Sega Dreamcast

Network Information

Sega Dreamcast

Network Information Table of Contents

1. Preface

When using Dream Passport to sign up with the SEGA provider, modem settings and provider settings are stored in the internal flash memory. Applications can use this information to connect to the network.

This document describes the organization of the `NetworkInfo` structure which can be obtained using the `ntInf` functions.

2. Structures

NetworkInfo

```
typedef struct _NetworkInfo {
          unsigned long                    flag;
          NetworkAccessInfo                isp_info1;
          NetworkAccessInfo                isp_info2;
          unsigned long                    reserve;
} NetworkInfo;
```

flag – The bit definition is given below.

Bit definition	Meaning
ISP_USE1	Use isp_info1 (SEGA provider). (This value is 0.)
ISP_USE2	Use isp_info2 (user provider).
CHANGE_NUMBER	When two or more telephone numbers are set in AccessPointNumber, the numbers will be tried in sequence when the first number is busy. (This value is 0.)
FIXED_NUMBER	Even if the number is busy, only the first number set in AccessPointNumber will be redialed.
DIAL_TONE	Tone dialing. Corresponds to the AT command ATDT. (This value is 0.)
DIAL_PULSE	Pulse dialing. Corresponds to the AT command ATDP.
PULSE_US_10	10 pps. Corresponds to AT&P0. (This value is 0.)
PULSE_UK_10	10 pps. Corresponds to AT&P1.
PULSE_US_20	20 pps. Corresponds to AT&P2.
PULSE_UK_20	20 pps. Corresponds to AT&P3.
SPEAKER_VOL_L0	Speaker volume low. Corresponds to ATL0. (This value is 0.)
SPEAKER_VOL_L1	Speaker volume low. Corresponds to ATL1.
SPEAKER_VOL_L2	Speaker volume medium. Corresponds to ATL2.
SPEAKER_VOL_L3	Speaker volume high. Corresponds to ATL3.
SPEAKER_CTL_M0	Speaker is always off. Corresponds to ATM0. (This value is 0.)
SPEAKER_CTL_M1	Speaker is on from start of dialing until receiving carrier. Corresponds to ATM1.
SPEAKER_CTL_M2	Speaker is always on. Corresponds to ATM2.
SPEAKER_CTL_M3	Speaker is off from after dialing until receiving carrier. Corresponds to ATM3.
BLIND_ENABLE	No tone detection is performed. Corresponds to ATX.

`isp_info1` (mandatory) – SEGA provider information. To connect to the network, all users must register with the SEGA provider.

`isp_info2` (optional) – User provider information. The user can also access the Internet using his or her own ISP.

`reserve` – Reserved

Note: To check whether information in this structure is correct, "magic" in `isp_info1` is used. When the API described below is used to obtain information, the check is carried out by the API. In this case, check the API return value.

NetworkAccessInfo

```
typedef struct _NetworkAccessInfo {
    unsigned long   magic;
    char            loginId[28];
    char            loginPasswd[16];
    char            accessPointNumber[3][40];
    struct in_addr  dnsServerAddress1;
    struct in_addr  dnsServerAddress2;
    char            mailAddress[48];
    char            mailServer[32];
    char            popServer[32];
    char            popId[16];
    char            popPasswd[16];
    char            proxyServer[32];
    unsigned short  proxyPortNum;
    unsigned short  reserve;
} NetworkAccessInfo;
```

magic – This is used to check whether correct information is set in the structure. It always must be 0x41474553("SEGA").

loginId – PPP login ID

loginPasswd – PPP login password

accessPointNumber – Access point telephone number. There are three entries, and at least one entry must be filled. The string to follow the ATDT (or ATDP) command must be entered here. Blank entries are padded with 0.

dnsServerAddress1 – IP address of primary DNS (long format, network byte order). 0 if DNS is not used.

dnsServerAddress2 – IP address of secondary DNS (long format, network byte order). 0 if DNS is not used._

mailAddress – Mail address

mailServer – Name of SMTP server.

popServer – Name of POP3 server

popId – POP3 ID

popPasswd – POP3 password

proxyServer – Name of proxy server

proxyPortNum – Proxy port number

reserve – Reserved

Note: The char array elements must be filled to the full length of the element. For example, the login ID element is 28 characters. If the actual entry is shorter, the rest must be padded with 0 (zeros).

API

Overview

The ntInf functions serve to obtain network information stored by Dream Passport. Except for `ntInflsNetworkInfo`, the functions must always be initialized with `ntInfInit` and terminated with `ntInfExit`.

Network information is structured as described in section 2. NetworkInfo and NetworkAccessInfo can be obtained in full. To obtain individual elements, refer to the structure description and determine the size of the copy target buffer accordingly.

For example, the Login ID information will require a 28- byte buffer. Because an element may be used until the last byte of its size, you should add 1 byte to the buffer size if the element is to be treated as a 0-terminate string.

Return Values

These return values are common to all functions.

```
#define NTD_OK                  ( 0)
```

No error

```
#define NTD_ERR_NOTINIT         (-1)
```

Not initialized. The ntInf function can only be used after initialization with ntInfInit.

```
#define NTD_ERR_INVALIDPARAM    (-2)
```

Invalid parameter

```
#define NTD_ERR_NOINFO          (-3)
```

Network information is not stored. Information must first be stored with Dream Passport.

3. Network Information API

Overview

The `ntInf` functions serve to obtain network information stored by Dream Passport. Except for `ntInflsNetworkInfo`, the functions must always be initialized with ntInfInit and terminated with `ntInfExit`.

Network information is structured as described in Chapter 2. `NetworkInfo` and `NetworkAccessInfo` can be obtained in full. To obtain individual elements, refer to the structure description and determine the size of the copy target buffer accordingly.

For example, the Login ID information will require a 28- byte buffer. Because an element may be used until the last byte of its size, you should add 1 byte to the buffer size if the element is to be treated as a 0-terminate string.

Return Values

These return values are common to all functions.

```
#define NTD_OK                          (  0)
```

No error

```
#define NTD_ERR_NOTINIT                 ( -1)
```

Not initialized. The ntInf function can only be used after initialization with ntInfInit.

```
#define NTD_ERR_INVALIDPARAM            ( -2)
```

Invalid parameter

```
#define NTD_ERR_NOINFO                  ( -3)
```

Network information is not stored. Information must first be stored with Dream Passport.

Functions

ntInflnit

Initialization function.

Description

Initialization function. Except for `ntInflsNetworkInfo`, this must always be executed before an `ntInf` function.

```
Sint32 ntInfInit(void *pNetInfo, void *pWork);
```

Arguments

`pNetInfo`(output):	Specifies the buffer for receiving `NetworkInfo`. When `NULL` is specified, malloc within API is used.
`pWork`(input):	Specifies a 16-KB work area. When `NULL` is specified, malloc within API is used.

Example

```
Sint32 rc;
NetworkInfo *netinfo;

    rc = ntInfInit(NULL, NULL);   /* Initialize ntInf */
    if (rc == NTD_OK) {
        rc = ntInfGetNetworkInfo(&netinfo);
        if (rc == NTD_ERR_NOINFO) {
            /* Information not stored with Dream Passport */
        }
        ntInfExit();    /* Terminate ntInf */
    }
```

ntInfExit

Shutdown function.

Description

Shutdown function. This must always be executed after an `ntInf` function.

```
Sint32 ntInfExit(void);
```

ntInfGetNetworkInfo

Gets NetworkInfo.

Description

Gets NetworkInfo.

```
Sint32 ntInfGetNetworkInfo(NetworkInfo **ppNetInfo);
```

Arguments

`ppNetInfo`(output):	Copies `NetworkInfo` pointer to `ppNetInfo`.

Example

```
/* Check whether network information has been stored */
Sint32 rc;
NetworkInfo *netinfo;

rc = ntInfInit(NULL, NULL);   /* Initialize ntInf */
if (rc == NTD_OK) {
    rc = ntInfGetNetworkInfo(&netinfo);
    if (rc == NTD_ERR_NOINFO) {
        /* Information not stored with Dream Passport */
    } else {
       /* Reference required information */
      }
    ntInfExit();    /* Terminate ntInf */
}
```

ntInfIsNetworkInfo

Checks whether network information has been stored with Dream Passport.

Description

Checks whether network information has been stored with Dream Passport. Can be used as stand-alone function without calling `ntInfInit` or `ntInfExit`.

```
Sint32 ntInfIsNetworkInfo(void);
```

Arguments

Example

```
/* Check whether network information has been stored */
 Sint32 rc;
 rc = ntInfIsNetworkInfo();
 if (rc == NTD_ERR_NOINFO) {
     /* Information not stored with Dream Passport */
 }
```

Remarks

Internally calls `ntInfInit(NULL,NULL), ntInfExit()`.

ntInfGetNetworkAccessInfo

Gets NetworkAccessInfo.

Description

Gets NetworkAccessInfo.

```
Sint32 ntInfGetNetworkAccessInfo(Sint32 isp, NetworkAccessInfo **ppNetAccInfo);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`ppNetAccInfo`(output):	Copies NetworkAccessInfo pointer.

ntInfGetFlag

Gets flag information.

Description

Gets flag information.

```
Sint32 ntInfGetFlag(Uint32 *fFlag);
```

Arguments

`fFlag`(output):	Copies flag information.

ntInfGetLoginId

Gets login ID.

Description

Gets login ID.

```
Sint32 ntInfGetLoginId(Sint32 isp, Uint8 *pLoginId);
```

Arguments

isp(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
pLoginId(output):	Specifies buffer where login ID is to be copied.

ntInfGetLoginPasswd

Gets login password.

Description

Gets login password.

```
Sint32 ntInfGetLoginPasswd(Sint32 isp, Uint8
*pLoginPasswd);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pLoginPasswd`(output):	Specifies buffer where login password is to be copied.

ntInfGetAccessPointNumber

Gets access point telephone number.

Description

Gets access point telephone number.

```
Sint32 ntInfGetAccessPointNumber(Sint32 isp, Uint8
*pAccessPointNumber);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pAccessPointNumber`(output):	Specifies buffer where access point telephone number is to be copied. Information for three numbers (120 bytes) is to be copied.

ntInfGetDnsServerAddress

Gets DNS IP address.

Description

Gets DNS IP address. When automatic DNS setting with IPCP is used, `0.0.0.0` is returned.

```
Sint32 ntInfGetDnsServerAddress(Sint32 isp, struct in_addr *pDns1, struct in_addr *pDns2);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pDns1`(output):	Primary DNS is to be copied. Address is in network byte order.
`pDns2`(output):	Secondary DNS is to be copied. Address is in network byte order.

ntInfGetMailAddress

Gets mail address.

Description

Gets mail address.

```
Sint32 ntInfGetMailAddress(Sint32 isp, Uint8 *pMailAddress);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pMailAddress`(output):	Specifies buffer where mail address is to be copied.

ntInfGetMailServer

Gets SMTP server name.

Description

Gets SMTP server name.

```
Sint32 ntInfGetMailServer(Sint32 isp, Uint8 *pMailServer);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pMailServer`(output):	Specifies buffer where SMTP server name is to be copied.

ntInfGetPopServer

Gets POP3 server name.

Description

Gets POP3 server name.

```
Sint32 ntInfGetPopServer(Sint32 isp, Uint8 *pPopServer);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pPopServer`(output):	Specifies buffer where POP3 server name is to be copied.

ntInfGetPopId

Gets POP3 ID.

Description

Gets POP3 ID.

```
Sint32 ntInfGetPopId(Sint32 isp, Uint8 *pPopId);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pPopId`(output):	Specifies buffer where POP3 ID is to be copied.

ntInfGetPopPasswd

Gets POP3 password.

Description

Gets POP3 password.

```
Sint32 ntInfGetPopPasswd(Sint32 isp, Uint8 *pPopPasswd);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pPopPasswd`(output):	Specifies buffer where POP3 password is to be copied.

ntInfGetProxyServer

Gets proxy server name.

Description

Gets proxy server name.

```
Sint32 ntInfGetProxyServer(Sint32 isp, Uint8 *pProxyServer);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pProxyServer`(output):	Specifies buffer where proxy server name is to be copied.

ntInfGetProxyPortNum

Gets proxy server port number.

Description

Gets proxy server port number.

```
Sint32 ntInfGetProxyPortNum(Sint32 isp, Uint16 *pPort);
```

Arguments

`isp`(input):	SEGA provider: 1; user provider: 2; get information for active provider: 0
`pPort`(output):	Specifies buffer where proxy server port number is to be copied.

Sega Dreamcast

AVE-PPP Script Specifications Guide

Access Co., Ltd.

Sega Dreamcast

AVE-PPP Script Specifications Guide TOC

1. AVE-PPP Script Specifications

Describe one command in one line in the order mentioned in the following, delimiting the present line and next line by 0x0a(LF).

In the case of no parameter	`<command>`
In the case of 1 parameter	`<command> <parameter>`
In the case of more than 1 parameter	`<command> <first parameter>, <second parameter>, ..., <n parameter>`

After the last line, 0x0a(LF), 0x00(null) is added to indicate that it is the last line.

In the case of the parameter having a numeric value, it is described as is. In the case of being a character string, it is enclosed by double quotation marks (").

AVE-PPP executes delivered script one line at a time. When the result of the command executed is an error, script execution is interrupted. In the case of completing all of the commands described up to the last line, the AVE-PPP judges script execution as successful and then each PPP connecting process is executed.

Example)

```
send "ATZ\r"<LF>
wait "OK",5<LF>
delay 1<LF>
dial "%t"<LF>
wait "ogin",5<LF>
send "%u\r"<LF>
wait "assword",5<LF>
send "%p\r"<LF>
wait "elcom",5<LF>
<NUL>
```

Commands

dial <telephone number character string>

When the character string is assumed to be a telephone number, the ATDT or STDP command is issued. Additionally, character strings returned by the modem, such as `BUSY`, `NO DIALTONE`, `CONNECT`, etc. are judged to determine success or failure. In the case of a failure, script execution is interrupted.

```
Example)
    dial "0123-456-789"
    dial "0,0123-456-789"
    dial "%t"
```

send <Transmission character string>

Character string designated for transmission.

wait "character string for receiving time", number of seconds

Time required to wait to receive the designated character string. However, if it has not been received after the designated time has past, script execution is interrupted.

delay, number of seconds

Wait a predetermined amount of time.

Character String List

Definitions inside the character string are enclosed in double quotation marks, so declarations such as the following are possible.

%t

Indicates the telephone number for connecting to the objective destination. It is necessary to deliver the telephone number to the AVE-PPP directory with a structure separate from script. In the case where `Extension use` is designated, the character string for sending the extension (for example, `0w`) is added automatically.

%u

Indicates personal ID character string.

%p

Indicates personal password character string.

%%

Use this when percent (%) itself is entered in the character string.

\r

Indicates return (0x0d).

\n

Indicates line feed (0x0a).

\¥°°

Value of the 2-digit hexadecimal is inserted in °°, indicating the word of the corresponding character code.

```
Example)    \¥20    Same as null (0¥20)
```

\"

Indicates double quotations.

\\

Indicates yen (\) currency mark

Sega Dreamcast

AVE-SOCKET ™ API Specifications V. 1.0

Access Co., Ltd.

Sega Dreamcast

AVE-SOCKET API Specifications TOC

1. Preface

These specifications describe AVE-SOCKET, which runs on the AVE-TCP that Access has developed for embedded devices.

AVE-SOCKET complies with the TCP/IP communications standard interface for UNIX and Windows 95/98/NT, and provides a similar interface in embedded environments.

Features of the AVE-SOCKET API are listed below.

- Does not require much memory when running.
- Conforms with the BSD socket, allowing programs that run in UNIX or on a PC to run in AVE-SOCKET with only minor modifications.
- Specialized for an embedded environment, AVE-SOCKET permits parallel programming for multiple ports without an OS by using a non-blocking interface.
- Because all APIs are coded in C, they flexibly support a wide range of CPU platforms, including x86, 680xx, SHx, V8xx, M32R/x, ARM, MIPS, and PowerPC.

When Reading This Manual

This manual is intended for readers with a basic knowledge of the Internet, TCP/IP, and BSD socket interface. For details on these topics, it will be necessary to refer to other sources.

Overview

An overview of the AVE-SOCKET functions is provided below.

- AVE-SOCKET provides APIs that conform with the BSD socket interface.
- AVE-SOCKET permits selection of a blocking or non-blocking interface.

2. AVE-SOCKET API

AVE-SOCKET supports the following APIs:

Description name	Terse desription
accept	Accepts connection to a socket.
bind	Assigns a name to a socket.
connect	Opens a connection to a socket.
getpeername	Gets the name of the peer on the other side of the connection.
getsockname	Gets the socket name.
getsockopt	Gets the socket options.
setsockopt	Sets the socket options.
listen	Shifts to the connection wait state in the socket.
recv	Receives data from the socket (stream).
recvfrom	Receives data from the socket (datagram).
select	Multiplexes synchronous I/O.
send	Sends data to the socket (stream).
sendto	Sends data to the socket (datagram).
shutdown	Partial shutdown of full-duplex communication.
socket	Creates a socket for communications.
sock_close	Closes a socket.
sock_read	Reads data from a socket.
sock_write	Writes data to a socket.
htonl	Converts the byte order.
htons	Converts the byte order.
ntohl	Converts the byte order.
ntohs	Converts the byte order.

Descriptional descriptions of each API are found on the pages that follow.

accept

Accepts connection to a socket.

Description

When there is a connection request in the socket descriptor that is indicated by parameter "s," this API writes the address information for the connection in "addr." If the socket is a blocked socket, control does not return until the connection request is accepted. If the socket is a non-blocked socket, control returns as soon as the connection information is written in "addr" if the connection has already been accepted. If the connection has not yet been accepted, "EWOULDBLOCK" is set in "errno" and then control returns immediately.

```
#include "types.h"
#include "socket.h"

int accept(int s, struct sockaddr *addr, int *addrlen)
```

Input

`int s;`	Socket descriptor
`struct sockaddr *addr;`	Area where the connection address information is to be returned
`int *addrlen;`	Length of address information (and area where length is to be returned)

Output

struct sockaddr *addr;	Connection address information is written here
int *addrlen;	Length of address information is written here

Return Value

0: Normal end

-1: Abnormal end

In the event of an abnormal end, the error details are set in "errno."

Error

EBADF	Not a valid descriptor.
ENOTSOCK	Descriptor is not a socket.
EOPNOTSUPP	The referenced socket is not of the stream type.
EWOUDBLOCK	The socket is a non-blocked socket, and there is no connection request that has been accepted.

bind

Assigns a name to a socket.

Description

This API assigns a name to a socket that does not have a name.

"bind" specifies a local IP address and protocol port number to a socket. "bind" is usually used by a server that needs to specify a "well-known" port.

```
#include "types.h"
#include "socket.h"

int bind(int s, struct sockaddr *name, int namelen)
```

Input

`int s;`	Socket descriptor
`struct sockaddr *name;`	Address information area
`int namelen;`	Address information length

Return Value

0: Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "errno."

Error

EADDRINUSE	The specified address is in use.
EBADF	Not a valid descriptor.
EINVAL	The size that was specified by "namelen" is invalid in the specified address family.
ENOTSOCK	"s" is not a socket descriptor.

connect

Opens a connection to a socket.

Description

When the socket is a stream-type socket, this API tests the connection with the address that is specified by "name."

When the socket is a datagram-type socket, this API specifies the communications endpoint for the address that is specified by "name," but does not send a packet.

```
#include "types.h"
#include "socket.h"

int connect(int s, struct sockaddr *name, int namelen)
```

Input

`int s;`	Scket descriptor
`struct sockaddr *name;`	Area where the address information is to be returned
`int namelen;`	Address information length

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "errno."

Error

EADDRINUSE	The specified address is already in use.
EADDRNOTAVAIL	The specified address cannot be used on a remote machine.
EAFNOSUPPORT	Addresses of the specified address family cannot be used with this socket.
ECONNREFUSED	The connection request was forcibly refused.
EINPROGRESS	The socket is a non-blocked socket, and the previous connection procedure is not yet completed.
EISCONN	This socket is already connected.
ENETUNREACH	The specified network cannot be accessed from this host.
ENOTSOCK	The descriptor is not a socket.
ETIMEDOUT	The connection timed out before it was completed.
EPROTOTYPE	The socket identifier referenced by name is of a different socket type than "s".

getpeername

Gets the name of the peer on the other side of the connection.

Description

This API returns the name (address information) of the peer on the other side to which the socket is connected.

```
#include "types.h"
#include "socket.h"

int getpeername(int s, struct sockaddr *name, int namelen)
```

Input

`int s;`	Socket descriptor
`struct sockaddr *name;`	Area where the connection address information is to be returned
`int *namelen;`	Length of address information (and area where length is to be returned)

Output

`struct sockaddr *name;`	Connection address information is written here
`int *namelen;`	Length of address information is written here

Return Value

0:Normal end
-1:Abnormal end

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOTCONN`	The socket is not connected.
`ENOTSOCK`	Descriptor is not a socket.

getsockname

Gets the socket name.

Description

This API returns the name (address information) that was assigned to the socket.

```
#include "types.h"
#include "socket.h"

int getsockname(int s, struct sockaddr *name, int *namelen)
```

Input

`int s;`	Socket descriptor
`struct sockaddr *name;`	Area where the socket adress information is to be returned
`int *namelen;`	Length of address information (and area where length is to be returned)

Output

`struct sockaddr *name;`	Socket address information is written here
`int *namelen;`	Length of address information (and area where length is to be returned)

Return Value

0:Normal end
-1:Abnormal end

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOTSOCK`	Descriptor is not a socket.

getsockopt

Gets the socket options.

Description

This API gets the socket option value.

```
#include "types.h"
#include "socket.h"

int getsockopt(int s, int level, int optname, char *optval, int *optlen)
```

Input

`int s;`	Socket descriptor
`int level;`	Protocol level (`SOL_SOCKET` can be specified)
`int optname;`	Option type (`SOL_NOBLK` can be specified)
`char *optval;`	Option value area
`int *optlen;`	Option length area

Output

`char *optval;`	Option value
`int *optlen;`	Option length

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOTPROTOPT`	The option is not recognized in the specified level.
`ENOTSOCK`	Descriptor is not a socket.

setsockopt

Sets the socket options.

Description

This API sets the socket option. The value that is specified can be referenced by using "getsockopt."

SOL_NOBLK

0: Sets the socket as a blocked socket. (default value)

1: Sets the socket as a non-blocked socket.

```
#include "types.h"
#include "socket.h"

int setsockopt(int s, int level, int optname, char *optval, int optlen)
```

Input

int s;	Socket descriptor
int level;	Protocol level (SOL_SOCKET can be specified)
int optname;	Option type (SOL_NOBLK can be specified)
char *optval;	Option value area
int *optlen;	Option length area

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "errno."

Error

ENOTPROTOPT	The option is not recognized in the specified level.
ENOTSOCK	Descriptor is not a socket.

listen

Shifts to the connection wait state in the socket.

Description

This API sets up the socket to accept a connection.

Only a stream-type socket is valid for socket "s."

```
#include "types.h"
#include "socket.h"

int listen(int s, int backlog)
```

Input

`int s;`	Socket descriptor
`int backlog;`	Maximum number of connections that can be waiting to be accepted

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "errno."

Error

ENOTSOCK	Descriptor is not a socket.
EOPNOTSUPP	Socket "s" does not support "listen."

Note

The "backlog" specification is currently invalid.

recv

Receives data from the socket (stream).

Description

This API receives data from a stream-type socket.

Control returns immediately if there is reception data in the stream.

If there is no reception data in the stream and the socket is blocked, control does not return from the reception call until data arrives. If there is no reception data in the stream and the socket is not blocked, "EWOULDBLOCK" is set in the external variable "errno" and "-1" is returned.

```
#include "types.h"
#include "socket.h"

int recv(int s, char *buf, int len, int flags)
```

Input

int s;	Socket descriptor
char *buf;	Address where the received data is held
int len;	Length of the receive buffer
int flags;	Load control flags (currently invalid; specify "0")

Output

char *buf;	The received data is stored here

Return Value

0 or more:Normal end	Returns the number of bytes that were received
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "errno."

Error

EFAULT	
ENOTPROTOPT	The option is not recognized in the specified level.
EWOULDBLOCK	The socket is not blocked, and no connection request has been received.
ECONNRESET	The reset signal was received.
ENOTCONN	The socket is not connected.
EPROTONOSUPPORT	The specified protocol is not supported.
ENOTSOCK	"s" is not a valid socket descriptor.

recvfrom

Receives data from the socket (datagram).

Description

This API receives data from a datagram-type socket.

Control returns immediately if there is reception data in the datagram.

If there is no reception data in the datagram and the socket is blocked, control does not return from the reception call until data arrives. If there is no reception data in the datagram and the socket is not blocked, "EWOULDBLOCK" is set in the external variable "errno" and "-1" is returned.

```
#include "types.h"
#include "socket.h"

int recvfrom(int s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen)
```

Input

`int s;`	Socket descriptor
`char *buf;`	Address where the received data is held
`int len;`	Length of the receive buffer
`int flags;`	Load control flags (currently invalid; specify "0")
`struct sockaddr *from;`	Area for source address information
`int *fromlen;`	Area for length of source address information

Output

`char *buf;`	The received data is stored here
`struct sockaddr *from;`	Source address information
`int *fromlen;`	Length of source address information

Return Value

0 or more:Normal end	Returns the number of bytes that were received
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "errno."

Error

EFAULT	
ENOTPROTOPT	The option is not recognized in the specified level.
EWOULDBLOCK	The socket is not blocked, and no connection request has been received.
EPROTONOSUPPORT	The specified protocol is not supported.
ENOTSOCK	"s" is not a valid socket descriptor.

select

Multiplexes synchronous I/O.

Description

The "select" API surveys the grouping of socket descriptors pointed at by "readfds," "writefds," and "exceptfds;" then, of those descriptors, this API checks those indicated by "readfds" for those that are ready to be read ("connect," "recv," and "recvfrom"), those indicated by "writefds" for those that are ready to be written, and those indicated by "exceptfds" for those that have generated an exception condition, and then returns those groupings.

If there is at least one socket that is ready for I/O when "select" is called, control returns immediately. If the timeout specification has been made, the socket is blocked either until the socket is ready for I/O, or the timeout time elapses.

Before the user sets the socket grouping, the socket grouping must be initialized by the FD_ZERO and FD_CLR macros. The FD_SET macro must be used to set the socket grouping.

If "timeout" is other than the NULL pointer, the timeout specification has been made; "timeout" specifies the maximum waiting time. When "timeout" is the NULL pointer, the program waits indefinitely until an event occurs. If "0" is set for both "t_sec" and "u_sec," control returns immediately, regardless of whether an event occurs.

The NULL pointer is specifies for unused socket groupings.

```
#include "types.h"
#include "time.h"

int select(int width, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval
*timeout)
```

Input

`int width;`	Maximum number of socket descriptors that should be checked
`fd_set *readfds;`	Socket grouping that is the target of the read check
`fd_set *writefds;`	Socket grouping that is the target of the write check
`fd_set *exceptfds;`	Socket grouping that is the target of the exception check (Currently not supported.)
`timeval *timeout;`	Timeout value

Output

`fd_set *readfds;`	Grouping of sockets that have been read
`fd_set *writefds;`	Grouping of sockets that have been written
`fd_set *exceptfds;`	Grouping of sockets for which an exception has been generated (Currently not supported.)

Return Value

0 or more:Normal end	Number of sockets ready for I/O
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "errno."

Error

EBADF	The socket descriptor grouping contains an invalid descriptor.
EINVAL	The specified time limit is outside of the allowable range. The range that can be set by the user is 0 <= t_sec <= 100000000, 0 <= uv_sec < 1000000.

Note

When not using an OS, the minimum time that can be specified for "timeout" depends on the precision of the timer that is being used. When using an OS, the minimum time that can be specified for "timeout" depends on the minimum unit of time that is managed by the OS. (We cannot guarantee precision at the level of microseconds.)

send

Sends data to the socket (stream).

Description

This API sends a data stream to the socket.

```
#include "types.h"
#include "socket.h"

int send(int s, char *msg, int len, int flags)
```

Input

`int s;`	Socket descriptor
`char *msg;`	Area where the data that is to be sent is stored
`int len;`	Size of the data to be sent
`int flags;`	Control bit string (This is currently not supported, so specify "`0`.")

Return Value

0 or more:Normal end	Returns the number of bytes that were sent.
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOPROTOOPT`	The option is not recognized in the specified level.
`ENOTSOCK`	Descriptor is not a socket.
`EBADF`	Not a valid descriptor.
`ENOTPROTOPT`	The option is not recognized in the specified level.
`ECONNRESET`	The reset signal was received.
`ENOTCONN`	The socket is not connected.

sendto

Sends data to the socket (datagram).

Description

This API sends a datagram to the socket.

```
#include "types.h"
#include "socket.h"

int sendto(int s, char *msg, int len, int flags, struct sockaddr *to, int tolen)
```

Input

`int s;`	Socket descriptor
`char *msg;`	Area where the data that is to be sent is stored
`int len;`	Size of the data to be sent
`int flags;`	Control bit string (This is currently not supported, so specify "`0`.")
`struct sockaddr * to;`	Destination address information
`int tolen;`	Length of destination address information

Return Value

0 or more:Normal end	Returns the number of bytes that were sent.
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOTPROTOPT`	The option is not recognized in the specified level.
`ENOTSOCK`	Descriptor is not a socket.
`EBADF`	Not a vaild descriptor.

shutdown

Partial shutdown of full-duplex communication.

Description

This API shuts down all or some of the full-duplex sockets that are connected. If "how" is "0," subsequent reception is not possible. If "how" is "1," subsequent sending is not possible. If "how" is "2," subsequent reception and sending are not possible.

```
#include "types.h"
#include "socket.h"

int shutdown(int s, int how)
```

Input

`int s;`	Socket descriptor
`int how;`	0:Read side is closed.
	1:Write side is closed.
	2:Read and write sides are closed.

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in c

Error

ENOTPROTOPT	The option is not recognized in the specified level.
ENOTSOCK	Descriptor is not a socket.
EBADF	Not a vaild descriptor.

socket

Creates a socket for communications.

Description

This API creates a socket for communications and returns the socket descriptor.

"socket()" creates a socket (a communications end point) by specifying the domain type, the protocol type and the protocol.

Only "AF_INET" (ARPA Internet protocol) can be specified for the domain type.

Either "SOCK_STREAM" or "SOCK_DGRAM" can be specified for the protocol type.

"SOCK_STREAM" supports reliable bit streams with no missing or duplicated data. In addition, communication becomes possible after the socket is connected, and "send()" and "recv()" can then be used to send and receive data, respectively.

"SOCK_DGRAM" supports non-connected, low-reliability datagram deliveries. This protocol provides no means for the sender to confirm that the recipient actually received the data, and is subject to data being missing, duplicated, or out of sequence. However, unlike "SOCK_STREAM," there is no overhead, since there is no connection procedure that must be followed when sending data.

```
#include "types.h"
#include "socket.h"

int socket(int domain, int type, int protocol)
```

Input

int domain;	Specifies the domain type. (Fixed to "AF_INET.")
int type;	Specifies the protocol type. Either "SOCK_STREAM" or "SOCK_DGRAM" can be specified.
int protocol;	Protocol specification (Specify "0.")

Return Value

0 or more:Normal end	Returns the socket descriptor.
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "errno."

Error

ENFILE	No more sockets of the protocol type in question can be created.
ENOTPROTOPT	The option is not recognized in the specified level.
EPROTONOSUPPORT	The specified protocol is not supported.
ENOBUFS	There is no buffer that can be used.
ENOPROTOOPT	The option is not recognized in the specified level.

sock_close

Closes a socket.

Description

This API closes the socket descriptor.

```
#include "types.h"
#include "socket.h"

int sock_close(int s)
```

Input

`int s;`	Socket descriptor

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ENOTSOCK`	Descriptor is not a socket.

sock_read

Reads data from a socket.

Description

This API reads data from a socket.

```
#include "types.h"
#include "socket.h"

int sock_read(int s, char *msg, int len)
```

Input

`int s;`	Socket descriptor
`char *msg;`	Read data buffer area
`int len;`	Read data buffer length

Output

`char *msg;`	Data that was read

Return Value

0:Normal end

-1:Abnormal end

In the event of an abnormal end, the error details are set in "`errno`."

Error

`EWOULDBLOCK`	The socket is not blocked, and no connection request has been received.
`ECONNRESET`	The reset signal was received.
`ENOTCONN`	The socket is not connected.
`ENOTSOCK`	Descriptor is not a socket.

sock_write

Writes data to a socket.

Description

This API writes data to a socket.

```
#include "types.h"
#include "socket.h"

int sock_write(int s, char *msg, int len)
```

Input

`int s;`	Socket descriptor
`char *msg;`	Address of area where write data is stored
`int len;`	Write data length

Return Value

0 or more:Normal end	Returns the number of bytes that were written.
-1:Abnormal end	

In the event of an abnormal end, the error details are set in "`errno`."

Error

`ECONNRESET`	The reset signal was received.
`ENOTCONN`	The reset signal was received.
`ENOTSOCK`	Descriptor is not a socket.

htonl

Converts the byte order.

Description

This API converts a host long integer into a network long integer.

```
#include "types.h"
#include "socket.h"

u_long htonl(u_long hostlong)
```

Input

`u_long hostlong;` Host long integer

Return Value

Network long integer

htons

Converts the byte order.

Description

This API converts a host short integer into a network short integer.

```
#include "types.h"
#include "socket.h"

u_short htons(u_short hostshort)
```

Input

`u_short hostshort;` Host short integer

Return Value

Network short integer

ntohl

Converts the byte order.

Description

This API converts a network long integer into a host long integer.

```
#include "types.h"
#include "socket.h"

u_long ntohl(u_long netlong)
```

Input

`u_long netlong;` Network long integer

Return Value

Host long integer

ntohs

Converts the byte order.

Description

This API converts a network short integer into a host short integer.

```
#include "types.h"
#include "socket.h"

u_short ntohs(u_short netshort)
```

Input

`u_short netshort;` Network short integer

Return Value

Host short integer

Sega Dreamcast

AVE-TCP 3.1 SDK Specifications Guide

Access Co., Ltd.

Notification Regarding Use

When using AVE-TCP® 3.1 SDK, please be aware of the following limitations regarding its use.

1. Copyrights and Other Rights

All rights inclusive of copyrights and industrial proprietary rights concerning the source file, object file and documents of this kit (AVE-TCP® 3.1 SDK) are the sole possession of Access Co., Ltd. At no time shall any of the proprietary rights of this kit be transferred to the customer. Furthermore, when beginning to use the kit, please be sure to complete the contract stating consent for usage rights immediately. If the contract is not completed, a request for return of the kit may be given.

2. Use of Kit

Customers shall use the kit in accordance with the usage rights consented by completion of the contract, concluded separately. In the case of using software developed with this kit in any hardware, a separate licensing contract with Access Co., Ltd. is required.

*Licensing conditions (example)

- Licensing fee
- Commitment
- Copyrights, trademark display
- Publicity cooperation
- Other . . .

3 Terms of Limitation

(1) Customer shall not perform any act that may possible cause damage to Access Co., Ltd., such as the reverse compiling, disassembly, reverse engineering, etc. of this kit or any other product.

(2) Customer shall not transfer this kit nor the consent for use provided the customer via the usage contract to any third party.

(3) Customer shall not disclose or make open to the public information concerning this kit (excluding information that has already been made public domain).

Sega Dreamcast

Table Of Contents

1. Preface

This guide explains the application programming interface (hereafter API) of AVE-TCP® 3.1, communication protocol software developed by Access Co., Ltd. for small-sized information machines.

AVE-TCP 3.1 has the following features:◦

1) Based on Request for Comments (RFC).
2) Described in C language and compatible with many CPUs.
3) Compatible to each type of operating system (OS).
4) High level of connectivity with each type of server, such as UNIX, NT, etc.
5) Compact, high performance.
6) Short turnaround time and excellent real-time tendency
7) ROM compatibility

Additionally, AVE-TCP 3.1 SKD API has the following features:

1) Realizes almost the same functions as socket.
2) Possible to ensure careful control throughout execution
3) Blocking/Non-Blocking compatibility
4) Ability to mount commands that realize suitable communication for real-time control

When Reading This Guide

This guide is prepared based on the premise that the customer already has an understanding of TCP/IP. Please refer to explanatory books available in the market for TCP/IP details.

Development Environment

Please refer to the AVE-TCP 3.1 SDK Transference Manual.

Debugging

Please refer to the AVE-TCP 3.1 SDK Transference Manual.

AVE-TCP3.1 SDK Data Type

The definitions for data types in AVE-TCP 3.1 are as follows. (Each data type is defined at the header file of each platform.)

1) Basic Data Types

```
AT_VOID             Null value
AT_SBYTE            1byte integer with code
AT_UBYTE            1byte integer without code
AT_SINT16           2byte integer with code
AT_UINT16           2byte integer without code
AT_SINT32           4byte integer with code
AT_UINT32           4byte integer without code
```

2) Network Byte Order Integer

AT_NINT16 16bit integer of network byte order

```
typedef union at_nint16 {
          AT_UINT16                     u16;
          AT_UBYTE                      b2;
} AT_NINT16;
```

AT_NINT32 32bit integer of network byte order

```
typedef union at_nint32 {
          AT_UINT32                     u32;
          AT_UBYTE                      b4;
} AT_NINT32;
```

Using AVE-TCP 3.1

1.0.1 Initialization

Format

```
AT_SINT16 AT_init(AT_VOID *initArg);
```

Call

The meaning of the parameter of the AT_init function is as follows.

```
initArg   Pointer for initialization option (not used presently)
```

Explanation

The following processes are executed using the AT_init function.

1) Acquires memory used by AVE-TCP.
2) Creates system resources such as semaphore, etc.
3) Creates and starts time task and receiving task.
4) Initializes each sub-module in AVE-TCP.
5) Initializes MIB variable.

1.0.2 Calling Each Command

Format

```
AT_SINT16 AT_apiCall(AT_UINT16 CommandCode, AT_VOID *InParamPtr,
AT_VOID
*OutParamPtr);
```

Call

The meaning of each parameter of the AT_api call function is as follows.

```
CommandCode          Command code assigned to each function
InParamPtr          Address of input parameter block
OutParamPtr         Address of output parameter block
```

Explanation

The module function mounted by AVE-TCP 3.1 is executed by noticing each command sent to the module through a common function (AT_apiCall). As long as there is no special designation, use of the function (AT_apiCall) is in the format mentioned above.

Note

- Please secure the input/output memory area of the parameter block from the calling side.
- Execution result is the return value for the `AT_apiCall` function. The meaning of the return value is different for each function.
- Please refer to the section on function details mentioned later.

Reference

The command code that can be designated by the `AT_apiCall` function is indicated in Table 1 Command Codes (next page).

1.0.3 End Process

Format

```
AT_SINT16 AT_disp(AT_VOID *dispArg);
```

Call

The meaning of the parameter of the `AT_disp` function is as follows.

```
dispArg             Pointer for release option (not used presently)
```

Explanation

The following processes are executed using the

`AT_disp function.`

1) Stops time task and receiving task.
2) Releases each sub-module in AVE-TCP.
3) Deletes system resources such semaphore.
4) Releases memory.

General/ Common Commands		Page
GETCONFIG	Acquires configuration block address	11
SETASR	Registers ASR routine	12
IFCONFIG	ets and starts network interface	13
IFDOWN	Stops network interface	14

TCP Command		Page
TCP_OPEN	Opens active TCP port	16
TCP_POPEN	Opens passive TCP port	17
TCP_ACCEPT	Receives connection request, assigns net handle	18
TCP_CLOSE	Closes connection	19
TCP_SHUTDOWN	Ends transmission	20
TCP_ABORT	Forces end of connection	21
TCP_GETADDR	Acquires connection address/port	22
TCP_STAT	Acquires opened connection's status	23
TCP_SEND	Transmits data	23
TCP_RECEIVE	Receives data	25
TCP_CANCEL	Cancels non-blocking call	27
TCP_SEND_F	Transmits data with user's buffer	28

UDP Commands		Page
UDP_OPEN	Opens UDP port	30
UDP_SEND	Transmits UDP	32
UDP_CLOSE	Closes UDP port	33

Route Control Command		Page
ROUTE_ADD	Registers route information	35
ROUTE_DEL	Deletes route information	37
ROUTE_LIST	Acquires route information	38

Table 1.1 ***Command Code List***

2. Blocking/Non-Blocking Calls

For AVE-TCP 3.1, blocking/non-blocking calls can be set for the operations executed by each the following commands: TCP_SEND, TCP_RECEIVE, TCP_ACCEPT and TCP_SEND_F. The switching of blocking/non-blocking is executed by changing the notifyaddr value in the input parameter block. A blocking call is realized by setting notifyaddr to null, and a non-blocking call is realized when an address is set.

In the case of a non-blocking call, control moves to notifyaddr at the time of command end. When the designated function (notify) is called by notifyaddr, the same return value as that for blocking call is stored in the first parameter (result) and net handle is stored in the second parameter (nh). (Therefore, it is possible to pass around one notifyaddr.)

The following, (1)-3)), lists the call formats for each command and notify call. Additionally, the blocking/non-blocking call operations are illustrated in Figs. 1-3.

Please refer to the notice (next page) regarding the recall and input of non-blocking calls.

1) Notify in TCP_ACCEPT

```
AT_VOID accept_notify(AT_SINT16 result, AT_SINT16 nh, AT_NINT32 dstaddr, AT_NINT16
srcport, AT_NINT16 dstport);
```

result	Execution result
nh	Net handle of accepted listen condition
dstaddr	Connection destination IP address
srcport	Connection destination port number
dstport	Connection destination port number (network order)

Please refer to 8. Return (Error) Code and Error Processing for details on results (execution results). Please refer to 5.3 TCP_ACCEPT for the details of each parameter and TCP_ACCEPT command. Please refer to the notice (next page) regarding the recall and input of non-blocking calls.

2) Notify in TCP_SEND

```
AT_VOID send_notify(AT_SINT16 result, AT_SINT16 nh);
```

result	Execution result
nh	Net handle that completes transmission

Please refer to 8. Return (Error) Code and Error Processing for details on results (execution results). Please refer to 5.9 TCP_SEND for the details of each parameter and TCP_SEND command. Please refer to notice (next page) regarding the recall and input of non-blocking calls.

3) Notify in TCP_RECEIVE

```
AT_VOID receive_notify(AT_SINT16 result, AT_SINT16 nh, AT_SINT16 reserve);
```

result	Execution result
nh	Net handle received
reserve	Unused area

Please refer to 8. Return (Error) Code and Error Processing for details on results (execution results). Please refer to 5.10 TCP_RECEIVE for the details of each parameter and TCP_RECEIVE command. Please refer to the notice (below) regarding the recall and input of non-blocking calls.

4) Notify in TCP_SEND

```
AT_VOID sendf_notify(AT_SINT16 result, AT_SINT16 nh, AT_UBYTE *buff);
```

result	Execution result
nh	Net handle that ends transmission
buff	Data address user designates when inputting TCP_SEND_F

Please refer to 8. Return (Error) Code and Error Processing for details on results (execution results). Please refer to 5.12 TCP_SEND_F for the details of each parameter and TCP_SEND_F command. Please refer to the notice (below) regarding the recall and input of non-blocking calls.

Note

- TCP/IP call cannot be issued from a notify call. If not returned immediately from the notify call, there is a possibility of an overflow of packet acceptance.
- One non-blocking call can be pending for each of the following commands: TCP_SEND, TCP_RECEIVE and TCP_ACCEPT. Regarding the TCP_SEND_F command, a maximum of 8 calls can be made simultaneously. In the case of making more than 9 calls at the same time, operation cannot be guaranteed. Additionally, the command call numbers are not saved in AVE-TCP 3.1. In the case of checking call numbers, please execute the process to count the number during command call time and notify call time on the application side.
- Non-blocking calls are automatically cancelled when the port is closed or at the time of abortion.

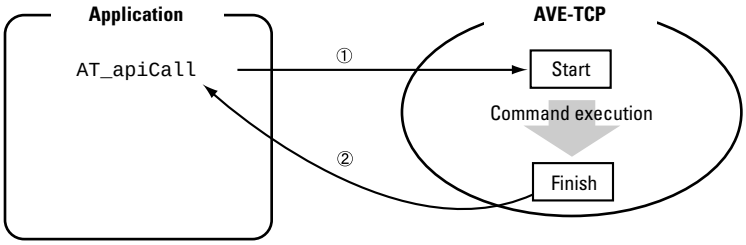

Figure 2.1 *Blocking Call Concept*

There are two methods in the case of non-blocking, shown as follows. The one to use depends on the method of communication exchange with the other party and the driver mounted in the network. The return values of (2) in Fig. 2 and (4) in Fig. 3 become `API_PENDING`(Oxffff). Regarding the content of operations, as some information is available in the explanation pages for each command, please make references there.

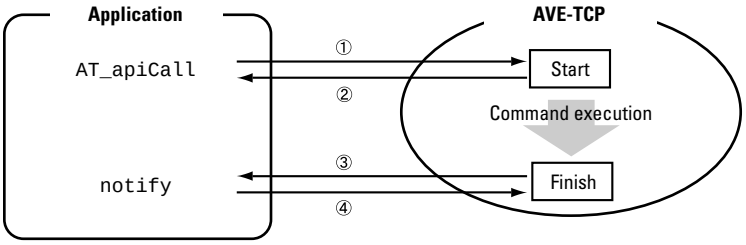

Figure 2.2 *Non-Blocking Call Concept - Pattern 1*

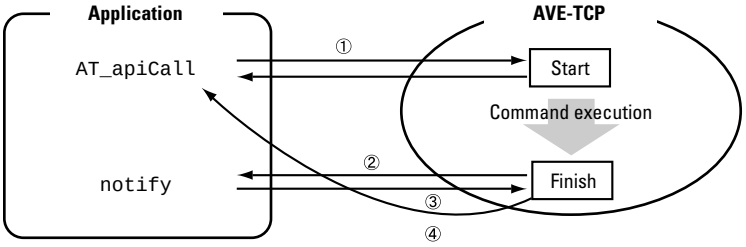

Figure 2.3 *Non-Blocking Call Concept - Pattern 2*

3. ASR

ASR enables a function to detect asynchronous events in AVE-TCP“ routines of an application. In AVE-TCP 3.1, an ASR routine can be requested using the SETASR command (refer to 4.2 SETASR) against a port that has been opened.

After an ASR routine has been registered against a port, it can detect if an asynchronous event for that port occurs. Only one ASR routine can be designated for each port.

The ASR routine is called using the following calling format.

```
AT_VOID asr(AT_SINT16 nh, AT_SINT16 event);

nh        Net handle
event     Event code
```

1.0.4 ASR Routine Event Codes

At the time an ASR routine is called from the AVE-TCP 3.1 module, the parameters of the ASR routine, event houses the event code (explained later) and nh houses the net handle in which the asynchronous event occurred.

Depending on the communication protocol used (TCP,UDP), the asynchronous event codes delivered when calling the ASR routine have the following meanings.

1) For TCP

Event code	Content	Label
1	Establishes connection (at opening)	ASR_ESTABLISHED
2	Receives FIN	ASR_FIN
4	Normal connection end	ASR_DISCONNECT
10	Receives RST	ASR_RST
11	Connection timeout occurs	ASR_TIMEOUT
12	NetWork/Host Unreachable	ASR_HOST
13	RST issued because old port remained at time of connection	ASR_PORT

(Note) Closure is necessary when connection ends abnormally.

2) For UDP

Event code	Content	Label
128	Network/Host Unreachable error when other party's address is designated	ASR_ADDR_UNREACH
129	Network/Host Unreachable error when other party's address and port are designated	ASR_ADDR_PORT_UNREACH

3) Common to TCP and UDP

Event code	Content	Label
255	Releases port	ASR_LAST

1.0.5 ASR Routine Use Notes

- Commands cannot be executed using the `AT_apiCall` function from an ASR routine. Additionally, if not returned immediately, there is a danger that packet overflow may occur.
- The ASR routine call is executed with every event occurrence, and there is no need to reset it. To cancel the ASR routine call, please use the SETASR command to designate `NULL` for the parameter housing the address of the ASR routine.
- In AVE-TCP 3.1, timeout processing is not executed except for timeout at the protocol level during communication. If timeout processing is necessary, please execute it using non-blocking call in the upper-level program and the `TCP_ABORT` command.

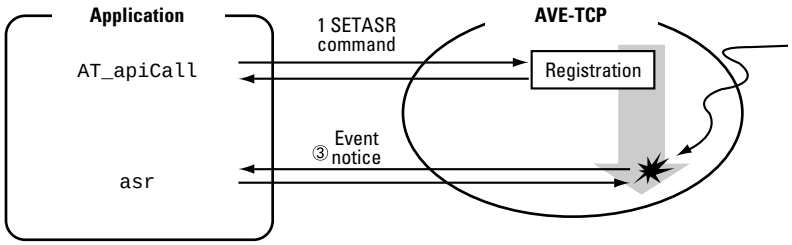

Figure 2.4 *ASR Concept*

General/Common Commands

The commands used for general and common purposes (settings, configuration information acquisition, etc.) are explained in the following.

General/Common Command		Page
GETCONFIG	Acquires configuration block address	11
SETASR	Registers ASR routine	12
IFCONFIG	Sets and starts network interface	13
IFDOWN	Stops network interface	14

Table 2.1 ***General/Common Command List***

1.0.6 GETCONFIG

Acquisition of configuration block address

Function

```
AT_SINT16 AT_apiCall((AT_UINT16)GETCONFIG,
            (AT_VOID* )NULL, (AT_VOID* )OutParamPtr)
```

Call

	Label	Value
Command Code	GETCONFIG	0x4101
InParamPtr	NULL	
OutParamPtr	AT_configptr configuration address	

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	Error from network driver	API_GENERAL

Output Parameter Block

```
typedef struct _AT_configptr {
          AT_VOID                                    *config_ptr;/* Configuration block
address */
} AT_configptr;
```

Configuration Block Data Structure

```
typedef struct _AT_config {
          AT_NINT32                                  ip_addr;/* IP address (network order)
*/
          AT_NINT32                                  netmask;/* Subnet mask */
          AT_NINT32                                  broadcast;/* Broadcast address */
} AT_config;
```

Explanation

Acquires the configuration block address loaded in the AVE-TCP 3.1 module, houses it in OutParamPtr, and returns it.

Acquires the address of the memory block housing the configuration parameter loaded in the AVE-TCP 3.1 module. It is only possible to refer to the memory block acquired by the `GETCONFIG` command, and the memory block value cannot be changed.

Note

- In the expression mentioned above, the layout on the memory of the designated data of `AT_NINT32` and `AT_NINT16` become Big Indian.
- In the case of changing the value in memory block with the `GETCONFIG` command, please be aware that the operation is not guaranteed.

1.0.7 SETASR

Registration of ASR routine

Format

```
AT_apiCall(SETASR, InParamPtr, NULL)
```

Call

	Label	Value
CommandCode	SETASR	0x4102
`InParamPtr`	`AT_setasr` configuration address	
`OutParamPtr`	`NULL`	

Input Parameter Block

```
typedef struct _AT_setasr {
         AT_SINT16                              nh;       /* Net handle */
         AT_VOID                                   (*asraddr)(AT_SINT16, AT_SINT16);
                                                      /* ASR routine address *//
} AT_setasr;
```

Return value	Return code	Content	Label
	0	Normal end	`SUCCESS`
	0xfffc	Net handle abnormal	`API_INVALIDHANDLE`

Explanation

SETASR command resets/cancels the ASR routine registered by `UDP_OPEN` or `TCP_OPEN` commands. By designating `NULL` to asraddr, ASR routine setup can be cancelled (to stop ASR routine call against that port).

Reference

Please refer to 3. ASR (pg. 8).

1.0.8 IFCONFIG

Setting and starting network interface

Format

```
AT_apiCall(IFCONFIG, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	IFCONFIG	0x4105
InParamPtr	AT_ifconfigparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_ifconfigparam {
        AT_SINT16                       if_num;/* Interface number(supports 0
only) */
        AT_NINT32                       ip_addr;/* IP address */
        AT_NINT32                       netmask;/* Subnet mask */
        AT_NINT32                       broadcast;/* Broadcast address */
} AT_ifconfigparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	Address abnormal, parameter abnormal or interface has started already	API_GENERAL

Explanation

Executes configuration of designated network interface and starts network interface.

Network interface supports interface number 0 only.

Note

- When using the IFCONFIG command, the network interface must be stopped. It is not possible to call the IFCONFIG command in duplicate. If called in duplicate, a general error (0xfff0) is returned.
- In the case of changing each parameter of the AT_ifconfigparam configuration, please use the IFDWON command to close all ports and stop the network interface.

1.0.9 IFDOWN

Stopping network interface

Format

```
AT_apiCall(IFDOWN, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	IFDOWN	0x4106
InParamPtr	AT_ifconfigdownparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_ifconfigdownparam {
          AT_SINT16      if_num;     /* Interface number(supports 0 only) */
} AT_ifconfigdownparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	Parameter abnormal	API_GENERAL

Explanation

Executes stopping of network interface.

Network interface supports interface number 0 only.

Note

- If the following processes are not executed before using the IFDOWN command, operation after using the command is not guaranteed.
- Closing of TCP and UDP ports
- Deletion (ROUTE_DEL command) of route information added from the application side (ROUTE_ADD command).

TCP Commands

The following is a description of the commands used in TCP.

Table 2.2 ***TCP Command List***

1.0.10 TCP_OPEN

Open active TCP port

Format

```
AT_apiCall(TCP_OPEN, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_OPEN	0x4110
InParamPtr	AT_openparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_openparam {
         AT_NINT32         youraddr;       /* Other party's IP address */
         AT_NINT16         yourport;       /* Other party's port number */
         AT_NINT16         myport;         /* Personal port number */
         AT_UBYTE                      reserve1;   /* Unused area */
         AT_UBYTE                      ttl;            /* Time to live */
         AT_UBYTE                      svctype;    /* Service type */
         AT_UBYTE                      df_flag;    /* Don't Fragment flag */
         AT_SINT16         reserve2;       /* Unused area */
         AT_VOID                       reserve3;   /* Unused area */
         AT_VOID                       (*asraddr)(AT_SINT16, AT_SINT16);
                                                        /* ASR routine address */
} AT_openparam;
```

Return value	Return code	Content	Label
	0xfffe	No network handle	API_PORTFULL
	0xfffd	Duplicate socket error	API_DUPSOCK
	0xfffa	Connection cut	API_CONNRESET
	0xfff9	Indefinite route	API_UNREACHABLE
	mentioned above	Net handle opened	-

Explanation

Opens active TCP port.

Executes TCP_OPEN command to secure port and transmit SYN packet to end. Please execute confirmation of established connection using ASR routine call or TCP_STAT command.

IP address and port number are designated by Big Indian.

When myport is set to 0, unused things in 1024-2047 are assigned automatically. When ttl is set to 0, the default value is used. The default value is 30.

df_flag designates prohibited fragment flag for the IP header. A designation of 1 prohibits fragments. Additionally, df_flag is designated 0 normally.

Service Type is ignored by the current implementation, so please do not use it.

Note

- Please designate 0 to reserve 1 and reserve 2, and NULL to reserve 3 in the AT_openparam configuration

1.0.11 TCP_POPEN

Open passive TCP port

Format

```
AT_apiCall(TCP_POPEN, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_POPEN	0x4111
InParamPtr	AT_popenparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_popenparam {
        AT_NINT32                       youraddr;/* Other party's IP address */
        AT_NINT16                       yourport;/* Other party's port number */
        AT_NINT16                       myport;  /* Personal port number */
        AT_UBYTE                                reserve1;/* Unused area */
        AT_UBYTE                                ttl;   /* Time to live */
        AT_UBYTE                                svctype;/* Service type */
        AT_UBYTE                                df_flag;/* Don't Fragment flag */
        AT_SINT16                       reserve2;/* Unused area */
        AT_VOID                                 reserve3;/* Unused area */
} AT_popenparam;
```

Return value	Return code	Content	Label
	0xfffe	No network handle	API_PORTFULL
	0xfffd	Duplicate socket error	API_DUPSOCK
	0xfff9	Indefinite route	API_UNREACHABLE
	Other than mentioned above	Net handle opened	-

Explanation

Opens passive TCP port.

Executes TCP_POPEN command to secure port in LISTEN condition. This is equivalent to socket+bind+listen in UNIX.

IP address and port number are designated by network order.

When myport is set to 0, unused things in 1024-2047 are assigned.

When 0 is designated for youraddr and yourport, connection requests from each optional address and port can be accepted.

0 is designated to ttl, and the default value is used. The default value is 30.

df_flag designates prohibited fragment flag for the IP header. A designation of 1 prohibits fragments. 0 is normally designated.

Service Type is ignored by the current implementation.x

Note

- Please designate 0 to reserve 1 and reserve 2, and NULL to reserve 3 in the AT_popenparam configuration.

1.0.12 TCP_ACCEPT

Reception of connection request, assignment of net handle

Format

```
AT_apiCall(TCP_ACCEPT, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_ACCEPT	0x4112
InParamPtr	AT_acceptparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_acceptparam {
          AT_SINT16         nh;                    /* Net handle */
          AT_VOID                    (*notifyaddr)(AT_SINT16, AT_SINT16,
                                                    AT_NINT32, AT_NINT16);
                                                       /* Notify routine address */
          AT_VOID                    (*asraddr)(AT_SINT16, AT_SINT16);
                                                       /* ASR routine address */
} AT_acceptparam;
```

Return value	Return code	Content	Label
	0xffff	Command pending (not an error)	API_PENDING
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	Other than mentioned above	Net handle accepted	-

Explanation

Accepts TCP_ACCEPT connection request and assigns new net handle.

This is equivalent to accept in UNIX.

asraddr is handled as a newly assigned net handle object.

In the AVE-TCP 3.1 module, after the TCP_ACCEPT command has been executed and the SYN packet is received, a net handle is assigned.

Verify that connection has been established by using the ASR function or the TCP_STAT command.

Reference

1) Please refer to ⌈2. Blocking/Non-Blocking⌋ Calls regarding notifyadd.
2) Please refer to ⌈3. ASR regarding⌋ regarding ASR routine.

1.0.13 TCP_CLOSE

Close connection.

Format

```
AT_apiCall(TCP_CLOSE, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_CLOSE	0x4113
InParamPtr	AT_nhparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_nhparam {
         AT_SINT16                              nh;    /* Net handle */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE

Explanation

Closes the connections of ports designated by the net handle.

Calling is returned when FIN packet transmission is cueing in the AVE-TCP 3.1 module. In the case that it is possible to send back the packet, the call of the TOP_CLOSE command is sent back immediately. If the ASR routine call is set, the ASR routine call will be executed when the connection ends. Access to the port is prohibited excluding the ASR routine call, pending calls are cancelled and data being received/transmitted are trashed.

1.0.14 TCP_SHUTDOWN

End transmission

Format

```
AT_apiCall(TCP_SHUTDOWN, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_SHUTDOWN	0x4114
InParamPtr	AT_nhparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_nhparam {
        AT_SINT16                                nh;    /* Net handle */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE

Explanation

Ends transmission to the port connection designated by the net handle.

The same as TCP_CLOSE it returns when the FIN packet is sent, however, the port is not released until the TCP_CLOSE command is executed. It is necessary to issue the TCP_CLOSE command when the connection has ended. Pending calls are cancelled and data being transmitted trashed when the TCP_SHUTDOWN command is executed.

1.0.15 TCP_ABORT

Forced end of connection

Format

```
AT_apiCall(TCP_ADORT, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_ABORT	0x4115
InParamPtr	AT_nhparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_nhparam {
        AT_SINT16                                nh;    /* Net handle */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE

Explanation

TCP_ABORT sends an RST packet forcing the connection to end. The port designated by the net handle is released at that time.

1.0.16 TCP_GETADDR

Connection address/port acquisition

Format

```
AT_apiCall(TCP_GETADDR, InParamPtr, OutParamPtr)
```

Call

	Label	Value
Command Code	TCP_GETADDR	0x4116
InParamPtr	AT_nhparam configuration address	
OutParamPtr	AT_getaddr_r configuration address	

Input Parameter Block

```
typedef struct _AT_nhparam {
         AT_SINT16                              nh;    /* Net handle */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE

Output Parameter Block

```
typedef struct _AT_getaddr_r {
         AT_NINT32                          ipaddr;     /* Other party's IP address */
         AT_NINT16                          myport;  /* Personal port number */
         AT_NINT16                          yourport;/* Other party's port number */
} AT_getaddr_r;
```

Explanation

Acquires the IP address and port number of the connection destination designated by the net handle.

1.0.17 TCP_STAT

Opened connection's status acquisition

Format

```
AT_apiCall(TCP_STAT, InParamPtr, OutParamPtr)
```

Call

	Label	Value
Command Code	TCP_STAT	0x4117
InParamPtr	AT_nhparam configuration address	
OutParamPtr	AT_stat_r configuration address	

Input Parameter Block

```
typedef struct _AT_nhparam {
          AT_SINT16                                nh;    /* Net handle */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	0xfffa	Connection cut	API_CONNRESET
	0xfff9	Indefinite route	API_UNREACHABLE
	0xfff5	Connection timeout	API_TIMEOUT

Output Parameter Block

```
typedef struct _AT_stat_r {
          AT_UINT16                              stat;  /* Connection condition */
          AT_UINT16                              reserve;/* Unused area */
          AT_UINT16                              sendwin;/* Residual volume of
transmission window */
          AT_UINT16                              recvwin;/* Volume of data already
received */
} AT_stat_r;
```

stat (connection condition)

Return code	Label	
0		CLOSED
1		LISTEN
2		SYN_SENT
3		SYN_RECEIVED
4		ESTABLISHED
5		CLOSE_WAIT
6		FIN_WAIT_1
7		CLOSING
8		LAST_ACK
9		FIN_WAIT_2
10		TIME_WAIT

Explanation

Acquires the status of opened connections designated by the net handle.

1.0.18 TCP_SEND

Data transmission

Format

```
AT_apiCall(TCP_SEND, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_SEND	0x4118
InParamPtr	AT_sendparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_sendparam {
          AT_SINT16                                   nh;       /* Net handle */
          AT_VOID                                        (*notifyaddr)(AT_SINT16,
AT_SINT16);
                                                                        /* Notify routine
address */
          AT_UBYTE                                       reserve;/* Unused area */
          AT_UBYTE                                       bufnum;/* Using buffer number */
          struct {
                                                      AT_SINT16len; /* Data length */
                                                      AT_UBYTE*buf; /* Data address */
          } buffersn;                                           /* n is 0~3 */
} AT_sendparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xffff	Command pending(not an error)	API_PENDING
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	0xfff6	Port closed already	API_PORTCLOSED
	0xfff4	Buffer could not be secured	API_NOBUF
	0xfff3	Parameter abnormal	API_BADOPT

Explanation

Transmits data to TCP connection designated by the net handle. The TCP_SEND command ends when the data is housed in the transmission window.

It is not possible for plural non-blocking calls to be pending when using the TCP_SEND command.

Please use the TCP_SEND command under the condition of when the connection is established by the ASR routine call or routine call or TCP_STAT command.

Note

- Please designate 0 for the reserve reserve of the AT_sendparam configuration. Please designate 0 for the reserve of the AT_sendparam configuration.
- Please designate 0~3 for n of the AT_sendparam configuration.

Reference

Please refer to 2. Blocking/Non-Blocking Calls regarding Calls regarding notifyadd

1.0.19 TCP_RECEIVE

Data reception

Format

```
AT_apiCall(TCP_RECEIVE, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_RECEIVE	0x4119
InParamPtr	AT_rcvparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_rcvparam {
          AT_SINT16                                nh;       /* Net handle */
          AT_SINT16                                len;      /* Data length */
          AT_UBYTE                                    *buf;     /* Data address */
          AT_VOID
(*notifyaddr)(AT_SINT16,AT_SINT16,AT_SINT16);
                                                                        /* Notify routine
address */
} AT_rcvparam;
Return valueReturn code                    Content       Label
          Positive number                  Normal end, receiving byte number-
          0(blocking call time)            Receiving FIN SUCCESS
          0xffff                           Command pending(not an error)API_PENDING
          0xfffc                           Net handle abnormalAPI_INVALIDHANDLE
          0xfff8                           Receiving FIN API_RECVFIN
          0xfff6                           Port closed alreadyAPI_PORTCLOSED
```

Explanation

Receives data from TCP connection designated by the net handle.

In the TCP_RECEIVE command, the notify routine can be registered, however, it is not possible to have plural non-blocking calls pending.

Please use the TCP_RECEIVE command under the condition of when the connection is confirmed by the ASR routine call or TCP_STAT command.

Note

- In the case of using the TCP_RECEIVE command as a non-blocking call, the return value always becomes API_PENDING(0xffff). In the case of using the TCP_RECEIVE command as a non-blocking call, the return value always becomes API_PENDDING (0xffff).

Reference

Please also refer to 2. Blocking/Non-Blocking Calls regarding notifyadd

1.0.20 TCP_CANCEL

Cancellation of non-blocking call

Format

```
AT_apiCall(TCP_CANCEL, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_CANCEL	0x411a
InParamPtr	AT_cancelparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_cancelparam {
         AT_SINT16                              nh;    /* Net handle */
         AT_SINT16                              what;  /* Cancel object */
} AT_cancelparam;
```

Object to be cancelled, 1:SEND, 2:RECEIVE, 4:ACCEPT is designated by OR. is designated by

OR.

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	0xfff3	Parameter abnormal	API_BADOPT

Explanation

Cancels non-blocking calls registered to designated connection by the net handle.

Non-blocking calls registered by the commands TCP_SEND, TCP_RECEIVE, and TCP_ACCEPT are designated by 1:SEND, 2:RECEIVE, 4:ACCEPT, respectively.

1.0.21 TCP_SEND_F

Data transmission with user's buffer

Format

```
AT_apiCall(TCP_SEND_F, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	TCP_SEND_F	0x411b
InParamPtr	AT_sendfparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_sendfparam {
          AT_SINT16                                  nh;       /* Net handle */
          AT_VOID
(*notifyaddr)(AT_SINT16,AT_SINT16,AT_UBYTE *);
                                                                  /* notify routine
address */
          AT_UBYTE                                     reserve;/* Unused area */
          AT_SINT16                                  len;      /* Data length */
          AT_UBYTE                                     *buf;     /* Data address */
} AT_sendfparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xffff	Command pending(not an error)	API_PENDING
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	0xfff6	Port closed already	API_PORTCLOSED
	0xfff3	Parameter abnormal	API_BADOPT

Explanation

Transmits data to the TCP connection via the user's buffer. The command call ends when the ACK packet is returned against all data in the buffer. In other words, the buffer given by the TCP_SEND_F command call is used as the window buffer as is. Therefore, it is possible to increase the transmission window size.

However, please do not use this command when the data length is short, as it reduces the window size (reverse), thus reducing efficiency.

It is possible to execute plural non-blocking calls simultaneously when using the TCP_SEND_F command. However, in the case of a pending TCP_SEND_F command, the calls cannot be executed until the command ends.

Please use the TCP_SEND_F command under the condition of when the connection is established by the ASR routine call or TCP_STAT command.

Note

- Since the buffer is used as the window buffer, this command cannot be cancelled.
- Please designated 0 for the reserve of the AT_sendfparam configuration.

UDP Command

The following is a description of the commands used in UDP.

UDP Command		Page
UDP_OPEN	Opens UDP port	30
UDP_SEND	Transmits UDP	32
UDP_CLOSE	Closes UDP port	33

Table 2.3 ***UDP Command List***

1.0.22 UDP_OPEN

Opens UDP port

Format

```
AT_apiCall(UDP_OPEN, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	UDP_OPEN	0x4120
InParamPtr	AT_uopenparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_uopenparam {
         AT_NINT32             youraddr;           /* Other party's IP address */
         AT_NINT16             yourport;           /* Other party's port number */
         AT_NINT16             myport;             /* Personal port number */
         AT_UBYTE                                  reserve1;       /* Unused area */
         AT_UBYTE                                  ttl;                 /* Time to live */
         AT_UBYTE                                  svctype;        /* Service type */
         AT_UBYTE                                  df_flag;        /* Don't Fragment flag */
         AT_SINT16             reserve2;           /* Unused area */
         AT_VOID                                   reserve3;       /* Unused area */
         AT_VOID                                   (*asraddr)(AT_SINT16, AT_SINT16);
                                                                        /* ASR routine
address */
         AT_SINT16             (*receiver)(AT_SINT16, AT_UDPPRM);
                                                   /* Receiver routine address */
} AT_uopenparam;
```

Return value	Return code	Content	Label
	0xfffe	Net handle abnormal	API_PORTFULL
	0xfffd	Duplicate socket error	API_DUPSOCK
	Other	Opened net handle	-

Receiver call

```
typedef struct _AT_UDPPRM {
          AT_SINT16                              handle;   /* Net handle */
          AT_VOID                                   *buf;     /* Data address */
          AT_SINT16                              len;      /* Data length */
          AT_NINT32                              srcaddr; /* Transmission origin IP
address */
          AT_NINT16                              srcport; /* Transmission origin port
number */
          AT_UBYTE                                  *usrbuff; /* Buffer address */
} AT_UDPPRM;

AT_SINT16 receiver(AT_SINT16 function, UDPPRM *param);
```

Here, in the case the function is 0, the call is for securing a buffer; and if the function is 1, the call is for delivering a datagram. The block address of the input parameter delivered is set such that net handle goes to handle, the lead address of receiving data goes to buf, its length goes to len, the transmission origin IP address goes to srcaddr, and the transmission origin port number goes to scrport. For the receiver, in the case of receiving, 0 is returned as the return value and the received datagram is set up in the lead address of the transfer area of the usrbuff to return control.

Explanation

Opens the UDP port. After the port is opened successfully, the receiver routine is called when a datagram is received.

The IP address and port number are designated by network order.

If 0 is designated for youraddr, packets can be received from optional hosts.

If 0 is designated for myport, unused things are assigned to 1024~2047.

Multiple designations for myport cannot be made.

For things less than ttl, it is the same as in the case of TCP.

Note

- Please designate 0 for reserve 1 and reserve 2, and `NULL` for reserve 3 of the `AT_uopenparam` configuration.

Datagram Buffering Using UDP Protocol

In the AVE-TCP 3.1 module, datagram buffer is not executed inside the module when using UDP. The receiver is registered when the port is open and, when it arrives, the datagram is delivered immediately to the upper-level program using the following process.

1) After protocol processing ends, the datagram length and necessary information for choosing if the datagram should be received are delivered to the receiver routine.
2) The receiver routine judges if the datagram should be received and secures a buffer with the necessary length, and returns control to the AVE-TCP 3.1 module. At this time, in the case of receiving, place the lead address of the buffer in the usrbuff of the parameter block, and make the return value 0.

   In the case of stopping reception, make the return value non-zero.

   In the case of acquiring part of a packet, input the required length for reception into len of the parameter block and return. In this case, the remainder of the packet is trashed.

   - The AVE-TCP 3.1 module copies the datagram to the secured buffer, and recalls the receiver routine. At this time, the datagram in the module is cancelled.

As the receiver routine is assigned by calling a processing part of the AVE-TCP 3.1 module, it cannot issue an AVE-TCP 3.1 command. Additionally, there is a danger of packet overflow occurring if it does not return immediately.

1.0.23 UDP_SEND

UDP transmission

Format

```
AT_apiCall(UDP_SEND, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	UDP_SEND	0x4121
InParamPtr	AT_usendparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_usendparam {
          AT_SINT16                              nh;    /* Net handle */
          AT_NINT32                              youraddr;/* Other party's IP
address */
          AT_NINT16                              yourport;/* Other party's port
number */
          AT_UBYTE                                  bufnum;/* Buffer number used */
          struct {
                                                 AT_SINT16len; /* Data length */
                                                 AT_UBYTE*buf; /* Data address */
          } buffersn;                                    /* n is 0~3 */
} AT_usendparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	Net handle abnormal, incorrect parameter or incorrect designated address	API_GENERAL

Explanation

Transmits UDP datagram.

For youraddr and yourport, the one designated by the UDP_OPEN command is given priority.

Note

- Please designate 0~3 to n for the AT_usendparam configuration.

1.0.24 UDP_CLOSE

Closes UDP port

Format

```
AT_apiCall(UDP_CLOSE, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	UDP_CLOSE	0x4113
InParamPtr	AT_nhparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_nhparam {
         AT_SINT16 nh;                          /*  Net handle  */
} AT_nhparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfffc	Net handle abnormal	API_INVALIDHANDLE
	Other than mentioned above	Refer to code table	-

Explanation

Closes the UDP port designated by the net handle.

For CommandCode, it designates the same thing as for TCP_CLOSE.

Route Control Commands

The following is a description of the commands used for route control.

Route Control Command		Page
ROUTE_ADD	Registers route information	35
ROUTE_DEL	Deletes route information	37
ROUTE_LIST	Acquires route information	38

Table 2.4 ***Route Control Command List***

1.0.25 ROUTE_ADD

Route information registration

Format

```
AT_apiCall(ROUTE_ADD, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	ROUTE_ADD	0x4130
InParamPtr	AT_addrtparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_addrtparam {
         AT_NINT32                          dstaddr;/* Other party's IP address */
         AT_NINT32                          gtwayaddr;/* Gateway address */
         AT_SINT16                          reserve;/* Unused area */
} AT_addrtparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	Registration incorrect (in case of table overflow, etc.)	API_GENERAL

Explanation

Registers route information.

The ROUTE_ADD command registers the network route information in the AVE-TCP 3.1 module.

The subnet value is calculated from dstaddr.

In the case of dstaddr being designated against the network, designated the host field as 0. In the case the host field is a value other than 0, dstaddr is handled as routing against the host.

When 0 is designated to dstaddr, it becomes the default gateway destination.

Note

- Please designate 1 to the reserve of the `AT_addrtparam` configuration.

In the route information registered by the `ROUTE_ADD` command, the route information regarding the network to which the user belongs is handled as a subnet extension form.

For example,

Network address:	192.168.0.0
Netmask:	255.255.255.0

In a class C network as mentioned above, the case of a subnet extension to four networks is assumed as follows.

Netmask:	255.255.255.4
Network A:	192.168.0.0
Network B:	192.168.0.64
Network C:	192.168.0.128
Network D:	192.168.0.192

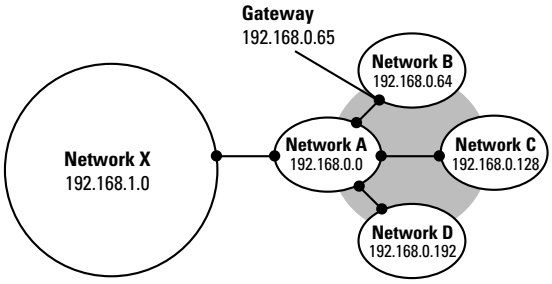

Figure 2.5 *Route Calculation by Subnet*

Given that the gate address of network B is 192.168.0.65, as the route from the AVE-TCP host in network B to network A is the route in 192.168.0.0, the mask value of 255.255.255.4 is used for route calculation.

On the other hand, when calculating the route from the AVE-TCP host in network B to network X, which is outside of the subnet, the default mask value of 255.255.255.0 is used.

1.0.26 ROUTE_DEL

Deletes route information

Format

```
AT_apiCall(ROUTE_DEL, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	`ROUTE_DEL`	0x4131
`InParamPtr`	`AT_delrtparam` configuration address	
`OutParamPtr`	`NULL`	

Input Parameter Block

```
typedef struct _AT_delrtparam {
         AT_NINT32                              dstaddr;/* Other party's IP address */
         AT_NINT32                              gtwayaddr;/* Gateway address */
} AT_delrtparam;
```

Return value	Return code	Content	Label
	0	Normal end	SUCCESS
	0xfff0	No entry	API_GENERAL

Explanation

Deletes route information.

Designates the IP address of the party for deletion to dstaddr and the gateway address of the party to gtwayaddr.

Additionally, designates parameters like the following in order to delete the default gateway address.

dstaddr	= 0
gtwyaddr	= gateway address at time of registration

Note

- It is necessary for dstaddr and gtwayaddr to be in accordance with the information acquired by the ROUTE_LIST command.

1.0.27 ROUTE_LIST

Route information acquisition

Format

```
AT_apiCall(ROUTE_LIST, InParamPtr, NULL)
```

Call

	Label	Value
Command Code	ROUTE_LIST	0x4132
InParamPtr	AT_lstrtparam configuration address	
OutParamPtr	NULL	

Input Parameter Block

```
typedef struct _AT_lstrtparam {
          AT_SINT16                              len;   /* Data length */
          AT_RouteEntry                          *buf;  /* Data address */
} AT_lstrtparam;
typedef struct _AT_RouteEntry {
          AT_NINT32                         dstaddr;     /* Other party's IP address */
          AT_NINT32                         gtwayaddr;   /* Gateway address */
          AT_UINT16                         reserve1;    /* Unused area */
          AT_UINT16                         reserve2;    /* Unused area */
          AT_UINT16                         reserve3;    /* Unused area */
} AT_RouteEntry;
```

Return Value

Acquired entry number

Explanation

Acquires route information.

The memory block acquired by the ROUTE_LIST command is only for reference; it is not possible to change values in the block.

Note

- In the case of no route information, 0 is returned instead of an error message.
- If values in the memory block are changed by the ROUTE_LIST command, please be aware that operation is not guaranteed.

Return (Error) Code and Error Processing

1.0.28 Return (Error) Code List

Return code	Content	Label
0	Normal end	SUCCESS
0xffff	Command pending(not an error)	API_PENDING
0xfffe	No network handle	API_PORTFULL
0xfffd	Duplicate socket error	API_DUPSOCK
0xfffc	Net handle abnormal	API_INVALIDHANDLE
0xfffb	Transmission pending	API_SENDPENDING
0xfffa	Connection cut	API_CONNRESET
0xfff9	Indefinite route	API_UNREACHABLE
0xfff8	Data reception end (FIN received)	API_RECVFIN
0xfff7	Reception pending	API_RECVPENDING
0xfff6	Port already closed and shutdown	API_PORTCLOSED
0xfff5	Connection timeout	API_TIMEOUT
0xfff4	Designated buffer could not be secured	API_NOBUF
0xfff3	Designated parameter abnormal	API_BADOPT
0xfff0	General error	API_GENERAL

1.0.29 Corresponding Table for Each Command and Error

Command / Return code	General/ Common				TCP												UDP			Route Control		
	GETCONFIG	SETASR	IFCONFIG	IFDOWN	TCP_OPEN	TCP_POPEN	TCP_ACCEPT	TCP_CLOSE	TCP_SHUTDOWN	TCP_ABORT	TCP_GETADDR	TCP_STAT	TCP_SEND	TCP_RECEIVE	TCP_CANCEL	TCP_SEND_F	UDP_OPEN	UDP_SEND	UDP_CLOSE	ROUTE_ADD	ROUTE_DEL	ROUTE_LIST
0	○	○	○	○				○	○	○	○	○	○			○		○	○	○	○	
0xffff							○						○	○		○						
0xfffe					○	○											○					
0xfffd					○	○											○					
0xfffc		○					○	○	○	○	○	○	○	○	○	○			○			
0xfffb																						
0xfffa					○							○										
0xfff9					○	○						○										
0xfff8														○								
0xfff7																						
0xfff6													○	○		○						
0xfff5												○										
0xfff4													○								I	
0xfff3													○		○	○						
0xfff0	○		○	○														○		○	○	

1.0.30 Error Processing

How to Read

Cause	Indicates cause of error occurrence.
Countermeasure	Indicates corresponding measure to be taken.
Retry	The following symbols indicate whether or not command execution can be reattempted in the event of an error. O: Indicates the possibility to execute again as the error that occurred was temporary, such as connection conditions, etc. x:Indicates that it is impossible to execute again as the error originated from user's application mounting, etc.

0xffff — Command pending (not an error)

Cause	Accepted a non-blocking call.
Countermeasure	No countermeasure required as it is not an error. No countermeasure required as it is not an error.
Retry	Please do not retry.

0xfffe — No network handle

Cause	Exceeded the limited number of connections.
Countermeasure	After closing an unused socket using `TCP_CLOSE` or `UDP_CLOSE` commands, please execute reconnection using `TCP_OPEN` or `TCP_POPEN` commands.
Retry	O

0xfffd — Duplicate socket error

Cause	Exceeded the limited number of connections.
Countermeasure	Duplicate socket error
Retry	x

0xfffc — Net handle abnormal

Cause 1	Attempted execution of transmission/reception using an unopened net handle in TCP or UDP.
Countermeasure	Check the user application again.
Retry	x
Cause 2	RST reception forced connection to end.
Countermeasure	After disconnecting the connection using the `TCP_CLOSE` command, please execute reconnection using the `TCP_OPEN` or `TCP_POPEN` commands.
Retry	O

0xfffa	Connection cut	
	Cause	Forced end to connection due to reception of RST from other party. (`TCP_RECEIVE` is issued after receiving RST.)
	Countermeasure	After disconnecting the connection using the `TCP_CLOSE` command, please execute reconnection using the `TCP_OPEN` or `TCP_POPEN` commands.
	Retry	O
0xfff9	Indefinite route Indefinite route	
	Cause	Could not locate route.
	Countermeasure	Please check to see if the command parameters and route information are correct.
	Retry	x
0xfff8	Data reception end (FIN received)	
	Cause	Received FIN from other party.
	Countermeasure	After disconnecting the connection using the After disconnecting the connection using the `TCP_CLOSE` command, please execute reconnection using the `TCP_OPEN` or `TCP_POPEN` commands.
	Retry	O
0xfff6	Port already closed and shutdown	
	Cause	Command issued for socket ID of a connection already closed by the `TCP_CLOSE` command. Command issued for socket ID of a connection already closed by the `TCP_CLOSE` command.
	Countermeasure	Please execute reconnection using the `TCP_OPEN` or `TCP_POPEN` commands.
	Retry	O
0xfff5	Connection timeout	
	Cause	No response from other party.
	Countermeasure	After disconnecting the connection using the `TCP_CLOSE` command, please execute reconnection using the `TCP_OPEN` or `TCP_POPEN` commands. If communication does not recover even after trying to reconnect, please confirm that there are no abnormalities with connectors and cables or problems on the side of the other party.
	Retry	O

0xfff4	Designated buffer could not be secured	
	Cause	Exceeded limitations for buffer array length.
	Countermeasure	It is necessary to check the user application. Please designate the buffer array length to 0~3.
	Retry	x
0xfff3	Designated parameter abnormal	
	Cause	Abnormal parameter is designated for AVE-TCP 3.1
	Countermeasure	It is necessary to check the user application.
	Retry	x
0xfff0	General error	
	`GETCONFIG`	
	Cause	Network driver error is returned.
	Countermeasure	If the network interface was initiated incorrectly, please start it again. If the problem continues, there is a possibility that there may be a problem with the mounting of the network driver, thus it is necessary to check the user application and/or the network driver.
	Retry	x
	`IFCONFIG`	
	Cause 1	Abnormal parameter is designated.
	Countermeasure	It is necessary to check the user application.
	Retry	x
	Cause 2	Network interface has started already.
	Countermeasure	After stopping the network interface using the `IFDOWN` command, please execute the `IFCONFIG` command to restart it.
	Retry	x
	`IFDOWN`	
	Cause	A number other than 0 was designated for the interface number.
	Countermeasure	It is necessary to check the user application. Please designate 0 for the interface number.
	Retry	
	x	
	`UDP_SEND`	
	Cause 1	Abnormal net handle is designated at the time of using the `UDP_SEND` command.

Countermeasure	It is necessary to check the user application.
Retry	x
Cause 2	Ethernet level error in UDP_SEND.
Countermeasure	Please execute the UDP_SEND command again. Additionally, check for any abnormalities in connectors and cables.
Retry	O
Cause 3	Abnormal value is designated for the other party's IP address at the time of using the UDP_SEND command.
Countermeasure	It is necessary to check the user application.
Retry	x

ROUTE_ADD

Cause	An entry has been duplicated or the number of entries exceeds the entry limitation for registration, thus registration is denied.
Countermeasure	Please check to see if the command parameters are correct. If the parameters are normal, it is assumed that the number of entries exceeds the allotted amount, please delete the route information using the ROUTE_DEL command.
Retry	x

ROUTE_DEL

Cause	Designated entry does not exist.
Countermeasure	It is necessary to check the user application. Please check to see if the command parameters are correct. Please confirm that the route information is registered.
Retry	x

Sega Dreamcast

AVE-PPP Specification Guide V. 1.3

Access Co., Ltd.

Sega Dreamcast

AVE-PPP Specifications Guide TOC

1. Preface

PPP is the protocol responsible for connection, communication and disconnection of the link and network layers. Replacing the packet driver that is normally used for LAN connections, PPP intermediates between the upper-level TCP/IP and the lover lever serial driver.

2. Functions

Conformity to RFC1661, 1662, 1332 and 1334. It does not support Link Quality Report (LQR). It is compatible with the following three user authorization types.

- PAP,
- CHAP
- Method of changing specific procedures described in the script.

Additionally, it is possible to use data compression for the following.

- ACFCOMP (Address header compression)
- PROTOCOMP (Protocol header compression)
- VJCOMP (TCP header compression)

3. Function Details

PPP_Initialize

Initializes PPP.

Description

Initializes PPP. It is necessary to execute this before calling `AvepppOpen()`.

```
int AvepppInitialize(void);
```

Parameters

`int AvepppInitialize()`

In the case of `0`, it is successful. In the case of `-1`, there is an error.

PPP_TERMINATE

Executes the ending process for PPP.

Description

Executes the ending process for PPP. Please execute this function as a pair with `AvepppInitialize()`. It is necessary to close the circuit beforehand.

```
void AvepppTerminate(void);
```

Parameters

None

PPP_OPEN

Connects the circuit.

Description

Connects the circuit. This function starts the PPP open return that operates in the background. Please call AvepppGetStatus() to acquire the current connection conditions.

```
int AvepppOpen(const AvepppOpenParam* aOpenParam);
```

Parameters

```
int AvepppOpen()
```

Handle. In the case of `-1`, there is an error. However, this function always ends normally.

```
const AvepppOpenParam* aOpenParam
```

Open parameter.

Details

Open parameter.

```
struct AvepppSerialParam {
            short speed;            /* Baud rate 0:2400, 1:4800, 2:9600, */
                                    /* 3:19200, 4:38400 5:57600 (,6:115200) */
            short stop;             /* Stop bit  0:1bit, 1:2bit */
            short parity;           /* Parity  0:even, 1:odd, 2:none */
            short databit;          /* Data length  0:8bit, 1:7bit */
            short flow;             /* Hard flow control  0:non, 1:RTS, 2:DTR */
            long reserved_1;        /* Make it 0 for extension use */
            long reserved_2;        /* Make it 0 for extension use */
        };

        struct AvepppTelephoneParam {
            short dialtype;         /* Dial type 0:tone, 1:pulse */
            short outside_line;     /* Outside line (call 0)  0:nonexistent 1:exists */
            short timeout;          /* Timeout during communication */
            short dial_retry;       /* Redial frequency  1~99 */
            short dial_interval;    /* Redial terms (sec), more than 30sec is recommended
*/
            char* outside_number;   /* Outside line ( call 0) number character string */
            long reserved_1;        /* Make it 0 for extension use */
            long reserved_2;        /* Make it 0 for extension use */
        };
```

```
        struct AvepppConnectionParam {
            n_long his_ip;           /* Connection destination IP address (unused) */
            n_long my_ip;            /* Personal IP address (unused) */
            short recognize;         /* User authorization 0:PAP, 1:chat, 2:CHAP, */
                                     /* 3:reservation  4:depends on designation of
connection destination */
            short mru;               /* Maximum receiving bytes less than 2,048 (invalid,
1,500 */
            long magic_number;       /* Magic number, 8-column hexadecimal */
            short acfcomp;           /* HDLC header compression  0:enable, 1:disable */
            short protocomp;         /* PPP header compression  0:enable, 1:disable */
            short vjcomp;            /* IPCP's IP-TCP header compression  0:enable,
1:disable */
            long reserved_1;         /* Make it 0 for extension use */
            long reserved_2;         /* Make it 0 for extension use */
        };

        struct AvepppOpenParam {
            struct AvepppSerialParam* serial;
            struct AvepppTelephoneParam* telephone;
            struct AvepppConnectionParam* pppconection;
            char* login_script1;     /* Login script 1 (for modem initialization use) */
            char* login_script2;     /* Login script 1 (for provider use) */
            char* telephone_number;  /* Objective destination telephone number */
            char* login;             /* Login name */
            char* password;          /* password */
            long reserved_1;         /* Make it 0 for extension use */
            long reserved_2;         /* Make it 0 for extension use */
        };
```

Note

The data length of the serial is fixed to 8bits.

Additionally, software flow control (XON/OFF) cannot be used.

Explanation

Magic number is an ID that is released to detect if a response is falling into an infinite loop when communication with a connection destination becomes abnormal, such as the condition of echo back; random numbers should be designated as much as possible. If not required, please enter 0. Please refer to "AVE-PPP Script Version 1.0."

Example

```
#include <aveppp.h>

int test(void) {
      AvepppSerialParam aSerialParam;
    AvepppTelephoneParam aTelephoneParam;
    AvepppConnectionParam aConnectionParam;
    AvepppOpenParam aOpenParam;

    aOpenParam.serial = &aSerialParam;
    aOpenParam.telephone = &aTelephoneParam;
    aOpenParam.pppconection = &aConnectionParam;

    aOpenParam.login_script1 = (
        "send\"AT\\r\"\n"
        "wait\"OK\",5\n"
        "send\"ATZ\\r\"\n"
        "wait\"OK\",5\n"
        "send\"ATX3\\r\"\n          "/* Dial tone 3=ignore,4=detection */
        "wait\"OK\",5\n"
        "send\"ATW2\\r\"\n          "/* Speed display 0=serial,1=both,2=circuit*/
        "wait\"OK\",5\n"
        "send\"ATM1\\r\"\n          "/* Sound volume 0=off,1=carrier,2=on */
        "wait\"OK\",5\n"
        "send\"AT&C1\\r\"\n         "/* CD 0=on,1=carrier */
        "wait\"OK\",5\n"
        "send\"AT&D2\\r\"\n         "/* ER 0=ignore,1=command,2=onhook,3=atz */
        "wait\"OK\",5\n"
        "send\"AT&K3\\r\"\n         "/* flow 0=non,3=rts,4=xon */
        "wait\"OK\",5\n"
     );
    aOpenParam.login_script2 = "dial\"%t\"\n";
    aOpenParam.telephone_number = "03-5678-1234";
    aOpenParam.login = "aladdin";
    aOpenParam.password = "opensesame";
    aOpenParam.reserved_1 = 0;
    aOpenParam.reserved_2 = 0;

    aOpenParam.serial->speed = 4;
    aOpenParam.serial->stop = 0;
    aOpenParam.serial->parity        = 2;
    aOpenParam.serial->flow = 1;
    aOpenParam.serial->databit = 0;
    aOpenParam.serial->reserved_1 = 0;
    aOpenParam.serial->reserved_2 = 0;
```

```
        aOpenParam.telephone->dialtype = 0;
    aOpenParam.telephone->outside_line = 0;
    aOpenParam.telephone->timeout = 60;
    aOpenParam.telephone->dial_retry = 1;
    aOpenParam.telephone->dial_interval = 30;
    aOpenParam.telephone->outside_number = "0,";
    aOpenParam.telephone->reserved_1 = 0;
    aOpenParam.telephone->reserved_2 = 0;

    aOpenParam.pppconection->mru = 1500;
    aOpenParam.pppconection->magic_number = 0x12345678;
    aOpenParam.pppconection->acfcomp = 0;
    aOpenParam.pppconection->protocomp = 0;
    aOpenParam.pppconection->vjcomp = 0;
    aOpenParam.pppconection->recognize = 4;
    aOpenParam.pppconection->reserved_1 = 0;
    aOpenParam.pppconection->reserved_2 = 0;

    AvepppOpen(&aOpenParam);
}
```

PPP_CLOSE

Executes circuit disconnection.

Description

Executes circuit disconnection. This function starts the PPP close routine, which operates in the background, and returns immediately. Please call `AvepppGetStatus()` to acquire the current connection conditions.

```
int AvepppClose(int handle);
```

Parameters

`int AvepppClose()`

In the case of `0`, it is successful. In the case of `-1`, there is an error. However, this function always ends normally.

`int handle;`

Handle number returned by `AvepppOpen()`.

PPP_GETSTATUS

Acquires current conditions of PPP.

Description

Acquires current conditions of PPP.

```
int AvepppGetStatus(AvepppStatus* aStatus);
```

Parameters

```
int AvepppGetStatus()
```

In the case of `0`, it is successful. In the case of `-1`, there is an error. However, this function always ends normally.

```
AvepppStatus* aStatus
```

This is the status.

Details

Status

```
enum AvepppConnect {
    AvepppConnectUnused,          /* Unopened */
    AvepppConnectOpenStandby,     /* Awaiting execution after open request */
    AvepppConnectDialing,         /* Dialing */
    AvepppConnectAuthen,          /* Processing authorization */
    AvepppConnectEstablished,     /* Opening completed */
    AvepppConnectDisconnected,    /* Disconnect by other party */
    AvepppConnectCloseStandby,    /* Processing disconnection */
    AvepppConnectOnhooking,       /* Processing circuit disconnection */
    AvepppConnectFail             /* Connection failure */
};

enum AvepppError {
    AvepppErrorNon,               /* No error */
    AvepppErrorModem,             /* Timeout error when initializing modem */
    AvepppErrorBusy,              /* Circuit in use */
    AvepppErrorNoDialTone,        /* No dial tone */
    AvepppErrorScript,            /* Timeout error during login */
    AvepppErrorLcp,               /* Error in LCP negotiation */
    AvepppErrorAuth,              /* Error during authorization */
    AvepppErrorIpcp,              /* Error during IPCP negotiation */
    AvepppErrorTcp                /* Error when initializing TCP */
};
```

```
struct AvepppStatus {
    n_long his_ip;        /* Connection destination IP */
    n_long my_ip;         /* Personal IP */
    short phase;          /* Operation phase (=AvepppConnect) */
    long baud_rate;       /* Circuit speed */
    short port_inuse;     /* Modem in use */
    short last_error;     /* Error status (=AvepppError) */
} AvepppStatus;
```

First, the phase `AvepppConnectUnused` changes to `AvepppConnectOpenStandby` when `AvepppOpen()` is executed and moves on to `AvepppConnectEstablished` via `AvepppConnectDialing` and `AvepppConnectAuthen`. In the case of an error during this process, it will become `AvepppConnectFail`. In the case of disconnection by the other party, it will become `AvepppConnectionDisconnected`.

As the circuit will not open in the case of `AvepppConnectFail` or `AvepppConnectDisconnected`, please call `AvepppClose()` to complete disconnection.

When `AvepppClose()` is called, the condition changes to `AvepppConnectCloseStandby` and `AvepppConnectOnhook`, and finally becomes `AvepppConnectUnused`.

`last_error` is effective only for the condition of `AvepppConnectFail`, and displays the factor that failed.

`his_ip`, `my_ip` and `baud_rate` are effective only for the condition `AvepppConnectEstablished`.

`port_inuse` is effective all of the time, and just after the execution of `AvepppOpen()`, `AvepppClose()` is executed and becomes `!0` just before completing the closing process (it changes to the condition of `AvepppConnectUnused`).

4. Additional APIs for PPP

Setting the IP Address and DNS Server Information that is Passed by IPCP

Description

This API sets the parameters that are passed to the other side during the negotiation process that is performed in the IPCP layer when connecting through PPP. These parameters consist of the local IP address (`myip`), the IP address for the other end point (`hisip`), and the DNS server addresses `mydns1`, `mydns2`, `hisdns1`, and `hisdns2`).

Normally, because the IP address and DNS server address specified by the other side are used, set `0` for these parameters. Also set `0` for the information that you wish to get by using `AvepppGetDns`.

```
#include "aveppp.h"

            void AvepppSetDns(long myip,
                    long hisip,
                    long mydns1,
                    long mydns2,
                    long hisdns1,
                    long hisdns2);
```

Parameters

`myip`

Local IP address

`hisip`

Other side's (other side's PPP end point) IP address

`mydns1`

Primary DNS server IP address

`mydns2`

Secondary DNS server IP address

`hisdns1`

Other side's primary DNS server IP address

`hisdns2`

Other side's secondary DNS server IP address

Return Value

None

Note

This API must be called if using `AvepppGetDns` to get the DNS addresses. Call this API before `AvepppOpen`.

Example

```
/* Sets "0" for DNS in order to issue a DNS request */
    AvepppSetDns(0, 0, 0, 0, 0, 0);

    /* Connects with PPP (dialing begins) */
    AvepppOpen(&iparam);

    /* Waits until connection with PPP is completed */
    for (;;)
    {

        :
        :

    }
```

Getting the IP Address and DNS Server Information through IPCP

This API gets the results of the negotiation process that is performed in the IPCP layer when connecting through PPP. The parameters that are determined are the local IP address (`myip`), the IP address for the other end point (`hisip`), and the DNS server addresses (`mydns1`, `mydns2`, `hisdns1`, and `hisdns2`).

As a result of the negotiation process, the local address is set in `myip`, and the addresses of the DNS servers that should be set on the local side are set in `mydns1` and `mydns2`.

`myip` is set and activated as the AVE-TCP interface when the IPCP negotiations are completed. `mydns1` and `mydns2` are used as parameters when the DNS library is initialized. (Because the above IP addresses are expressed in Little Endian format, the byte order must be changed when they are used in the initialization of the DNS library.)

The IP address of the end point on the other side is set in `hisip`. Because `hisdns1` and `hisdns`" are used to report by the local side to notify the other side of the DNS server addresses, they are not normally used when the local side is operating as a client. (However, because the PPP's internal parameters are copied by the above function, dummy variables must be provided.)

The values are undetermined if this function is called before the IPCP negotiations are completed.

If the IPCP negotiations fail (example: when the IPCP connection is with a PPP server with settings that do not report the DNS server), `mydns1` and `mydns2` are both `0`. In this case, the DNS library must be initialized so that a fixed DNS server is used on the application side.

```
#include "aveppp.h"

             void AvepppGetDns(long* myip,
                     long* hisip,
                     long* mydns1,
                     long* mydns2,
                     long* hisdns1,
                     long* hisdns2);
```

Parameters

`myip`

Starting address of area where the local IP address is stored

hisip

Starting address of area where the IP address for the other side (the other side's PPP end point) is stored

mydns1

Starting address of area where the IP address for the primary DNS server is stored

mydns2

Starting address of area where the IP address for the secondary DNS server is stored

hisdns1

Starting address of area where the IP address for the other side's primary DNS server is stored

hisdns2

Starting address of area where the IP address for the other side's secondary DNS server is stored

Return Value

None

Note

In order to get the DNS addresses, `AvepppSetDns` must be called before this API is called.

Example

```
#define PEER_HTONL(n) \
        (((((unsigned long)(n))&0xFF)<<24)
        |((((unsigned long)(n))&0xFF00)<<8)
        |((((unsigned long)(n))&0xFF0000)>>8)
        |((((unsigned long)(n))&0xFF000000)>>24))

    void
    peerPPPSetUpDNS(void)
    {
        extern unsigned char PrimaryDNS[4];
        extern undigned char SecondaryDNS[4];
        long myip, hisip, mydns1, mydns2, hisdns1, hisdns2;

        /* Gets the configuration data */
        wave_memcpy(&hisdns1, PrimaryDNS, 4);
        wave_memcpy(&hisdns2, SecondaryDNS, 4);

        /* Sets the DNS server addresses when they could not be gotten through IPCP */
        AvepppGetDns(&myip, &hisip, &mydns1, &mydns2, &hisdns1, &hisdns2);

        /* If the primary DNS server address was gotten, substitute that address */
        if (mydns1 != 0) {
            hisdns1 = PEER_HTONL(mydns1);
        }

        /* If the secondary DNS server address was gotten, substitute that address */
        if (mydns2 != 0) {
            hisdns2 = PEER_HTONL(mydns2);
        }
        ADNS_Initialize(nil, hisdns1, hisdns2);
    }
```

Sega Dreamcast

DNS API Specification

Draft

Sega Dreamcast

DNS API
Table of Contents

1. DNS API Specification

This specification stipulates the requirements on the library for resolving host name using the DNS protocol installed on the AVE-TCP.

The library provides the following features:

1) Library initialization.
2) Library termination.
3) Getting a ticket.
4) Releasing a ticket.
5) Name resolution.
6) Getting ticket information.

Because these features require AVE-TCP and AVE-PPP, be sure to initialize these protocols before use.

To avoid the blocking state which occurs during the DNS resolution, the concept of “ticket” is introduced into this library. Here, the application program desired to resolve the host name using DNS must perform the following tasks:

1) Issue a ticket
2) Try to resolve a name using the ticket
3) If the name can be resolved, get its IP address and release the ticket.

2. DNS API

This library provides the following APIs…

ADNS_Initialize

Initializes the DNS library.

Description

This API initializes the DNS library.

```
#include "dns.h"

ADNS_Initialize(wave_char *in_def_domain,
          wave_32bit in_prim_svr,
          wave_32bit in_sec_svr);
```

Argument

`in_def_domain`	Domain name of host
`in_prim_svr`	IP address of the primary server
`in_sec_svr`	IP address of the secondary server

Return Value

None

ADNS_Finalize()

Terminates the DNS library.

Description

This API terminates the DNS library.

```
#include "dns.h"
ADNS_Finalize();
```

Argument

None

Return Value

None

ADNS_GetTicket

Issues a ticket which contains the name of the host to be resolved by DNS.

Description

This API issues a ticket which contains the name of the host to be resolved by DNS. If it fails to issue a ticket, it returns -1.

```
#include "dns.h"
ADNS_GetTicket(wave_char *in_host);
```

Argument

`in_host`	Name of host for name resolution using DNS

Return Value

Value of ticket

ADNS_ReleaseTicket

Releases a ticket.

Description

This API releases a ticket.

```
#include "dns.h"
ADNS_ReleaseTicket(wave_int in_ticket_id);
```

Argument

`in_ticket_id`	Value of the ticket to release

Return Value

None

ADNS_LookUp

Uses ticket to resolve host name.

Description

This API attempts to use the ticket to resolve the host name. If name resolution can succeed, `eOK` is returned and the IP address of the resolved name is stored in `out_addr`. If it is blocked due to server response waiting, `eWouldBlock` is returned; in this case be sure to try again. `eGeneric` is returned when protocol errors occur. `eTimeout` is returned when no response on its request can be obtained from the server.

The maximum timeout duration is 50 seconds for each of the primary server and the secondary server. (A total of three attempts of 5 seconds, 15 seconds and 30 seconds will be attempted.)

```
#include "dns.h"
ADNS_LookUp(wave_int in_ticket_id, wave_byte *out_addr);
```

Argument

`in_ticket_id`	Value of ticket
`out_addr`	IP address of name resolved

Return Value

`eOK`	Normal return
`eGeneric`	Protocol error
`eWouldBlock`	Blocked
`eTimedOut`	Timeout occurs

ADNS_GetTicketInfo

Sets to out_host the lead address of the character string.

Description

This API sets to `out_host` the lead address of the character string containing the host name written in the ticket as indicated by `in_ticket_id`,

```
#include "dns.h"
ADNS_GetTicketInfo(wave_int in_ticket_id, wave_char **out_host);
```

Argument

`in_ticket_id`	Value of ticket
`out_host`	Host name written in the ticket

Return Value

None