2LobbyMemberDataInternalUpdateInfo

Lobby Message Events

Event	Event Data Type
`SCE_NP_MATCHING2_LOBBY_MSG_EVENT_CHAT_MESSAGE`	`SceNpMatching2LobbyMessageInfo`

Signaling Events

Signaling events do not have event data.

Context Events

Context events do not have event data.

4 Processing Flow and Examples

This chapter explains the typical processing flow for applications that use the NP Matching 2 library. Examples are also provided. For the code examples in this chapter, the following variable is supposed to be declared for storing the return value of functions.

```
int ret;
```

Loading the PRX

Call `sceSysmoduleLoadModule()` with `SCE_SYSMODULE_NP_MATCHING2` specified as the module ID to load the PRX.

Initializing the NP Library

Initialize the NP library by `sceNpInit()`. Set the NP Communication ID, NP communication passphrase and NP communication signature to the `SceNpCommunicationConfig` structure. These values are issued per application by applying on PlayStation®Vita Developer Network (https://psvita.scedev.net/). Be sure to set the issued values correctly.

Initializing the NP Matching 2 Library

Execute `sceNpMatching2Init()` to initialize the NP Matching 2 library.

The initialization of the NP Matching 2 library entails the following.

(1) Creation of an internal thread

(2) Creation of heap area to be used by the NP Matching 2 library

Upon executing `sceNpMatching2Init()`, specify the stack size and priority of the internal thread. Specify 0 to use the default settings.

```
//E Initialize NP Matching 2 Library
ret = sceNpMatching2Init(SCE_NP_MATCHING2_POOLSIZE_DEFAULT,
    SCE_NP_MATCHING2_THREAD_PRIORITY_DEFAULT,
    SCE_KERNEL_THREAD_CPU_AFFINITY_MASK_DEFAULT,
    SCE_NP_MATCHING2_THREAD_STACK_SIZE_DEFAULT);
if (ret < 0) {
    //E Error handling
}
```

Creating a Context

Create a context for accessing the various features of the NP Matching 2 library. Upon creating the context, specify the NP ID of the user already signed onto the NP, as well as the NP Communication ID and NP communication passphrase obtained upon registering to use the NP and the NP Matching 2 system. At this time, if NULL is specified as argument for NP Communication ID and NP communication passphrase, the value specified in `sceNpInit()` will be used. Specify target NP Communication IDs and NP communication passphrases other than NULL when using multiple NP Communication IDs.

```
SceNpMatching2ContextId ctxId = 0;
SceNpId npId; //E NP ID of the signed-in user

//E Create context
ret = sceNpMatching2CreateContext(
       &npId, NULL, NULL, &ctxId);
if (ret < 0) {
    //E Error handling
}

//E The context ID of the created context will be stored in ctxId
```

Registering the Context Callback Function

Once a context has been created, make sure to register the context callback function. The context may become unavailable at any time; the context callback function and its registration are required for the application to detect state transitions of the context and to carry out the appropriate processing in response to a transition.

To register the context callback function, execute `sceNpMatching2RegisterContextCallback()`.

```
    //E Context callback function
    void
    context_cb(
        SceNpMatching2ContextId id,
        SceNpMatching2Event event,
        SceNpMatching2EventCause eventCause,
        int errorCode,
        void *arg
        )
    {

        . . . .

        if (event == SCE_NP_MATCHING2_CONTEXT_EVENT_START_OVER) {
            //E Processing when the context is in the unavailable state
        }
        else if (event == SCE_NP_MATCHING2_CONTEXT_EVENT_STARTED) {
            //E Processing for the completion of the context start processing
        }
        else if (event == SCE_NP_MATCHING2_CONTEXT_EVENT_STOPPED) {
            //E Processing for the completion of the context stop processing
        }

        . . . .

    }

    //E Register the context callback function
    ret = sceNpMatching2RegisterContextCallback(
            context_cb, NULL);
    if (ret < 0) {
        ///E Error handling
    }
```

Starting the Context

To use the created context, execute processing to change the context state to STARTED. Execute `sceNpMatching2ContextStart()` to start the context.

`sceNpMatching2ContextStart()` is a non-blocking function. After its execution, wait for the completion of the start processing to be notified to the context callback function. When using

`sceNpMatching2ContextStart()` to start the context, make sure that the context callback function is registered in advance.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;

//E Context callback function
void
context_cb(
   SceNpMatching2ContextId id,
   SceNpMatching2Event event,
   SceNpMatching2EventCause eventCause,
   int errorCode,
   void *arg
   )
{
   if (id != ctxId)
      return;

   if (event == SCE_NP_MATCHING2_CONTEXT_EVENT_STARTED) {
      if (errorCode < 0)
         //E Error handling
      else
         //E Context start processing completed
   }
}

//E Register the context callback function
ret = sceNpMatching2RegisterContextCallback(
        context_cb, NULL);
if (ret < 0) {
   //E Error handling
}

//E Start the context
//   Set time-out time to 10 seconds
ret = sceNpMatching2ContextStart (ctxId, (10*1000*1000));
if (ret < 0) {
   //E Error handling
}

//E Wait for SCE_NP_MATCHING2_CONTEXT_EVENT_STARTED to be notified
//  to the context callback function
```

Setting the Default Request Option Parameters

Set the default request option parameters - the default request callback function and the request time-out time - as necessary, with `sceNpMatching2SetDefaultRequestOptParam()`.

To register the request callback function and to set a time-out time of 20 seconds, execute as follows.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;

SceNpMatching2RequestOptParam optParam;

//E Request callback function
void
default_request_cb(
   SceNpMatching2ContextId id,
   SceNpMatching2RequestId reqId,
   SceNpMatching2Event event,
   int errorCode,
   const void *data,
   void *arg
   )
{

   . . . .

   if (event == SCE_NP_MATCHING2_REQUEST_EVENT_GET_WORLD_INFO_LIST) {
       //E Processing for the completion of sceNpMatching2GetWorldInfoList()
   }
   else if (event == SCE_NP_MATCHING2_REQUEST_EVENT_SET_USER_INFO) {
       //E Processing for the completion of sceNpMatching2SetUserInfo()
   }
   else if (    . . . .

   . . . .

}

memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = default_request_cb;
optParam.cbFuncArg = NULL;
optParam.timeout = (20 * 1000 * 1000);
optParam.appReqId = 0;

ret = sceNpMatching2SetDefaultRequestOptParam(ctxId, &optParam);
if (ret < 0) {
   //E Error handling
}
```

Registering Other Callback Functions

Register other context callback functions, as necessary.

For details on each callback function registering function, refer to the "NP Matching 2 Library Reference" document.

Getting Server Information

Getting Server Information and Confirming the Server Status

Before an application accesses a server, it must first obtain server information and make sure that the status of the target server is "Available". To obtain the server information, execute `sceNpMatching2GetServerLocal()`.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2Server server;

ret = sceNpMatching2GetServerLocal(ctxId, &server);
if(ret < 0) {
    //E Error handling
}
```

Selecting a World

Upon registering to use the NP Matching 2 system, the number of worlds belonging to the server can be set according to the design of your application. After specifying the number of worlds belonging to the server, select the world to use. It is up to the application to either let the user make a selection from a displayed list of worlds, or to select one automatically.

Execute `sceNpMatching2GetWorldInfoList()` to obtain a list of information regarding worlds belonging to the selected server. It is also possible to set the session or number of session members (included in the world information) as a criteria to select a world.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2RequestId assignedReqId;
SceNpMatching2GetWorldInfoListResponse *respData = NULL;

//E Request callback function
void
getWorldInfoList_request_cb(
    SceNpMatching2ContextId id,
    SceNpMatching2RequestId reqId,
    SceNpMatching2Event event,
    int errorCode,
    const void *data,
    void *arg
    )
{
    if (id != ctxId || reqId != assignedReqId)
        return;

    if (event != SCE_NP_MATCHING2_REQUEST_EVENT_GET_WORLD_INFO_LIST)
        return;

    if (errorCode < 0) {
        //E Error handling when world information list obtainment fails
    }

    respData = (SceNpMatching2GetWorldInfoListResponse *)data;
    if (respData == NULL) {
        //E Error handling
    }

    //E Processing for the completion of sceNpMatching2GetWorldInfoList()

    return;
}


//E Assuming that the server ID of the selected server is stored:
SceNpMatching2ServerId serverId;
SceNpMatching2GetWorldInfoListRequest reqParam;
SceNpMatching2RequestOptParam optParam;

//E Request parameters
memset(&reqParam, 0, sizeof(reqParam));
reqParam.serverId = serverId;

//E Request option parameters
memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = getWorldInfoList_request_cb;
```

```
ret = sceNpMatching2GetWorldInfoList(
   ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   //E Error handling
}

//E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_GET_WORLD_INFO_LIST to be
//   notified to the request callback function

//E Use obtained world information as a criteria and select world
```

For details on the above functions, refer to the "NP Matching 2 Library Reference" document.

Creating a Room

In an application that uses the NP Matching 2 library, messaging and information sharing is possible between users who are in a common session (room or lobby). To join a room, either create your own room or join one that has been created by another user. This section describes how to create your own room.

Parameters for Room Creation

Create a room belonging to the selected world. To create a room, execute `sceNpMatching2CreateJoinRoom()`. This function enables you to create a new room, and then join it.

Specify the request parameters for determining the configuration of that room upon a room's creation. Request parameters representing the room configuration are as follows.

Request Parameter	Structure Member	Specification	Change after Creation
Total number of slots (maximum 64)	*maxSlot*	Required	Not possible
Room flag attributes	*flagAttr*	Optional	Possible
Room password	*roomPassword*	Optional	Not possible
Room group configuration	*groupConfig*	Optional	Only possible to enable/disable password
Room password slot mask	*passwordSlotMask*	Optional	Possible
Users allowed to join	*allowedUser*	Optional	Not possible
Users blocked from joining	*blockedUser*	Optional	Not possible
Signaling option parameter	*sigOptParam*	Optional	Possible

To change the room configuration after a room has been created, execute `sceNpMatching2SetRoomDataInternal()`.

In addition, initial values for room attributes can be set upon a room's creation. The request parameters to specify here are as follows.

Request Parameter	Structure Member	Specificatio n	Changes after Creation
Internal room binary attribute	*roomBinAttrInternal*	Optional	Possible
External room search integer attribute	*roomSearchableIntAttrExternal*	Optional	Possible
External room search binary attribute	*roomSearchableBinAttrExternal*	Optional	Possible
External room binary attribute	*roomBinAttrExternal*	Optional	Possible

To change the room attributes after a room has been created, execute `sceNpMatching2SetRoomDataInternal()` and `sceNpMatching2SetRoomDataExternal()`.

Because `sceNpMatching2CreateJoinRoom()` enables you to create a room and then join it, a request parameter for joining the room must also be specified, as follows.

Request Parameter	Structure Member	Specification
Group label required to join a room group	*joinRoomGroupLabel*	Required for a group room

The room creator will join the number 1 slot of the created room; specify the group label required to join the room group containing the number 1 slot.

Initial values for the room member data can be set upon joining the room. Specify the request parameters, as follows.

Request Parameters	Structure Member	Specification	Change after Creation
Internal room member binary attribute	*roomMemberBinAttrInternal*	Optional	Possible
Team ID	*teamId*	Optional	Possible

To change the room member data after creating and joining the room, execute `sceNpMatching2SetRoomMemberDataInternal()`.

Code Example for Setting the Request Parameters

The following is a code example for setting the request parameters for `sceNpMatching2CreateJoinRoom()`.

```
//E Request parameters
SceNpMatching2CreateJoinRoomRequest reqParam;

memset(&reqParam, 0, sizeof(reqParam));
```

Specify Space for Creating a Room

Specify the world ID of the world, or the lobby ID of the lobby, to which the room to be created will belong. When a lobby ID is specified, the world ID - if specified - will be ignored, and a room belonging to the specified lobby will be created. 0 represents no specification. An error will occur if 0 is specified for both the world ID and the lobby ID.

```
//E Assuming that the world ID of a selected world is stored:
SceNpMatching2WorldId worldId;

reqParam.worldId = worldId;
reqParam.lobbyId = 0;
```

Specify the Total Number of Slots

Create a room with a capacity to hold 16 members maximum.

```
reqParam.maxSlot = 16;
```

Specify Room Flag Attributes

Specify room flag attributes. Specify the flag attributes you want to enable by logical OR. To disable all room flags, specify 0.

```
//E Enable flag to automatically transfer room ownership and to restrict room
//  joining by the NAT type
reqParam.flagAttr = (SCE_NP_MATCHING2_ROOM_FLAG_ATTR_OWNER_AUTO_GRANT |
                     SCE_NP_MATCHING2_ROOM_FLAG_ATTR_NAT_TYPE_RESTRICTION);
```

Specify Room Password

When setting a room password to a room, specify the room password, as follows. Specify NULL to disable passwords.

```
SceNpMatching2SessionPassword password;

memset(&password, 0, sizeof(password));
strncpy(password.data, "ROOMPSWD", SCE_NP_MATCHING2_SESSION_PASSWORD_SIZE);

reqParam.roomPassword = &password;
```

Specify Room Group Configuration

When creating a group room, specify the room group configuration. To create a room without room groups, specify NULL here.

The room group configuration is represented by an array of `SceNpMatching2RoomGroupConfig` structures, with the number of room groups as the number of elements. Structures correspond to room groups, in ascending order of array indices and slot numbers. The *`slotNum`* of each `SceNpMatching2RoomGroupConfig` structure in the array represents the number of slot numbers in the room group, and the total sum of *`slotNum`* of all the structures must match the total number of slots for the room (*`maxSlot`*).

```
//E Create a room with 2 room groups, each with 8 slots
SceNpMatching2RoomGroupConfig groupConfig[2];

memset(groupConfig, 0, sizeof(groupConfig));

//E Set room group with group ID 1 as its ID
//  Include slot numbers 1 to 8 for this room group
//  Set group label. Only users presenting "LABEL01" label can join.
//  Enable room password. Only users presenting room password set in
//  reqParam.roomPassword can join this room group.
groupConfig[0].slotNum = 8;
groupConfig[0].withLabel = true;
strncpy(groupConfig[0].label.data, "LABEL01",
        SCE_NP_MATCHING2_GROUP_LABEL_SIZE);
groupConfig[0].withPassword = true;

//E Set room group with group ID 2 as its ID
//  Include slot numbers 9 to 16 for this room group
//  Do not set group label. Group label presented by the first user to join room
//  group will automatically be set as this room group's label.
//  Disable room password.
groupConfig[1].slotNum = 8;
groupConfig[1].withLabel = false;
groupConfig[1].withPassword = false;

//E Set request parameters
reqParam.groupConfig = groupConfig;
reqParam.groupConfigNum = 2;
```

Specify the Room Password Slot Mask

When creating a room that is not a group room (`reqParam.groupConfig` = NULL) and setting a password, the password can be enabled/disabled per slot. Specify the room password slot mask. Specify NULL when creating a group room or when not setting a password.

To create a room password slot mask to be specified as a request parameter, use a macro to add a slot number to the room password slot mask, and add slot numbers of the slots you want to enable the room password for to the room password slot mask.

```
SceNpMatching2RoomPasswordSlotMask slotMask;

memset(&slotMask, 0, sizeof(slotMask));

//E Enable room password for slots with slot numbers 9 to 16
for (int i = 9; i <= 16; i++) {
    SCE_NP_MATCHING2_ADD_SLOTNUM_TO_ROOM_PASSWORD_SLOT_MASK(slotMask, i);
}

reqParam.passwordSlotMask = &slotMask;
```

Specify Users Who Are Allowed to Join a Room and Users to Be Blocked from Joining

After you've created a room and set a password for it, and you want certain users to join a slot or room group without having to present the room's password, specify these users by their NP IDs. Likewise, if you want to prevent certain users from joining the room, set them as users blocked from joining the room. Specify the NP IDs of allowed users and blocked users as an array to the request parameters.

```
SceNpId allowedUser[SCE_NP_MATCHING2_ROOM_ALLOWED_USER_MAX];
SceNpId blockedUser[SCE_NP_MATCHING2_ROOM_BLOCKED_USER_MAX];

for (int i = 0; i < SCE_NP_MATCHING2_ROOM_ALLOWED_USER_MAX; i++) {
    //E Copy the NP ID of users registered as friends, or
    //  NP IDs of users obtained by the application to
    //  allowdUser[i].
}

for (i = 0; i < SCE_NP_MATCHING2_ROOM_BLOCKED_USER_MAX; i++) {
    //E Copy the NP ID of users on the block list, or
    //  the NP IDs of users obtained by the application to
    //  blockedUser[i].
}

reqParam.allowedUser = allowedUser;
reqParam.allowedUserNum = SCE_NP_MATCHING2_ROOM_ALLOWED_USER_MAX;
reqParam.blockedUser = blockedUser;
reqParam.blockedUserNum = SCE_NP_MATCHING2_ROOM_BLOCKED_USER_MAX;
```

Specify the Signaling Option Parameter

When establishing P2P connection between room members immediately after creating a room, specify the signaling option parameter. Every time a new member joins the room, the processing to establish P2P connection will be started automatically. For details, refer to the subsequent section "Establishing P2P Connection".

```
SceNpMatching2SignalingOptParam sigOptParam;

//E Set star format for the signaling topology with the room owner as hub
memset(&sigOptParam, 0, sizeof(sigOptParam));
sigOptParam.type = SCE_NP_MATCHING2_SIGNALING_TYPE_STAR;
sigOptParam.hubMemberId = 0; //E  0 Indicates the room owner
```

```
reqParam.sigOptParam = &sigOptParam;
```

Specify Initial Values for Room Attributes

Specify the initial room attribute values upon the room's creation, as necessary. Set information to be shared between room members as internal room attributes, and information to be shared with an outsider as external room attributes. For example, set information unique to the application and representing the characteristics of the room, such as "stages" and "languages" as external room search attributes, so that a user searching for a room can use this information as search conditions to filter rooms. To set initial values to multiple attributes, specify structures representing attributes as an array to the request parameters.

```
#define STAGE_1 (1)
#define STAGE_2 (2)

#define LANG_ENGLISH  (1)
#define LANG_JAPANESE (2)

SceNpMatching2IntAttr searchableIntAttrExternal[2];
SceNpMatching2BinAttr binAttrInternal;
char binAttrInternalData[SCE_NP_MATCHING2_ROOM_BIN_ATTR_INTERNAL_MAX_SIZE];

memset(searchableIntAttrExternal, 0, sizeof(searchableIntAttrExternal);
memset(binAttrInternal, 0, sizeof(binAttrInternal);
memset(binAttrInternalData, 0, sizeof(binAttrInternalData));

//E Set information to serve as search conditions to the external room search
//  integer attributes.
//  Define external room search integer attribute ID 1 as "stages".
//  Define external room search integer attribute ID 2 as "languages".
searchableIntAttrExternal[0].id =
    SCE_NP_MATCHING2_ROOM_SEARCHABLE_INT_ATTR_EXTERNAL_1_ID;
searchableIntAttrExternal[0].num = STAGE_1;
searchableIntAttrExternal[1].id =
    SCE_NP_MATCHING2_ROOM_SEARCHABLE_INT_ATTR_EXTERNAL_2_ID;
searchableIntAttrExternal[1].num = LANG_ENGLISH;

//E Set information to be shared among room members to internal room binary
//  attribute ID 1.
strncpy(binAttrInternalData, "SHARED_DATA",
    SCE_NP_MATCHING2_ROOM_BIN_ATTR_INTERNAL_MAX_SIZE);
binAttrInternal.id = SCE_NP_MATCHING2_ROOM_BIN_ATTR_INTERNAL_1_ID;
binAttrInternal.ptr = binAttrInternalData;
binAttrInternal.size = strlen(binAttrInternalData);

//E Set request parameters
reqParam.roomBinAttrInternal = &binAttrInternal;
reqParam.roomBinAttrInternalNum = 1;
reqParam.roomSearchableIntAttrExternal = searchableIntAttrExternal;
reqParam.roomSearchableIntAttrExternalNum = 2;

reqParam.roomSearchableBinAttrExternal = NULL;
reqParam.roomSearchableBinAttrExternalNum = 0;
reqParam.roomBinAttrExternal = NULL;
reqParam.roomBinAttrExternalNum = 0;
```

Specify the Group Label Required to Join the Room

A user joining a group room must present a group label. When you are creating a group room (*reqParam.groupConfig* != NULL), specify the group label that must be presented to join that group room. Because the room creator always occupies the number 1 slot, specify the group label required to join the room group containing slot number 1 (room group with group ID 1 as its ID). If a group label for group ID 1 was specified upon setting the room group configuration when the room was created, specify the same group label here. If a group label was not set then, any group label can be specified. Because the room creator presents a group label and joins the room, the group label for the group ID 1 room group will be the one presented by the room creator. Specify NULL if the room is not a group room.

```
SceNpMatching2GroupLabel groupLabel;

//E Because the group label was set upon creating the room, specify the
//  group label set to the group ID 1 room group
memset(&groupLabel, 0, sizeof(groupLabel));
strncpy(groupLabel.data, "LABEL01",
        SCE_NP_MATCHING2_GROUP_LABEL_SIZE);

reqParam.joinRoomGroupLabel = &groupLabel;
```

Specify Initial Values for the Room Member Data

As necessary, set the initial values for room member data upon creating the room and joining it. Use the internal room member binary attributes for sharing information exclusively among room members.

```
SceNpMatching2BinAttr memberBinAttrInternal;
char memberBinData[SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_MAX_SIZE];

memset(&memberBinAttrInternal, 0, sizeof(memberBinAttrInternal));
memset(memberBinData, 0, sizeof(memberBinData));

//E Set common room member data to the internal room member binary attribute
//  ID 1.
strncpy(memberBinData, "NOT AVAILABLE",
   SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_MAX_SIZE);
memberBinAttrInternal.id =
   SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_1_ID;
memberBinAttrInternal.ptr = memberBinData;
memberBinAttrInternal.size = strlen(memberBinData);

reqParam.roomMemberBinAttrInternal = &memberBinAttrInternal;
reqParam.roomMemberBinAttrInternalNum = 1;
reqParam.teamId = 0;
```

Execute the Request Function

Specify the set request parameters and execute `sceNpMatching2CreateJoinRoom()`. After executing this function, wait for `SCE_NP_MATCHING2_REQUEST_EVENT_CREATE_JOIN_ROOM` to be notified to the request callback function.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2RequestId assignedReqId;
SceNpMatching2CreateJoinRoomResponse *respData = NULL;

//E Request callback function
void
createJoinRoom_request_cb(
   SceNpMatching2ContextId id,
   SceNpMatching2RequestId reqId,
   SceNpMatching2Event event,
   int errorCode,
   const void *data,
   void *arg
   )
{
   if (id != ctxId || reqId != assignedReqId)
       return;

   if (event != SCE_NP_MATCHING2_REQUEST_EVENT_CREATE_JOIN_ROOM)
       return;

   if (errorCode < 0) {
       //E Error handling when room creation fails
   }

   respData = (SceNpMatching2CreateJoinRoomResponse *)data;
   if (respData == NULL) {
       //E Error handling
   }

   //E Processing for the completion of sceNpMatching2CreateJoinRoom()

   return;
}


SceNpMatching2RequestOptParam optParam;

//E Request option parameters
memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = createJoinRoom_request_cb;

ret = sceNpMatching2CreateJoinRoom(
   ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   //E Error handling
}

//E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_CREATE_JOIN_ROOM to be notified
//  to the request callback function
```

Searching for Rooms

To join a room created by another user, the room ID of the room to join must be obtained. This section describes how to perform a room search, which is 1 way to obtain the room IDs of existing rooms.

Room Search Parameters

To search for rooms, execute `sceNpMatching2SearchRoom()`. Specify search conditions to the request parameters of `sceNpMatching2SearchRoom()`, and filter rooms according to room attribute values. The `SceNpMatching2RoomDataExternal` structure obtained as a search result contains the ID of the room, the configuration of the room, and the attributes of the room. Information on the rooms obtained as results from the search can be displayed on the application screen, for example, for the user to use the displayed information as a way of selecting a room to join.

The items to specify in the request parameters of `sceNpMatching2SearchRoom()` can be largely divided into the following 5 elements.

- Search space
- Filter (search conditions)

- Obtainment scope
- Room search options
- Room attributes to obtain together with the search results

Code Example for the Setting Request Parameters

A code example for setting the request parameters of `sceNpMatching2SearchRoom()` is as follows.

```
//E Request parameters
SceNpMatching2SearchRoomRequest reqParam;

memset(&reqParam, 0, sizeof(reqParam));
```

Specify the Search Space

First, specify the space in which to search for a room. A room search can be executed on a world or lobby, where rooms that match the specified search conditions can be obtained out of all the rooms in the specified world or lobby. Specify either the world ID or lobby ID for the search space. As long as the lobby ID of the request parameters is not 0, it will be assumed that a lobby has been specified, and the world ID will be ignored.

```
//E Assuming that the world ID of the world in which to search is stored:
SceNpMatching2WorldId worldId;

reqParam.worldId = worldId;
reqParam.lobbyId = 0;
```

Specify Filter (Search Conditions)

Specify the search conditions by which to filter rooms belonging to the specified search space. When search conditions are not specified at all, all the rooms belonging to the specified search space will be obtained as the results of the search. Specify room attribute values as search conditions.

Specify flag attribute filters as follows. *`flagFilter`* represents the flag attribute to use as a search condition. A room with *`flagAttr`* bit values matching the applicable *`flagFilter`* bit values will be obtained. For example, when using the value of the FULL flag as a search condition, make `flagFilter = SCE_NP_MATCHING2_ROOM_FLAG_ATTR_FULL`. To obtain rooms without the FULL flag set, specify `flagAttr = 0`. To obtain rooms with the FULL flag set, specify `flagAttr = SCE_NP_MATCHING2_ROOM_FLAG_ATTR_FULL`. When not using the flag attribute filter, specify 0.

```
//E Set flag attribute filter.
//  Get rooms that are not full (without FULL flag set) and accepting
//  members (without CLOSED flag set).
reqParam.flagFilter = (SCE_NP_MATCHING2_ROOM_FLAG_ATTR_CLOSED |
                       SCE_NP_MATCHING2_ROOM_FLAG_ATTR_FULL);
reqParam.flagAttr = 0;
```

Specify the search integer attribute filters. To specify multiple search integer attribute values, specify `SceNpMatching2IntSearchFilter` structures as an array. When not using this filter, specify NULL. When using the external room search integer attribute values of a room created in the section "Creating a Room" as search conditions, specify as follows.

```
#define STAGE_1 (1)
#define STAGE_2 (2)

#define LANG_ENGLISH  (1)
#define LANG_JAPANESE (2)

SceNpMatching2IntSearchFilter intFilter[2];
```

```
memset(intFilter, 0, sizeof(intFilter));

//E Set search integer attribute filters.
//  Use "stages" of the external room search integer attribute ID 1, and
//  "languages" of the external room search integer attribute ID 2 as
//  search conditions.
//  Search for the rooms with both "stages" as STAGE_1
//  and "Languages" as LANG_ENGLISH
intFilter[0].searchOperator = SCE_NP_MATCHING2_OPERATOR_EQ;
intFilter[0].attr.id =
   SCE_NP_MATCHING2_ROOM_SEARCHABLE_INT_ATTR_EXTERNAL_1_ID;
intFilter[0].attr.num = STAGE_1;
intFilter[1].searchOperator = SCE_NP_MATCHING2_OPERATOR_EQ;
intFilter[1].attr.id =
   SCE_NP_MATCHING2_ROOM_SEARCHABLE_INT_ATTR_EXTERNAL_2_ID;
intFilter[1].attr.num = LANG_ENGLISH;

reqParam.intFilter = intFilter;
reqParam.intFilterNum = 2;
```

Specify the search binary attributes. When making multiple specifications, use `SceNpMatching2BinSearchFilter` structures as an array. Specify NULL when not using a search binary attribute filter.

```
SceNpMatching2BinSearchFilter binFilter;
char binData[SCE_NP_MATCHING2_ROOM_SEARCHABLE_BIN_ATTR_EXTERNAL_MAX_SIZE];

memset(&binFilter, 0, sizeof(binFilter));
memset(binData, 0, sizeof(binData));

//E Set search binary attribute filters.
//  Use external room search binary attribute values as search conditions.
strncpy(binData, "BIN_SEARCH_DATA",
   SCE_NP_MATCHING2_ROOM_SEARCHABLE_BIN_ATTR_EXTERNAL_MAX_SIZE);
binFilter.searchOperator = SCE_NP_MATCHING2_OPERATOR_EQ;
binFilter.attr.id = SCE_NP_MATCHING2_ROOM_SEARCHABLE_BIN_ATTR_EXTERNAL_1_ID;
binFilter.attr.ptr = binData;
binFilter.attr.size = strlen(binData);

reqParam.binFilter = &binFilter;
reqParam.binFilterNum = 1;
```

Specify the Obtainment Scope

The maximum number of rooms that can be obtained by 1 room search is 20. The specified search space may contain more than 20 rooms. Thus, it is possible to specify the scope of rooms to obtain from a list of rooms obtained earlier, by specifying search conditions. Specify the starting position from which to obtain rooms from, and the maximum number of rooms to obtain. The list of rooms that can be obtained by a room search are sorted in order, beginning with the room that has the oldest creation date. This means the room occupying the starting position 1 is the oldest. If the number of rooms matching the search conditions is less than the specified maximum number of rooms that can be obtained in 1 room search, all the matching rooms will be obtained.

In an application that lists the information of rooms that have been obtained from the room search, the list can be separated into pages by adding/decreasing the starting position from which to obtain rooms. Refer also to the section "Notes on the Implementation of the Application's Room Selection Screen" in the chapter "Recommended Implementation Items and Precautions".

```
//E Specify the scope by which to obtain rooms.
//  Obtain the maximum number of rooms that can be obtained (20) from the
//  beginning of the list.
```

```
reqParam.rangeFilter.startIndex = 1;
reqParam.rangeFilter.max = SCE_NP_MATCHING2_RANGE_FILTER_MAX;
```

Specify Room Search Options

Specify the room search options, which entail the following.

- Whether or not to obtain room owner information - when specifying to obtain room owner information, make further specification on which information to obtain
- Whether or not to apply the NAT type filter
- Whether or not to make random selections from the search results

When specifying multiple room search options, specify them by logical OR.

Room information (`SceNpMatching2RoomDataExternal` structure) to be obtained from a room search can include information on the room's owner (pointer to the `SceNpId` structure): the room owner's NP ID. To obtain this information, make sure to specify the applicable room search options. If your application does not require this information, disable the information that is not required to reduce network bandwidth and memory usage.

Even when you make settings to establish P2P connection between room members upon join-in, P2P connection may not be established for some combinations of room members' NAT types. When specifying to apply the NAT type filter, the rooms to which members belong with whom you will be unable to establish P2P connections with will be filtered upon executing the search.

When specifying the random search option, you will be able to obtain a list of rooms randomly selected from the rooms matching the specified search conditions. When specifying this option, the starting position of the obtainment scope specified in the request parameters will be ignored, and randomly-selected rooms equaling the maximum number of rooms that can be obtained, will be obtained. Use this option to display information of a specific number of rooms in your application without using pages, or to have the application select a room to join at random from among the rooms matching the search conditions.

```
//E Specify the room search options.
//  Get NP ID for the room owner information.
//  Apply a NAT type filter.
reqParam.option = (SCE_NP_MATCHING2_SEARCH_ROOM_OPTION_WITH_NPID |
                   SCE_NP_MATCHING2_SEARCH_ROOM_OPTION_NAT_TYPE_FILTER);
```

Specify Room Attributes to Obtain with the Search Results

Room information (`SceNpMatching2RoomDataExternal` structure) that can be obtained from a room search corresponds to the external room attributes. The attribute IDs of the desired external room attributes can be specified upon executing a room search to obtain information about them. When specifying multiple external room attributes, specify the attribute IDs (`SceNpMatching2AttributeId`) by an array. When room states or information specific to the application are set as external room attributes, more details can be displayed for the rooms when the application displays the search results. If this specification is not required, specify NULL.

```
SceNpMatching2AttributeId attrId[2];

//E Get external room attributes.
//  Get external room search integer attribute ID 3 and the external room
//  search binary attribute ID 1.
attrId[0] = SCE_NP_MATCHING2_ROOM_SEARCHABLE_INT_ATTR_EXTERNAL_1_ID;
attrId[1] = SCE_NP_MATCHING2_ROOM_BIN_ATTR_EXTERNAL_1_ID;

reqParam.attrId = attrId;
reqParam.attrIdNum = 2;
```

Execute the Request Function

Specify the set request parameters and execute `sceNpMatching2SearchRoom()`. After executing this function, wait for `SCE_NP_MATCHING2_REQUEST_EVENT_SEARCH_ROOM` to be notified to the request callback function.

```
//E Assuming that context ID of the created context is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2RequestId assignedReqId;
SceNpMatching2SearchRoomResponse *respData = NULL;

//E Request callback function
void
searchRoom_request_cb(
    SceNpMatching2ContextId id,
    SceNpMatching2RequestId reqId,
    SceNpMatching2Event event,
    int errorCode,
    const void *data,
    void *arg
    )
{
    if (id != ctxId || reqId != assignedReqId)
        return;

    if (event != SCE_NP_MATCHING2_REQUEST_EVENT_SEARCH_ROOM)
        return;

    if (errorCode < 0) {
        //E Error handling when room search fails
    }

    respData = (SceNpMatching2SearchRoomResponse *)data;
    if (respData == NULL) {
        //E Error handling
    }

    //E Processing for the completion of sceNpMatching2SearchRoom()

    return;
}


SceNpMatching2RequestOptParam optParam;

//E Request option parameters
memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = searchRoom_request_cb;

ret = sceNpMatching2SearchRoom(
    ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
    //E Error handling
}

//E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_SEARCH_ROOM to be notified to the
// request callback function
```

Obtaining Room QoS Information (Ping Information)

In the NP Matching 2 library, the Ping round-trip time (RTT) for the room owner can be obtained as the room's QoS information. To obtain this information, execute `sceNpMatching2SignalingGetPingInfo()`.

Obtain room QoS information for each of the rooms returned from the room search; display them on the application screen or use them to sort room information on the list so that the user can select a room that ensures good communication quality.

After executing `sceNpMatching2SignalingGetPingInfo()`, wait for `SCE_NP_MATCHING2_REQUEST_EVENT_SIGNALING_GET_PING_INFO` to be notified to the request callback function. When the applicable room owner is connected to the internet via a NAT router that does not respond to Ping, a time-out error may occur (`SCE_NP_SIGNALING_ERROR_TIMEOUT`).

Joining a Room

Once the room ID of the desired room has been obtained (from a room search, session information of a joined-in user, or from a game boot message), join the room by executing `sceNpMatching2JoinRoom()`.

Parameters for Joining a Room

The request parameters of `sceNpMatching2JoinRoom()` are comprised of the following elements.

- Parameters required for joining a room
- Initial values for the room member data after joining the room
- Presence option data

The request parameters required for joining a room are as follows.

Request Parameter	Structure Member	Specification
Room ID	*`roomId`*	Required
Room password	*`roomPassword`*	Optional
Group label required to join a room group	*`joinRoomGroupLabel`*	Required for a group room

The request parameters to specify upon setting the initial values for room member data after joining a room are as follows.

Request Parameter	Structure Member	Specification	Change after Joining
Internal room member binary attributes	*`roomMemberBinAttrInternal`*	Optional	Possible
Team ID	*`teamId`*	Optional	Possible

Code Example for Setting Request Parameters

A code example for setting the request parameters of `sceNpMatching2JoinRoom()` is as follows.

```
//E Request parameters
SceNpMatching2JoinRoomRequest reqParam;

memset(&reqParam, 0, sizeof(reqParam));
```

Specify the Room ID of the Room to Join

Specify the room ID of the room to join.

```
//E Assuming that the room ID of the room to join is stored:
SceNpMatching2RoomId roomId;

reqParam.roomId = roomId;
```

Specify the Room Password for Joining a Room

If a room password is set to the room to join, and this password is enabled for the target slot or room group, specify the room password upon joining. Specify NULL when not presenting a room password.

```
//E Specify the room password set upon creating a room in the "Creating a Room"
//  section
SceNpMatching2SessionPassword password;

memset(&password, 0, sizeof(password));
strncpy(password.data, "ROOMPSWD", SCE_NP_MATCHING2_SESSION_PASSWORD_SIZE);

reqParam.roomPassword = &password;
```

Specify the Group Label for Joining a Room

If the room to join is a group room, a group label must be specified. Specify NULL if the room to join is not a group room.

```
//E Specify the label required to join the room group with group ID 1 as its
// ID, created in the "Creating a Room" section
SceNpMatching2GroupLabel groupLabel;

memset(&groupLabel, 0, sizeof(groupLabel));
strncpy(groupLabel.data, "LABEL01",
        SCE_NP_MATCHING2_GROUP_LABEL_SIZE);

reqParam.joinRoomGroupLabel = &groupLabel;
```

Specify Initial Values for the Room Member Data

As necessary, set initial values for the room member data upon joining the room. Use the internal room member binary attributes, for example, to share information exclusive to room members among the room members.

```
SceNpMatching2BinAttr memberBinAttrInternal;
char memberBinData[SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_MAX_SIZE];

memset(&memberBinAttrInternal, 0, sizeof(memberBinAttrInternal));
memset(memberBinData, 0, sizeof(memberBinData));

//E Set common information exclusive to room members in the internal room
//  member binary attribute ID 1
strncpy(memberBinData, "AWAY",
   SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_MAX_SIZE);
memberBinAttrInternal.id =
   SCE_NP_MATCHING2_ROOMMEMBER_BIN_ATTR_INTERNAL_1_ID;
memberBinAttrInternal.ptr = memberBinData;
memberBinAttrInternal.size = strlen(memberBinData);

reqParam.roomMemberBinAttrInternal = &memberBinAttrInternal;
reqParam.roomMemberBinAttrInternalNum = 1;
reqParam.teamId = 0;
```

Specify Presence Option Data

As necessary, specify the presence option data. When a room is successfully joined, an event notifying the joining of a new member will be notified to the members of that room (`SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_JOINED`). The event data (`SceNpMatching2RoomMemberUpdateInfo` structure) accompanying this event will contain presence option data specified upon joining the room.

```
//E Set presence option data.
strncpy(reqParam.optData.data, "OPTDATA",
    SCE_NP_MATCHING2_PRESENCE_OPTION_DATA_SIZE);
reqParam.optData.len = strlen("OPTDATA");
```

Execute the Request Function

Specify the set request parameters and execute `sceNpMatching2JoinRoom()`. After executing this function, wait for `SCE_NP_MATCHING2_REQUEST_EVENT_JOIN_ROOM` to be notified to the request callback function.

```
//E Assuming that the context ID of the created context is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2RequestId assignedReqId;
SceNpMatching2JoinRoomResponse *respData = NULL;

//E Request callback function
void
joinRoom_request_cb(
    SceNpMatching2ContextId id,
    SceNpMatching2RequestId reqId,
    SceNpMatching2Event event,
    int errorCode,
    const void *data,
    void *arg
    )
{
    if (id != ctxId || reqId != assignedReqId)
        return;

    if (event != SCE_NP_MATCHING2_REQUEST_EVENT_JOIN_ROOM)
        return;

    if (errorCode < 0) {
        //E Error handling when room joining fails
    }

    respData = (SceNpMatching2JoinRoomResponse *)data;
    if (respData == NULL) {
        //E Error handling
    }

    //E Processing for the completion of sceNpMatching2JoinRoom()

    return;
}


SceNpMatching2RequestOptParam optParam;

//E Request option parameters
memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = joinRoom_request_cb;
```

```
    ret = sceNpMatching2JoinRoom(
        ctxId, &reqParam, &optParam, &assignedReqId);
    if (ret < 0) {
        //E Error handling
    }

    //E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_JOIN_ROOM to be notified to
    //  the request callback function
```

Leaving a Room

To leave a room, execute `sceNpMatching2LeaveRoom()`. For the request parameters of `sceNpMatching2LeaveRoom()`, specify the ID of the room to leave and the presence option parameter.

When presence option data is specified, the specified presence option data will be included in the event data (`SceNpMatching2RoomMemberUpdateInfo` structure) that will accompany the `SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT` event for notifying the other room members of the user leaving the room. For example, this can be used to know whether the user left by his/her own initiative, or whether `sceNpMatching2LeaveRoom()` was executed, irrespective of a user operation, because of a condition specific to the application.

After executing `sceNpMatching2LeaveRoom()`, wait for `SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_ROOM` to be notified to the request callback function. There will be no event data accompanying this event.

Processing by `sceNpMatching2LeaveRoom()` will always succeed; thus, error handling is not necessary for the notification of `SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_ROOM`. Refer also to the "Error Handling upon Leaving a Session" section in the "Recommended Implementation Items and Precautions" chapter.

```
    //E Assuming that the context ID of the created context is stored:
    SceNpMatching2ContextId ctxId;
    SceNpMatching2RequestId assignedReqId;

    //E Request callback function
    void
    leaveRoom_request_cb(
        SceNpMatching2ContextId id,
        SceNpMatching2RequestId reqId,
        SceNpMatching2Event event,
        int errorCode,
        const void *data,
        void *arg
        )
    {
        if (id != ctxId || reqId != assignedReqId)
            return;

        if (event != SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_ROOM)
            return;

        //E sceNpMatching2LeaveRoom() completed.

        //E errorCode is always 0.
        //E No event data

        return;
    }
```

```
//E Assuming that the room ID of the room to leave is stored:
SceNpMatching2RoomId roomId;
SceNpMatching2LeaveRoomRequest reqParam;
SceNpMatching2RequestOptParam optParam;

//E Request parameters
memset(&reqParam, 0, sizeof(reqParam));
reqParam.roomId = roomId;
strncpy(reqParam.optData.data, "USER ACTION",
   SCE_NP_MATCHING2_PRESENCE_OPTION_DATA_SIZE);
reqParam.optData.len = strlen("USER_ACTION");

//E Request option parameters
memset(&optParam, 0, sizeof(optParam));
optParam.cbFunc = leaveRoom_request_cb;

ret = sceNpMatching2LeaveRoom(
   ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   //E Error handling
}

//E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_LEAVE_ROOM to be notified
//  to the request callback function.

//E Processing to leave room completed.
```

Processing for a Room Member Joining/Leaving a Room

When a user executes sceNpMatching2JoinRoom() and joins a room, the SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_JOINED event for notifying that a new member joined the room will be notified to the room event callback functions of the members already joined into that room. In addition, when a room member leaves the room by executing sceNpMatching2LeaveRoom() or because of a network error occurrence, the other room members will be notified of this by receiving the SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT event in their room event callback functions.

When the room owner leaves the room to which the automatic transfer of room ownership has not been set, SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED will be notified to the remaining room members instead of SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT. Refer to the section "Processing upon Room Destruction or Kick-out".

The event data accompanying SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_JOINED and SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT is the SceNpMatching2RoomMemberUpdateInfo structure. The SceNpMatching2RoomMemberUpdateInfo structure contains the following information.

- Internal room member data of the room member who joined/left the room
- Event cause (SCE_NP_MATCHING2_EVENT_CAUSE_*)
- Presence option data

The SceNpMatching2RoomMemberUpdateInfo structure only contains the internal room member data of the room member who joined/left the room. If the room member list to be managed by the application is implemented, perform operation on the room list to add a new member or delete the member who left.

A value for the event cause will only be stored when the SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT event is notified. Use it to acknowledge the reason why the member left the room on the application-side. The main event causes that can be contained in the SceNpMatching2RoomMemberUpdateInfo structure and their corresponding meanings are as follows (make sure to program your application so that it does not malfunction even when other event causes are notified).

Event Cause	Meaning
SCE_NP_MATCHING2_EVENT_CAUSE_LEAVE_ACTION	The room member explicitly executed `sceNpMatching2LeaveRoom()` and left room
SCE_NP_MATCHING2_EVENT_CAUSE_KICKOUT_ACTION	The room member was kicked out by the room owner
SCE_NP_MATCHING2_EVENT_CAUSE_MEMBER_DISAPPEARED	The room member left because of a network error occurrence or for another reason

Presence option data corresponds to the presence option data specified as a request parameter upon the user explicitly executing `sceNpMatching2LeaveRoom()` and leaving the room.

Processing to Transfer Room Ownership

When the room ownership is transferred, the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_OWNER_CHANGED` event is notified to the room event callback function. Factors that may cause the room ownership to be transferred are as follows.

- The room owner explicitly executed `sceNpMatching2GrantRoomOwner()`
- The room owner left a room to which automatic transfer of room ownership has been set

The event data accompanying the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_OWNER_CHANGED` event is the `SceNpMatching2RoomOwnerUpdateInfo` structure. This structure contains the following information.

- Room member IDs of the old room owner and the new room owner
- Event cause (`SCE_NP_MATCHING2_EVENT_CAUSE_*`)
- Room password
- Presence option data

The room member IDs of the old room owner and new room owner will help you identify between which room members the room ownership has been transferred. If the application is managing room member information, make sure to reflect this change onto the application-side.

The event cause is used to acknowledge the reason for the room ownership transfer, on the application-side. The main event causes that can be contained in the `SceNpMatching2RoomOwnerUpdateInfo` structure upon a room ownership transfer, and their corresponding meanings, are as follows (make sure to program your application so that it does not malfunction even when other event causes are notified).

Event Cause	Meaning
SCE_NP_MATCHING2_EVENT_CAUSE_LEAVE_ACTION	The room ownership was automatically transferred because the room owner explicitly executed `sceNpMatching2LeaveRoom()` and left the room to which automatic transfer of room ownership has been set
SCE_NP_MATCHING2_EVENT_CAUSE_GRANT_OWNER_ACTION	The room ownership was transferred because the room owner explicitly executed `sceNpMatching2GrantRoomOwner()`
SCE_NP_MATCHING2_EVENT_CAUSE_MEMBER_DISAPPEARED	The room ownership was automatically transferred because the room owner left the room, to which automatic transfer of room ownership has been set, because of a network error occurrence, or for some other reason

When the room ownership of a room - with a room password set to it - is transferred, the room's password will be included in the `SceNpMatching2RoomOwnerUpdateInfo` structure received by the new room owner.

Presence option data corresponds to the presence option data specified by the old room owner as a request parameter upon executing `sceNpMatching2GrantRoomOwner()` or `sceNpMatching2LeaveRoom()`.

Processing upon Room Destruction or Kick-out

When a room is destroyed, the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` event is notified to the room event callback function. If the room owner executed `sceNpMatching2KickoutRoomMember()` and kicked out the user from the room, `SCE_NP_MATCHING2_ROOM_EVENT_KICKEDOUT` will be notified to the room event callback function.

The factors that may cause the room to be destroyed and the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` event to be notified are as follows.

- The room owner left the room to which automatic transfer of room ownership has not been set (either the room owner explicitly executed `sceNpMatching2LeaveRoom()` or a network error or some other reason caused him/her to leave the room)
- Connection to the room was disconnected because of an NP sign-out or a network error (the member whose connection has been disconnected will be notified with the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` event. `SCE_NP_MATCHING2_ROOM_EVENT_MEMBER_LEFT` will be notified to the other members)

The `SceNpMatching2RoomUpdateInfo` structure representing the event data accompanies `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` and `SCE_NP_MATCHING2_ROOM_EVENT_KICKEDOUT`. This structure contains the following information.

- Event cause (`SCE_NP_MATCHING2_EVENT_CAUSE_*`)
- Error code
- Presence option data

The event cause is used to acknowledge the reason for the room being destroyed or for being kicked-out, on the application-side.

The main event causes that can be contained in the `SceNpMatching2RoomUpdateInfo` structure upon a room being destroyed, and their corresponding meanings, are as follows (make sure to program your application so that it does not malfunction even when other event causes are notified).

Event Cause	Meaning
`SCE_NP_MATCHING2_EVENT_CAUSE_LEAVE_ACTION`	The room explicitly executed `sceNpMatching2LeaveRoom()` and left room
`SCE_NP_MATCHING2_EVENT_CAUSE_MEMBER_DISAPPEARED`	The room owner left the room because of a network error or for some other reason
`SCE_NP_MATCHING2_EVENT_CAUSE_CONNECTION_ERROR`	Connection with the room was disconnected by a network error
`SCE_NP_MATCHING2_EVENT_CAUSE_NP_SIGNED_OUT`	Signed-out from the NP

The main event cause that can be contained in the `SceNpMatching2RoomUpdateInfo` structure upon being kicked out from a room, and its corresponding meaning, are as follows (make sure to program your application so that it does not malfunction even when other event causes are notified).

Event Cause	Meaning
`SCE_NP_MATCHING2_EVENT_CAUSE_KICKOUT_ACTION`	Kicked out because the room owner executed `sceNpMatching2KickoutRoomMember()`

An error code value will only be stored when the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` event is notified.

Presence option data corresponds to the presence option data specified by the room owner as a request parameter upon executing `sceNpMatching2LeaveRoom()` or `sceNpMatching2KickoutRoomMember()`.

When the `SCE_NP_MATCHING2_ROOM_EVENT_ROOM_DESTROYED` or `SCE_NP_MATCHING2_ROOM_EVENT_KICKEDOUT` event is notified, the management information for the joined-in room in the NP Matching 2 library will be completely cleared. Thus, there is no need for the application to perform processing to leave the room by executing `sceNpMatching2LeaveRoom()`. However, make sure to also clear any room information or room member information on the application-side.

Establishing P2P Connection

P2P connections are established between room members either by specifying the signaling option parameter for the request parameter of `sceNpMatching2CreateJoinRoom()` or by setting the signaling option parameter in `sceNpMatching2SetSignalingOptParam()` after the room is created.

For example, to establish P2P connections in a star topology with the room owner as the hub, set the signaling option parameter as follows.

```
SceNpMatching2SignalingOptParam sigOptParam;

memset(&sigOptParam, 0, sizeof(sigOptParam));
sigOptParam.type = SCE_NP_MATCHING2_SIGNALING_TYPE_STAR;
sigOptParam.hubMemberId = 0; //E 0 indicates that the room owner is the hub
```

Processing for establishing P2P connections is started as soon as the signaling option parameter is set. When P2P connections are established or disconnected, the event is reported to the signaling callback function. The signaling callback function is registered by executing `sceNpMatching2RegisterSignalingCallback()`.

The events for establishing and disconnecting P2P connections are as follows.

- Establishing P2P connections: `SCE_NP_MATCHING2_SIGNALING_EVENT_ESTABLISHED`
- Disconnecting P2P connections: `SCE_NP_MATCHING2_SIGNALING_EVENT_DEAD`

When a room member leaves the room or when the room is destroyed, the P2P disconnection event will be notified for each room member with whom a P2P connection was established.

When P2P connections are established in a star topology and the room member who was set as the hub leaves the room, the operation of the NP Matching 2 library will be as follows.

- When automatic transfer of room ownership is set and the room owner is set as the hub:
  P2P connection with the room owner who left will be disconnected, and processing to establish P2P connection will automatically be started with the new room owner, to whom room ownership has been transferred, as the hub
- When automatic transfer of room ownership is not set, or when the room owner is not the hub:
  P2P connection with the room member who was the hub and left will be disconnected. If it is necessary to re-establish P2P connection, the application must reset the signaling option parameter.

For details concerning each function or definition, refer to the "NP Matching 2 Library Reference" document.

Inviting a User to Join a Session

To join a room created by another user, the room ID of that room is required. This section describes the use of an invitation to join a session; invitations can be used as a way to obtain the room IDs of existing rooms.

When you want to invite a user to a session, pass the data (invitation data) necessary for the user to join the session. Invitation data may include the following.

- Lobby ID (`SceNpMatching2LobbyId`)
- Room ID (`SceNpMatching2RoomId`)
- Session password (`SceNpMatching2SessionPassword`)
- Group label (`SceNpMatching2GroupLabel`)

If there is additional data required by the application, pass it together with the above data. The application can select the data to be included in the invitation data, as necessary, in the NP Matching 2 library.

For example, when inviting a user to a room for which a room password has been set, comprise the invitation data as follows.

```
struct invitation_info {
    SceNpMatching2RoomId roomId;
    bool withPassword;
    SceNpMatching2SessionPassword roomPassword;
};
```

To pass the invitation data to the target user, the game boot message feature can be used. Regarding the game boot message, refer to the "NP Message Overview" document..

A user who receives an invitation data sets the data contained in the invitation data as the request parameters of `sceNpMatching2JoinRoom()` and joins the room.

Stopping and Destroying the Context

Upon termination of the application, execute processing to stop and destroy the context. When performing to stop the context, wait for `SCE_NP_MATCHING2_CONTEXT_EVENT_STOPPED` to be notified.

```
//E Context ID of the context no longer to be used
SceNpMatching2ContextId ctxId;

//E Stop the context
ret = sceNpMatching2ContextStop(ctxId);
if (ret < 0) {
    //E ERROR HANDLING
}

//E Wait for SCE_NP_MATCHING2_CONTEXT_EVENT_STOPPED to be notified
//  to the context callback function.

//E Destroy the context
ret = sceNpMatching2DestroyContext(ctxId);
if (ret < 0) {
    //E ERROR HANDLING
}
```

Terminating the NP Matching 2 Library

When you no longer need to use the NP Matching 2 library, such as, upon termination of the application, execute processing for terminating the NP Matching 2 library.

```
//E Terminate the NP Matching 2 library
sceNpMatching2Term();
```

Unloading PRX and Terminating the NP Library

Unload the NP Matching 2 PRX.

When you no longer need to use the NP library, such as, upon termination of the application, terminate the NP library.

```
//E Unload the NP Matching 2 PRX
ret = sceSysmoduleUnloadModule(SCE_SYSMODULE_NP_MATCHING2);
if (ret < 0) {
   //E ERROR HANDLING
}
//E Terminate the NP library
sceNpTerm();
```

5 Use Cases and Implementation

This chapter describes use cases of applications that use the NP Matching 2 library and how to implement them.

Displaying the World, and Lobby Names

When displaying world or lobby names for the user to make a selection, the world numbers, and lobby numbers must be statically bound to the names on the application.

To obtain world numbers and lobby numbers, use the `SCE_NP_MATCHING2_GET_WORLD_NUMBER()` and `SCE_NP_MATCHING2_GET_LOBBY_NUMBER()` macros for obtaining them from the world ID or lobby ID, respectively.

```
//E World numbers
#define WORLD_NUMBER_1 (1)
#define WORLD_NUMBER_2 (2)

//E Return names bound to the world numbers
char *
get_world_name(
    SceNpMatching2WorldId worldId
    )
{
    switch(SCE_NP_MATCHING2_GET_WORLD_NUMBER(worldId)) {
    case WORLD_NUMBER_1:
        return("World Name 1");
    case WORLD_NUMBER_2:
        return ("World Name 2");
    default:
        return ("Unknown");
    }

    return ("Unknown");
}
```

Reserve Some Room Slots

To reserve some room slots, create a room without setting room groups. Set a room password, and a room password slot mask for representing the slots to be reserved.

The following is an example where a room with 8 slots is created, of which slots 5 to 8 are reserved.

```
SceNpMatching2WorldId worldId;
SceNpMatching2SessionPassword sessionPassword;
SceNpMatching2RoomPasswordSlotMask roomPasswordSlotMask;

SceNpMatching2ContextId ctxId;
SceNpMatching2CreateJoinRoomRequest reqParam;
SceNpMatching2RequestOptParam optParam;
SceNpMatching2RequestId assignedReqId;

//E Total slots
int maxSlot = 8;

//E Room password
memset(&sessionPassword, 0, sizeof(sessionPassword));
memcpy(sessionPassword.data, "PASSWORD" strlen("PASSWORD"));
```

```
//E Room password slot mask
memset(&roomPasswordSlotMask, 0, sizeof(roomPasswordSlotMask));
for (int i = 0; i < 4; i++) {
        SCE_NP_MATCHING2_ADD_SLOTNUM_TO_ROOM_PASSWORD_SLOT_MASK(
        roomPasswordSlotMask, (maxSlot - i));
}

//E Request parameters
memset(&reqParam, 0, sizeof(reqParam));
reqParam.worldId = worldId; //E Specify world to which the created room will belong
reqParam.lobbyId = 0;  //E Specify 0 if the room will not belong to a lobby
//E Parameters to create a room
reqParam.maxSlot = maxSlot;
reqParam.groupConfig = NULL;
reqParam.groupConfigNum = 0;
reqParam.roomPassword = &sessionPassword;
reqParam.passwordSlotMask = &roomPasswordSlotMask;
reqParam.flagAttr = 0;
reqParam.roomSearchableIntAttrExternal = NULL;
reqParam.roomSearchableIntAttrExternalNum = 0;
reqParam.roomBinAttrInternal = NULL;
reqParam.roomBinAttrInternalNum = 0;
reqParam.roomSearchableBinAttrExternal = NULL;
reqParam.roomSearchableBinAttrExternalNum = 0;
reqParam.roomBinAttrExternal = NULL;
reqParam.roomBinAttrExternalNum = 0;
reqParam.allowedUser = NULL;
reqParam.allowedUserNum = 0;
reqParam.blockedUser = NULL;
reqParam.blockedUserNum = 0;
//E Parameters to join the room
reqParam.roomMemberBinAttrInternal = NULL;
reqParam.roomMemberBinAttrInternalNum = 0;
reqParam.teamId = 0;
reqParam.joinRoomGroupLabel = NULL;
//E Signaling parameter
reqParam.sigOptParam = NULL;

//E Assuming that appropriate values are stored in ctxId and optParam:

ret = sceNpMatching2CreateJoinRoom(
   ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   /* Error handling */
}
```

Preventing Users Registered on the Blocked User List from Joining a Room

A setting can be made in the NP Matching 2 system to block certain users from joining a room. Note that this is a separate feature from the blocked user list of the system software. However, it is possible to obtain users registered on the blocked user list of the system software and to specify them upon creating a room to prevent them from joining the room.

Obtain the NP IDs of users registered on the blocked user list using a PSN℠ Web API and specify to block those users from joining a room upon creating the room. For the procedure to obtain the NP IDs of users registered on the blocked user list using a PSN℠ Web API, refer to the "User Profile Web APIs Reference (Profile, FriendList, Presence, BlockList)" document.

```
SceSize count;
```

```
SceNpId blockedUser[SCE_NP_MATCHING2_ROOM_BLOCKED_USER_MAX];
SceNpMatching2CreateJoinRoomRequest reqParam;

memset(blockedUser, 0, sizeof(blockedUser));

//E Obtain blocked user list with a PSN(SM) Web API (omitted)

//E Set as the request parameter for creating the room
reqParam.blockedUser = blockedUser;
reqParam.blockedUserNum = count;
//E Other request parameters omitted
```

Joining a Session Joined by a Specific User

A session into which a specific user, such as, a registered friend, is joined can be searched for, in order to display information on the user and/or to join that session.

To obtain information on the session joined in by a specific user or to obtain information on that user, execute `sceNpMatching2GetUserInfoList()`. Because the target user's NP ID must be specified as this function's request parameter, obtain the user's NP ID using the friend obtainment function of the NP Basic library or by some other application-specific method.

If the target user is joined into more than 1 session, the session IDs of the sessions into which the user is joined will be included in the `SceNpMatching2JoinedSessionInfo` structure obtained as part of the user's information. Once you obtain the session IDs, it will be possible to join any one of these sessions. To display session information before joining that session, specify its session ID and execute `sceNpMatching2GetRoomDataExternalList()`.

```
//E Assuming that an appropriate value is stored:
SceNpMatching2ContextId ctxId;
SceNpMatching2ServerId serverId;
//E Assuming that the NP ID of the target user for obtaining user information
//  is stored:
SceNpId npId;

int ret = 0;
SceNpMatching2AttributeId attrId;
SceNpMatching2GetUserInfoListRequest reqParam;
SceNpMatching2GetUserInfoListResponse *respData;
SceNpMatching2RequestOptParam optParam;
SceNpMatching2RequestId assignedReqId;

//E Set request parameters
attrId = SCE_NP_MATCHING2_USER_BIN_ATTR_1_ID;
memset(&reqParam, 0, sizeof(reqParam));
reqParam.serverId = serverId;
reqParam.npId = &npId;
reqParam.npIdNum = 1;
reqParam.attrId = &attrId;
reqParam.attrIdNum = 1;

//E Assuming that appropriate values are stored for the request option
//  parameters:

ret = sceNpMatching2GetUserInfoList(
       ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   //E ERROR HANDLING
}

void
```

```
request_cb(
   SceNpMatching2ContextId id,
   SceNpMatching2RequestId reqId,
   SceNpMatching2Event event,
   int errorCode,
   const void *data,
   void *arg
   )
{
   //E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_GET_USER_INFO_LIST to be
   //  notified to the request callback function

   //E Assuming that processing to obtain user information completed
   //  and that respData is pointing to the event data:

   if (respData->userInfo == NULL) {
      //E Target user did not exist on this server.
      return;
   }

   if (respData->userInfo[0].joinedSessionInfo != NULL) {
      //E The user is currently joined into
      //  respData->userInfo[0].joinedSessionInfoNum number of sessions
      for (int j = 0; j < respData->userInfo[0].joinedSessionInfoNum; j++) {
         if (respData->userInfo[0].joinedSessionInfo[j].sessionType
            == SCE_NP_MATCHING2_SESSION_TYPE_LOBBY) {
            //E The user is joined into the lobby indicated by
            //  respData->userInfo[0].joinedSessionInfo[j].lobbyId
         }
         else if (respData->userInfo[0].joinedSessionInfo[j].sessionType
            == SCE_NP_MATCHING2_SESSION_TYPE_ROOM) {
            //E The user is joined into the room indicated by
            //  respData->userInfo[0].joinedSessionInfo[j].roomId
         }
      }
   }

   if (respData->userInfo[0].userBinAttr != NULL) {
      //E Obtained user attributes.
      //  If application-specific information (state of the user in the
      //  session, for example) is set as a user attribute, the obtained user
      //  information can be displayed here.
   }
}
```

Specifying the Priority of Room Members to whom Room Ownership Is to Be Automatically Transferred

When room ownership is to be automatically transferred, the target room member is randomly determined by default. However, it is also possible to prioritize a certain room member depending on the state of communication and conditions specific to the application, so that he/she will be appointed the next room owner.

To set a priority order for the automatic transfer of room ownership over the room members of a room, execute `sceNpMatching2SetRoomDataInternal()` after joining that room. Specify the room member IDs of room members to be prioritized from highest to lowest as an array, to the *`ownerPrivilegeRank`* request parameter of this function.

The setting of priorities by the `sceNpMatching2SetRoomDataInternal()` function will be completely overwritten by the setting indicated by the *`ownerPrivilegeRank`* request parameter. When the room members change and you want to change the priority order, re-execute `sceNpMatching2SetRoomDataInternal()`.

```
//E Number of room members to set the priority order over for the automatic
//  transfer of room ownership
#define CANDIDATE_ROOM_MEMBER_NUM (16)

//E Assuming that an appropriate value is stored:
SceNpMatching2ContextId ctxId;
//E Set priority order for room members to be transferred room ownership.
//  Assuming that the room ID of the room is stored:
SceNpMatching2RoomId roomId;

int ret = 0;
SceNpMatching2SetRoomDataInternalRequest reqParam;
SceNpMatching2RequestOptParam optParam;
SceNpMatching2RequestId assignedReqId;

//E Assuming that the room member IDs of room members to be prioritized
//  are stored as an array with the room member having the highest priority
//  taking the smallest index:
SceNpMatching2RoomMemberId memberId[CANDIDATE_ROOM_MEMBER_NUM];

//E Request parameters
memset(&reqParam, 0, sizeof(reqParam));
reqParam.roomId = roomId;
reqParam.ownerPrivilegeRank = memberId;
reqParam.ownerPrivilegeRankNum = CANDIDATE_ROOM_MEMBER_NUM;

//E Assuming that appropriate values are stored in the request option parameters:

ret = sceNpMatching2SetRoomDataInternal(
   ctxId, &reqParam, &optParam, &assignedReqId);
if (ret < 0) {
   //E ERROR HANDLING
}

//E Wait for SCE_NP_MATCHING2_REQUEST_EVENT_SET_ROOM_DATA_INTERNAL to be
//  notified to the request callback function
```

Performing Team-vs.-Team Play that Can Only Be Joined by Specific Team Members

Team-vs.-team play can be realized between teams - teams/user groups created specifically for the application - that are maintained over an extended period of time, irrespective of the continuation of rooms. Note that this is different from assigning a room member - who has just joined a play room - to a team for the time being to perform team-vs.-team play.

Certain restrictions may be applied, such as, restricting the number of members that each team can have in order to prevent one team from having too many members, and restricting members so that only users who belong to the teams playing against each other can join the room.

In this example, a room with the following conditions is created to realize team-vs.-team play.

- 2 teams will play against each other
- Each team can have up to 8 members

To satisfy these conditions, create a group room with 2 room groups, and a total of 16 slots.

Set teams, as follows.

- One team will have the room creator in it
- The opponent team will not have specific specifications set to it. The opponent team will be set when a room member, who belongs to a team different from that of the room creator, joins the room.
- Subsequently, only the users belonging to either of these 2 teams will be able to join this room.

To satisfy the condition that only members belonging to either of the 2 teams can join the room, use the team ID as the group label of the respective room groups. For the group label of the room group to which the room creator belongs, create the group room with the team ID of the room creator set as its group label. For the group label of the opponent room group, leave it "unset" upon creating the room. The user joining this group room must present the team ID of the team to which he/she belongs as the group label. Thus, the room member belonging to a different team from the room creator will join the other room group and have his/her team ID set as the group label of the opponent team. Subsequently, users belonging to a team other than the 2 that have been set will be unable to join this room.

In the above method, any user belonging to either of the 2 teams is able to join the room. To create a room into which only certain users belonging to the either of the 2 set teams can join, set a room password in addition to the steps already described above. A user will then be required to present a room password, in addition to the team ID that is used as the group label, upon entering the room.

```
//E Assuming that the world ID of the world in which to create a room is
//  stored:
SceNpMatching2WorldId worldId;
SceNpMatching2TeamId myTeamId;

SceNpMatching2RoomGroupConfig groupConfig[2];
SceNpMatching2GroupLabel label;
SceNpMatching2CreateJoinRoomRequest reqParam;

//E Use the team ID of the team to which the room creator belongs, as the group
//  label of the room group to join
memset(label, 0, sizeof(label));
memcpy(label.data, &s_labels[myTeamId], LABEL_SIZE);

//E Set room groups
memset(groupConfig, 0, sizeof(groupConfig));

//E Set room group to be joined by the room creator
groupConfig[0].slotNum = 8;
groupConfig[0].withLabel = true;  //E Only members belonging to the same team
                                  //  as the room creator can join
memcpy(groupConfig[0].label, &label, sizeof(label));
groupConfig[0].withPassword = false;  //E Any member from the same team
                                      //  can join

//E Set the other room group
groupConfig[1].slotNum = 8;
groupConfig[1].withLabel = false; //E First member can be from any team
groupConfig[1].withPassword = false; //E Any member from the same team can join

//E Request parameters
memset(&reqParam, 0, sizeof(reqParam));
reqParam.worldId = worldId;
reqParam.lobbyId = 0;
reqParam.maxSlot = 16;
reqParam.roomPassword = NULL; //E Any member from the same team can join
reqParam.groupConfig = groupConfig;
reqParam.groupConfig = 2;
reqParam.joinRoomGroupLabel = &label;
//E Specification of other request parameters omitted here
```

6 Items That Must Be Implemented

This chapter explains items that must be implemented by all applications that use the NP Matching 2 library.

Use in NP Signed-in State

The NP Matching 2 library must be used in an NP signed-in state. Make sure you start Network Check Dialog to enter the NP signed-in state before transitioning to the state for using the NP Matching 2 library.

For details concerning Network Check Dialog, refer to the "Network Overview" and "libnetctl Reference" documents.

When Terminating a Context

Make sure you wait for the notification of `SCE_NP_MATCHING2_CONTEXT_EVENT_STOPPED` when terminating a context.

7 Recommended Implementation Items and Precautions

This chapter explains items that are recommended when implementing an application that uses the NP Matching 2 library and related precautions.

Timeout Time upon Leaving a Session

It may take some time for the completion of a request by `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()` for leaving a lobby or room to be reported to the request callback function due to the effect of network delay or other factors.

Although the timeout interval can be specified for each request by setting the request option parameter when the request function is executed, if the application is adverse to time being taken by processing for leaving a room or lobby and makes the timeout interval of `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()` too short, processing may not be appropriately synchronized with the server that manages the session state, and it may become inconsistent with the actual session state on the server.

Make sure that the application does not make the timeout interval of `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()` too short. Setting the timeout interval to more than 10 seconds, approximately, is recommended.

Error Handling upon Leaving a Session

After `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()` returns 0 (when the function call succeeds), the completion of the request processing will be notified to the request callback function. This notification will always indicate that the request succeeded. Even when a network delay occurs, a wait will occur until the set time-out time passes, the session information in the NP Matching 2 library will then be cleared, and the processing will be notified as having succeeded (this excludes the overflow of the event data queue in the system).

Thus, when the application receives the completion notification of a request from `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()`, there is no need to perform error handling.

When the return value of `sceNpMatching2LeaveLobby()` or `sceNpMatching2LeaveRoom()` is a negative value, the specified argument, request parameter, or execution condition may not be valid - check these settings.

Proper Use of Session Internal and External Attributes

The attributes of NP Matching 2 system components consist of session internal attributes and session external attributes. Session internal attributes can only be accessed from clients that have joined the session. Session external attributes can also be accessed from clients that have not joined the session.

Although session external attributes can also be accessed from clients that have joined the session, since accessing a session external attribute causes a greater server load than accessing a session internal attribute, the NP Matching 2 system can exhibit better performance by using session internal and session external attributes appropriately.

The following precautions should be kept in mind when implementing an application.

- Use session internal attributes when handling data that is to be shared only between session members.
- Use session external attributes when handling data (data used as room search conditions, data that indicates game progress, etc.) that is to be referenced from clients that have not joined the session.

- Be careful not to generate unnecessary transactions for the server. In particular, make sure not to access session external attributes by polling.

Execution Order of Request Functions and of the Server

For the request functions indicated below, the execution of the request function and the order by which that request will be executed on the server may differ. Moreover, the order by which the request is executed on the server and the order by which the processing completion regarding that request is notified to the request callback function may also differ.

- `sceNpMatching2GetWorldInfoList()`
- `sceNpMatching2SetUserInfo()`
- `sceNpMatching2GetUserInfoList()`
- `sceNpMatching2SearchRoom()`
- `sceNpMatching2CreateJoinRoom()`
- `sceNpMatching2JoinRoom()`
- `sceNpMatching2GetRoomDataExternalList()`
- `sceNpMatching2SetRoomDataExternal()`
- `sceNpMatching2GetRoomMemberDataExternalList()`
- `sceNpMatching2GetLobbyInfoList()`
- `sceNpMatching2JoinLobby()`

Besides the above request functions, for 2 or more requests with differing request target sessions, the execution order of the requests and the order by which the respective processing completions is notified cannot be guaranteed.

Note the above point and implement an application that is not dependent on the execution order of request functions. Especially for requests in which values are set, such as `sceNpMatching2SetUserInfo()` and `sceNpMatching2SetRoomDataExternal()`, it is recommended that the application execute the second request after receiving notification on the processing completion of the first request, for example, and prevent issuing multiple requests at the same time.

Notes on the Implementation of the Application's Room Selection Screen

An application in which the user selects a room from a list of rooms obtained from a room search, and displayed onscreen, must be implemented with consideration for the following points.

- The rooms obtained by executing `sceNpMatching2SearchRoom()` is sorted in order of the rooms' creation date, beginning with the oldest room
- The room list and room information obtained by executing `sceNpMatching2SearchRoom()` are valid at the timing when `sceNpMatching2SearchRoom()` was executed

If a scheme is adopted where only a certain number of rooms are displayed from the beginning of the list of rooms obtained from a room search, many users will fail in their attempt to join a room for an application with many online users, and of a large scale. In such an application, all the rooms listed onscreen will most often be full and users will be unable to join a room unless these rooms are destroyed or their current room members leave.

To avoid this problem, take one of the appropriate measures, as follows.

- Specify `SCE_NP_MATCHING2_SEARCH_ROOM_OPTION_RANDOM` to `sceNpMatching2SearchRoom()` and obtain rooms that are randomly-selected from among those that match the specified search conditions. By obtaining and displaying randomly-selected rooms, the situation of having all the users attempting to join the same few rooms on display can be avoided.

- Implement paging for displaying the list of rooms obtained from a room search. The starting position of rooms to obtain can be specified in `sceNpMatching2SearchRoom()`. Vary this starting position and separate the list of rooms to display it in page installments. Because user operation allows all the currently existing rooms to be obtained, this will ease the concentration of users attempting to join the same rooms, and provide a means to find more rooms that can be joined.

Limiting the Frequency of Requests

Request functions (described in the "NP Matching 2 Library Reference" document) are functions that send requests to the server on the NP Matching 2 system. If the game application keeps calling these functions automatically, this will result in a significant load on the server. Therefore, request functions must not be called periodically, and only when triggered by user operation.

The following are examples of problematic implementation.

- The game application executes searches periodically.
- The game application obtains or modifies external room data periodically.

Do not employ programs that poll the server on the NP Matching 2 system.

Joining Multiple Rooms and/or Lobbies

The NP Matching 2 library supports the joining of multiple rooms and/or lobbies. The number of rooms and/or lobbies that can be joined at the same time is dependent on the memory pool size.

Moreover, even when there is enough memory, the maximum number of rooms/lobbies that can be joined is ten. For example, `SCE_NP_MATCHING2_ERROR_JOINED_SESSION_MAX` will return when attempting to join a room/lobby when already joined into five rooms and five lobbies.