Ad Hoc Matching Library Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

1 Library Overview 3
Functional Overview and Features 3
Used Resources 4
Files 4
Sample Programs 4
Reference Materials 4
2 Using the Library 5
Initialization 5
Matching Process 5
Termination 7
Event Handler Function Sample 7
3 Explanation of Operation 11
Group Models 11
Matching Protocol 12
Event Model 13
Optional Data 15
Data Transmission and Reception 16
4 Precautions 17
Specifying the Pool Size When Initializing the Library 17
Conditions for Sending Hello Messages 17
When the Number of Connected Players Reaches the Maximum 18
SCE_NET_CTL_ERROR_NOT_CONNECTED 18

1 Library Overview

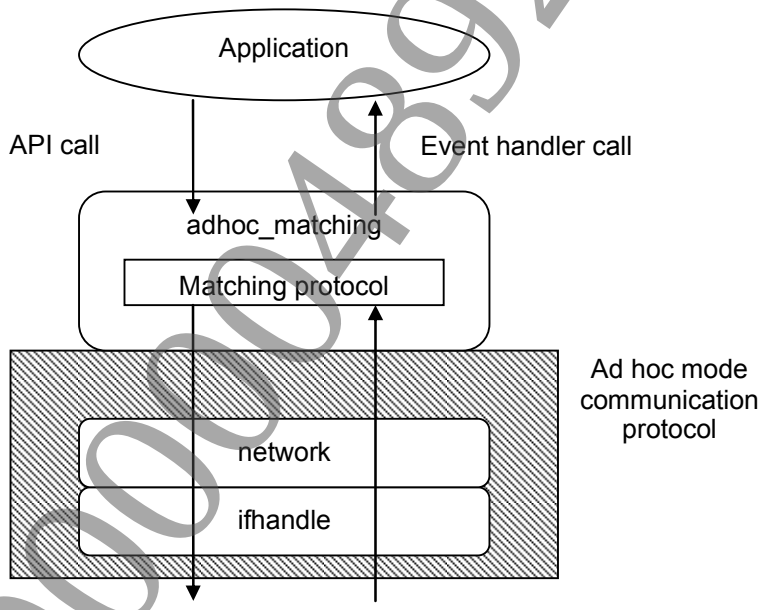
Functional Overview and Features

The Ad hoc Matching library (adhoc_matching) provides services for configuring (matching) a group of participating game players in ad hoc mode. The main services provided by the library are as follows.

- Handling notifications for players waiting to join the group and discovering their peers
- Negotiation
  - Notification of join requests
  - Notification of acceptance/denial of join requests
- Group management
  - Confirming that a player exists who has agreed to participate
  - Handling notifications for players who cancel their participation agreements

A matching protocol is implemented in the library using the ad hoc mode communication protocol. An application uses these services through the library API and through event handlers that are called from the library.

Figure 1 Library Configuration



Used Resources

The following are the system resources consumed by the Ad hoc Matching library.

Resource	Description
Footprint	Approximately 20 KB (Total of Text, Data, RO-Data and BSS of `SCE_SYSMODULE_NET_ADHOC_MATCHING`)
Work memory	Only uses the memory passed from the application. Refer to the "Precautions" chapter for the appropriate memory size.

Thread	The following 3 threads are created in the application process.
	`"SceNetAdhocMatchingEvent"`
	`"SceNetAdhocMatchingInput"`
	`"SceNetAdhocMatchingCallout"`

Processor time	Negligible

Files

The files required for using the Ad hoc Matching library are as follows.

Description	File
Header file	adhoc_matching.h
Stub library file	libSceNetAdhocMatching_stub.a libSceNetAdhocMatching_stub_weak.a

The Ad hoc Matching library can be linked in the PRX format only. Link libSceNetAdhocMatching_stub.a or libSceNetAdhocMatching_stub_weak.a statically to use the Ad hoc Matching library. The PRX module is stored in storage managed by system software and can be loaded/unloaded using API of libsysmodule.

For details on the PRX format, refer to the "libsysmodule Overview" document.

Sample Programs

The following program is provided as an Ad hoc Matching library sample program for reference purposes.

sample_code/network/api_adhoc_matching/adhoc_matching_robot/

This is a sample using Ad hoc Matching in game form.

For sample information, refer to the readme.

Reference Materials

Concerning connection in the ad hoc communication mode, refer to the following document.

- Network Overview

2 Using the Library

Initialization

To use the Ad hoc Matching library, first initialize libnet and then initialize the Ad hoc Matching library.

(1) Loading the PRX

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

(2) Initializing the library

Load NET module, and perform initialization. For details, refer to the "libnet Overview" and "libnet Reference" documents.

Then, initialize the Ad hoc Matching library by `sceNetAdhocMatchingInit()`.

```
ret = sceNetAdhocMatchingInit(
              SCE_NET_ADHOC_MATCHING_POOLSIZE_DEFAULT,
              poolptr
              );
if (ret < 0) {
        // Error handling
}
```

Matching Process

(1) Start Matching

To start matching, create a context and call the start function.

```
int ctx_id;

// Specify the operating mode by mode and create a context
#define MATCHING_PORT          20000
#define MATCHING_RXBUFLEN      1024 // Approximately twice the maximum size of the
optional data that is handled
#define HELLO_INTERVAL         1 * 1000 * 1000 //msec
#define KEEPALIVE_INTERVAL     5 * 1000 * 1000 //msec
#define INIT_COUNT             10
#define REXMT_INTERVAL         1 * 1000 * 1000 //msec
ret = sceNetAdhocMatchingCreate(
        mode,
        SCE_NET_ADHOC_MATCHING_MAXNUM,
        MATCHING_PORT,
        MATCHING_RXBUFLEN,
        HELLO_INTERVAL,
        KEEPALIVE_INTERVAL,
        INIT_COUNT,
        REXMT_INTERVAL,
        eventHandler
        );
if (ret < 0) {
        // Error handling
}
ctx_id = ret;
```

```
//Start matching
ret = sceNetAdhocMatchingStart(ctx_id);
if (ret < 0) {
        //Error handling
}
```

(2) Select/Accept a Target Participating Player

Call `sceNetAdhocMatchingSelectTarget()` to select and accept a target participating player associated with a Hello or Request event.

```
//Select/accept player indicated by addr
ret = sceNetAdhocMatchingSelectTarget(ctx_id, &addr, optlen, opt);
if (ret < 0) {
        // Error handling
}
```

(3) Cancel/Deny Selection of Target Participating Player

Call `sceNetAdhocMatchingCancelTarget()` or `sceNetAdhocMatchingCancelTargetWithOpt()` to deny a Request event or cancel an agreement with a participating player who had previously been selected.

```
// Cancel/deny selection of participating player indicated by addr
ret = sceNetAdhocMatchingCancelTargetWithOpt(ctx_id, &addr, optlen, opt);
if (ret < 0) {
        // Error handling
}
```

(4) Get Member List

During the matching operation, you can get a list of members who belong to a group of participating game players by calling the `sceNetAdhocMatchingGetMembers()`.

```
// Get member list
ret = sceNetAdhocMatchingGetMembers(ctx_id, &memberNum, members);
if (ret < 0) {
        // Error handling
}

// Display member list
printf("MEMBER==>");
for (i=0; i<memberNum; i++) {
        sceNetInetNtop(
             SCE_NET_AF_INET,
             &member[i].addr,
             addrStr,
             sizeof(addrStr)
        );
        printf(" [%s]", addrStr);
}
printf("\n");
```

(5) Stop Matching

Call the stop matching function and delete the context.

```
// Stop matching
sceNetAdhocMatchingStop(ctx_id);
// Delete context
sceNetAdhocMatchingDelete(ctx_id);
```

Termination

(1) Termination Processing

Terminate the Ad hoc Matching library by calling `sceNetAdhocMatchingTerm()`. Delete all contexts before calling `sceNetAdhocMatchingTerm()`.

```
ret = sceNetAdhocMatchingTerm();
if (ret < 0) {
        // Error handling
}
```

(2) Unloading the PRX

Call `sceSysmoduleUnloadModule()` with `SCE_SYSMODULE_NET_ADHOC_MATCHING` specified as the module ID to unload the PRX.

Event Handler Function Sample

The Ad hoc Matching library reports various types of events by calling event handler functions. The following samples show the operations that should be performed by the event handlers.

(1) Start Matching

```
static char *eventtypestr[] = {
        "NONE",
        "HELLO",
        "REQUEST",
        "LEAVE",
        "DENY",
        "CANCEL",
        "ACCEPT",
        "ESTABLISHED",
        "TIMEOUT",
        "ERROR",
        "BYE",
        "DATA",
        "DATA_ACK",
        "DATA_TIMEOUT"
};

static void
handler_default(
        int id,
        int ev,
        struct SceNetEtherAddr *peer,
        int optlen,
        void *opt
        )
{
        char addrStr[16];

        sceNetInetNtop(
              SCE_NET_AF_INET,
              peer,
              addrStr,
              sizeof(addrStr)

        printf("%s from %s", eventtypestr[ev], addrStr);

        for (i = 0; i < optlen; i++) {
```

```
                printf("%02x", *((char *)opt+i));
        }
        printf("\n");
}
```

(2) Multiplayer Mode (Parent) Sample

```
static void
handler_parent(
        int id,
        int ev,
        struct SceNetEtherAddr *peer,
        int optlen,
        void *opt
        )
{
        int ret;

        switch (ev) {
        case SCE_NET_ADHOC_MATCHING_EVENT_REQUEST:
              // Ask user whether to accept request to participate
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_CANCEL:
              // Notify user that request to participate was canceled
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_LEAVE:
              // Notify user that agreement with target participating player
was canceled
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ESTABLISHED:
              // Notify user that agreement with target participating player
was established
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_TIMEOUT:
              // Notify user that there was no response from target participating
player
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_HELLO:
        case SCE_NET_ADHOC_MATCHING_EVENT_DENY:
        case SCE_NET_ADHOC_MATCHING_EVENT_ACCEPT:
              // These events do not occur
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ERROR:
              // Communication state error (perform error handling)
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_BYE:
              // Target participating player terminated Ad hoc Matching library
              break;
        }
}
```

(3) Multiplayer Mode (Child) Sample

```
static void
handler_child(
        int id,
        int ev,
        struct SceNetEtherAddr *peer,
        int optlen,
        void *opt
        )
{
        int ret;

        switch (ev) {
        case SCE_NET_ADHOC_MATCHING_EVENT_HELLO:
              // Notify user that a target participating player exists
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_LEAVE:
              // Notify user that an agreement with a target participating player
was canceled
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_DENY:
              // Notify user that a request to participate was denied
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ACCEPT:
              // Notify user that a request to participate was accepted
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ESTABLISHED:
              // Notify user that an agreement with a target participating player
was  established
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_TIMEOUT:
              // Notify user that there was no response from a target
participating player
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_REQUEST:
        case SCE_NET_ADHOC_MATCHING_EVENT_CANCEL:
              // These events do not occur
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ERROR:
              // Communication state error (perform error handling)
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_BYE:
              // Target participating player terminated Ad hoc Matching library
              break;
        }
}
```

(4) Peer-to-Peer Mode Sample

```
static void
handler_p2p(
        int id,
        int ev,
        struct SceNetEtherAddr *peer,
        int optlen,
        void *opt
        )
{
        int ret;

        switch (ev) {
        case SCE_NET_ADHOC_MATCHING_EVENT_HELLO:
              // Notify user that a target participating player exists
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_REQUEST:
              // Ask user whether to accept request to participate
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_LEAVE:
              // Notify user that an agreement with a target participating player
was canceled
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_DENY:
              // Notify user that a request to participate was denied
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_CANCEL:
              // Notify user that a request to participate was canceled
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ACCEPT:
              // Notify user that a request to participate was accepted
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ESTABLISHED:
              // Notify user that an agreement with a target participating player
was established
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_TIMEOUT:
              // Notify user that there was no response from a target
participating player
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_ERROR:
              // Communication state error (perform error handling)
              break;
        case SCE_NET_ADHOC_MATCHING_EVENT_BYE:
              // Target participating player terminated Ad hoc Matching library
              break;
        }
}
```

3 Explanation of Operation

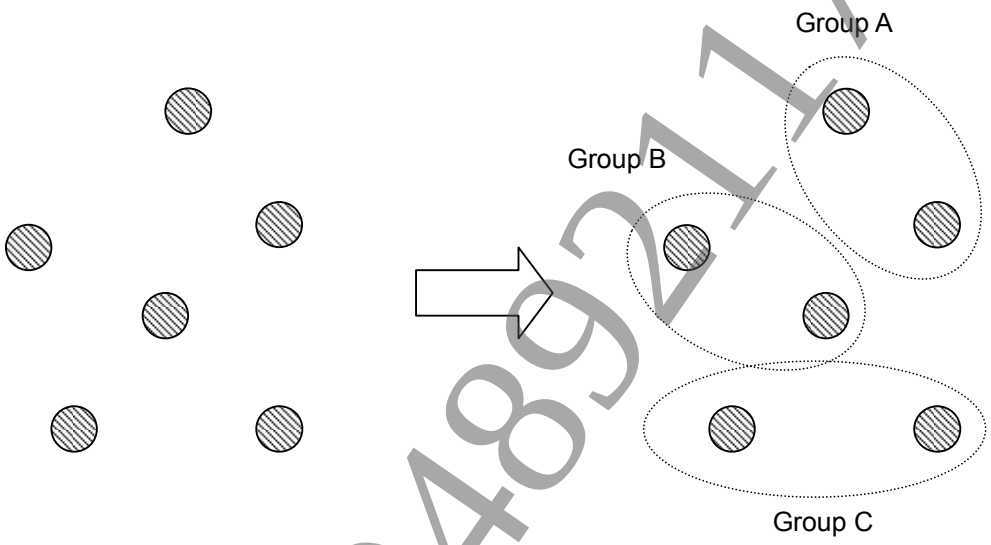
Group Models

The Ad hoc Matching library supports two models for setting up groups of participating players.

(1) Peer-to-Peer (Two-Player Competition)

A two-player model in which the participants are connected in a symmetric configuration. No parent-child relationship is required between the players. Each peer is configured such that it forms a group with one other arbitrary peer.

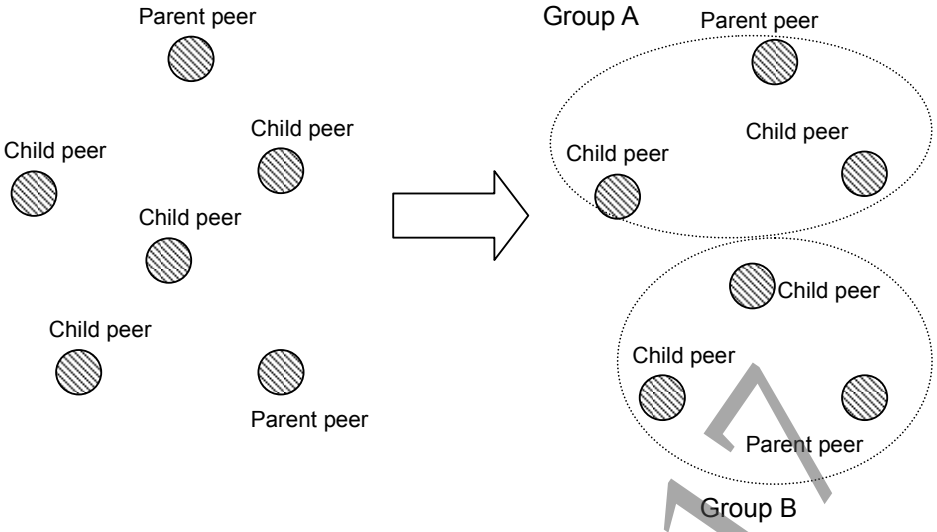
Figure 2 Peer-to-Peer Group Model



(2) Multiplayer

A model in which two or more participants are connected in a star configuration. This model has explicit parent-child relationships between the players. A parent peer is configured such that it forms a group with one or more child peers. Groups with two or more parent peers or groups consisting of only child peers are not permitted.

Figure 3 Multiplayer Group Model



When the library creates a context, the group model to be used for the group of participating game players is specified as the operating mode. If the multiplayer group model is specified, the choice of parent or child for the peer is also specified at the same time. The Ad hoc Matching library requires that peers that will be playing in the same group use the same group model and that the group model will be decided beforehand. In other words, a group cannot be made up of peers that use different group models.

The group model prescribes the structure of the group, but does not impose any restrictions on the communication paths used during actual game play. In other words, in the multiplayer model, each player can arbitrarily communicate with any other player or players without restriction.

Matching Protocol

The Ad hoc Matching library defines a matching protocol which is used to exchange matching information between peers.

Overview of Protocol Operation

The following is an overview of the matching protocol from initial negotiation until a group of participating game players is formed.

(1) A peer advertises that it is waiting to join a group
A peer set up in Peer-to-Peer or multiplayer (parent) mode starts sending a Hello message indicating that it is waiting to join a group. The Hello message is broadcast at regular intervals.

(2) Another peer makes a request to join the group
Another peer set up in Peer-to-Peer or multiplayer (child) mode discovers a waiting target by receiving its Hello message. If the user chooses, a JoinRequest message will be sent to the target.

(3) The first peer replies to the join request
The waiting peer discovers the target (the other player) by receiving its JoinRequest message. The user can choose to accept the target, in which case the peer sends a JoinReply message, or to deny the target, in which case it sends a Leave message.

(4) The peers establish an agreement or the request is denied
If the peer that sent the join request receives a JoinReply message it knows that its request has been accepted, and that an agreement for it to participate has been established. On the other hand, if the peer receives a Leave message, it knows that its join request was denied.

(5) Keep the connection alive
After an agreement to participate in the group has been established between the peers, they each confirm that their targets are alive by exchanging KeepAlive messages at regular intervals.

(6) Terminate the matching protocol
After the structure of the group has been decided, the matching protocol is terminated. A Bye message will be broadcast at this time to report the termination of the Ad hoc Matching library.

Timers and Counters

The matching protocol makes use of the following timers and counters.

- Hello interval timer
This timer sets the interval between successive Hello message transmissions.
- KeepAlive interval timer
This timer sets the interval between KeepAlive message transmissions.
- KeepAlive counter
Once an agreement to participate in a group has been established with a target player, the KeepAlive counter is used to check that the target is still alive. Every time the KeepAlive interval timer is started, the KeepAlive counter is decremented by 1. After KeepAlive messages are successfully exchanged between the peers, the counter is reset to its initial (non-zero) value. Whenever the counter reaches 0, a timeout is reported to the application.
- Resend timer
This timer sets the resend interval for JoinRequest messages, JoinReply messages, or data transmissions.
- Resend counter
This counter is used to monitor whether the target exists when JoinRequest or JoinReply messages are sent, or when data transmission takes place. Every time the resend timer is activated, the counter decrements by 1. Upon successful confirmation that the target exists, the counter is reset to its initial (non-zero) value. When the counter reaches 0, a timeout is reported to the application.

Event Model

The Ad hoc Matching library is designed so that an application can set up a group of participating players without having to be directly aware of the details of the matching protocol described above. This is accomplished by using events to exchange information between the application and the Ad hoc Matching library.

Application -> Ad hoc Matching library

An application reports events to the Ad hoc Matching library through API calls.

- Start event
Start matching. Corresponds to a call to `sceNetAdhocMatchingStart()`.
- Stop event
Stop matching. Corresponds to a call to `sceNetAdhocMatchingStop()`.
- Select event
Report a join request to the target player. Or, report that a previously received join request has been accepted. Corresponds to a call to `sceNetAdhocMatchingSelectTarget()`.
- Cancel event
Report a denial to a previously received join request. Alternatively, report the cancellation of a participation agreement with the target player. Corresponds to a call to `sceNetAdhocMatchingCancelTarget()` or `sceNetAdhocMatchingCancelTargetWithOpt()`.

Ad hoc Matching library -> Application

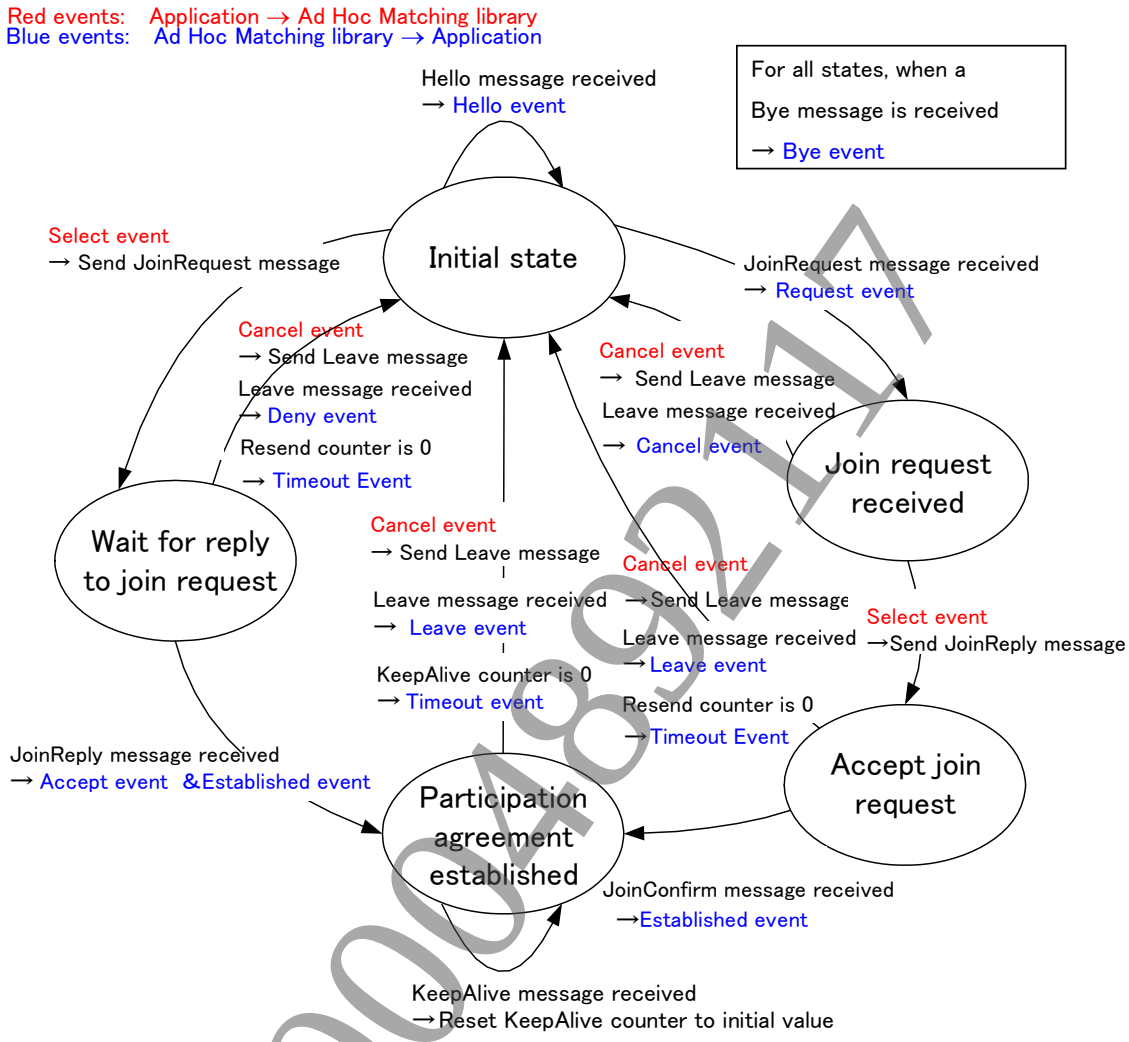
The Ad hoc Matching library reports events to an application by calling an event handler function.

- Hello event (`SCE_NET_ADHOC_MATCHING_EVENT_HELLO`)
  Reported when a Hello message is received.
- Request event (`SCE_NET_ADHOC_MATCHING_EVENT_REQUEST`)
  Reported when a join request is received.
- Accept event (`SCE_NET_ADHOC_MATCHING_EVENT_ACCEPT`)
  Reported when a join request is accepted.
- Deny event (`SCE_NET_ADHOC_MATCHING_EVENT_DENY`)
  Reported when a join request is denied.
- Cancel event (`SCE_NET_ADHOC_MATCHING_EVENT_CANCEL`)
  Reported when a join request is canceled.
- Established event (`SCE_NET_ADHOC_MATCHING_EVENT_ESTABLISHED`)
  Reported when a participation agreement is established. This event will not occur between CHILD peers.
- Leave event (`SCE_NET_ADHOC_MATCHING_EVENT_LEAVE`)
  Reported when a participation agreement is canceled (Leave is received).
- Timeout event (`SCE_NET_ADHOC_MATCHING_EVENT_TIMEOUT`)
  Reported when either the Resend counter or the Keep Alive counter has decremented to 0.
- Error event (`SCE_NET_ADHOC_MATCHING_EVENT_ERROR`)
  Reported when some error occurred in the matching protocol.
- Bye event (`SCE_NET_ADHOC_MATCHING_EVENT_BYE`)
  Reported when a Bye message is received.

State Transition Diagram

Figure 4 shows the relationships between state transitions and events between two peers.

Figure 4 Matching Protocol State Transition Diagram



Optional Data

The Ad hoc Matching library enables additional messages to be exchanged between peers by specifying optional data. The following messages can be extended by optional data.

(1) Hello message

The optional data specified in the argument of `sceNetAdhocMatchingStart()` is added to the Hello message. The optional data is reported to the target player together with the Hello event.

Optional data can be dynamically changed by using `sceNetAdhocMatchingSetHelloOpt()`. Also, the currently set optional data can be obtained by using `sceNetAdhocMatchingGetHelloOpt()`.

(2) JoinRequest message

The optional data specified in the argument of `sceNetAdhocMatchingSelectTarget()` is added to the JoinRequest message. The optional data is reported to the target player together with the JoinRequest event.

(3) JoinReply message

The optional data specified in the argument of `sceNetAdhocMatchingSelectTarget()` is added to the JoinReply message. The optional data is reported to the target player together with the Accept event.

(4) Leave message

The optional data that was specified in the argument of `sceNetAdhocMatchingCancelTargetWithOpt()` is added. The optional data that was added to the Leave message is reported to the target player together with the Cancel, Deny, or Leave event.

Data Transmission and Reception

The Ad hoc Matching library can send and receive arbitrary data between peers that are in an ESTABLISHED state. Therefore, direct means of communication between CHILD peers cannot be used in this library. The transmission and reception of data provided with this library are intended to be used for exchanging auxiliary data for matching. Use the socket for communication beyond this scope.

(1) Data Transmission

`sceNetAdhocMatchingSendData()` is used to send data to a specific peer. The data is resent the number of times indicated by the resend counter with the interval specified by the resend timer until a confirmation response is received from the peer. The reception of a confirmation response is reported by a Data Ack event. If a confirmation response is not received, such as in the case of a timeout, then a Data Timeout event is reported.

New data cannot be sent until a Data Ack or Data Timeout event is reported. To send new data before that time, first use `sceNetAdhocMatchingAbortSendData()` to cancel the current data transmission.

If the peer was not in an ESTABLISHED state (a Leave event, Timeout event, or Error event was reported), no Data Ack or Data Timeout event is generated.

(2) Reception

When data is received, a Data event is reported along with the data.

4 Precautions

Specifying the Pool Size When Initializing the Library

Memory that the Ad hoc Matching library uses internally is allocated from an independent memory pool, which is allocated when the Ad hoc Matching library is initialized. The amount of memory required depends on such factors as the number of contexts that are used, number of members that are handled by each context, and the size of the optional data that is handled.

The maximum amount of memory needed per context is determined by the items listed below. Use approximately 100 bytes for the maximum optional data size if no optional data is used.

- rxbuflen size specified in `sceNetAdhocMatchingCreate()`
  (approximately twice the maximum size of the optional data that is handled)
- hellooplen size specified in `sceNetAdhocMatchingStart()`
- optlen size specified in `sceNetAdhocMatchingSelectTarget()`
- optlen size specified in `sceNetAdhocMatchingCancelTargetWithOpt()`
- Size of temporary buffer when sending a message (approximately twice the maximum size of the optional data that is handled)
- Size of temporary buffer when receiving a message (approximately twice the maximum size of the optional data that is handled)
- Size of context management area (approximately 200 bytes)
- Size of member list management area (approximately 300 bytes)

The amount of memory needed for each member that is handled by each context is as follows.

- Size of member management area (approximately 100 bytes)

Therefore, using the following values:

- Number of contexts: 1
- Maximum size of optional data that is handled: 1000 bytes
- hellooptlen: 1000 bytes
- optlen: 1000 bytes
- Number of members: 4

the total amount of memory needed would be at most approximately 9900 bytes.

However, this value assumes no memory fragmentation, so in actuality a larger pool size must be specified.

Conditions for Sending Hello Messages

Although Peer-to-Peer mode and multiplayer mode parents send Hello messages at a specified interval, transmission is automatically stopped when the number of members of a matching group who have established a participation agreement reaches the maximum number of members as specified by `sceNetAdhocMatchingCreate()`. When a participation agreement is canceled and the number of members becomes less than the maximum number of members, the sending of Hello messages is automatically restarted.

When the Number of Connected Players Reaches the Maximum

When a Peer-to-Peer mode player or multiplayer mode parent has already established a participation agreement with the maximum number of members allowed (the maximum capacity has been reached), then even if a new JoinRequest message is sent to this peer, no Request event will be generated. In this case, the library automatically returns a Leave message internally, and a Deny event is reported to the peer that sent the JoinRequest message.

Moreover, because the sending of Hello messages will be stopped when a Peer-to-Peer mode player or multiplayer mode parent establishes participation agreement with the maximum number of members allowed, subsequent devices that attempt to join will be unable to discover the parent device until the participation agreement is canceled and the number of members becomes less than the maximum number of members.

SCE_NET_CTL_ERROR_NOT_CONNECTED

The following function will return `SCE_NET_CTL_ERROR_NOT_CONNECTED` if no connection has been established to the ad hoc network. Be sure to call this function only after a connection is established.

- `sceNetAdhocMatchingStart()`