AdaSockets reference manual

for AdaSockets version 1.8.4.7
16 May 2004

Samuel Tardieu

Table of Contents

1 What is AdaSockets?

AdaSockets is a set of free software Ada packages allowing Ada programmers to use the so-called BSD sockets from their favourite programming language. AdaSockets has been designed and tested with the GNAT free software Ada compiler, but should be portable to other compilers quite easily.

Starting from release 3.14, the GNAT compiler started to integrate a `GNAT.Sockets` package. However, this package is GNAT specific and contains at this time less features than AdaSockets. At some point, AdaSockets may use `GNAT.Sockets` as its underlying sockets structure.

AdaSockets philosophy is to help the Ada programmer by providing easy-to-use objects. Special care has been taken to ensure that performances do however remain good.

2 Installing AdaSockets

Installing AdaSockets on a Unix or OpenVMS machine is as simple as typing a few commands. Once you got the latest version of AdaSockets (see Appendix B [Resources on the Internet], page 29), uncompress and untar it and go to the top-level directory of the distribution.

You must configure the AdaSockets distribution by using the `configure` command, such as in:

```
./configure --prefix=/users/sam/adasockets
```

If you want to install AdaSockets under the '`/usr/local`' hierarchy, you do not need to specify the '`--prefix`' option. Make sure you have write permission on the target directories.

Once AdaSockets is configured, you can compile and install it by using the `make` command:

```
make install
```

The GNU make program is recommended but not mandatory. On your system, it may be installed under the `gmake` name.

3 Using AdaSockets

3.1 Compiling an Ada application

AdaSockets comes with an `adasockets-config` application that can be used to retrieve installation parameters while using `gnatmake` to compile your Ada application. The `--cflags` parameters tells `adasockets-config` to output the path to the Ada packages sources, while `--libs` asks for the path to the Ada library.

```
gnatmake 'adasockets-config --cflags' mainprog
  -largs 'adasockets-config --libs'
```

3.2 Setting up unicast sockets

Unicast sockets are used everywhere on the Internet, for surfing the web, sending electronic mails or accessing remote files. They come in two flavours:

TCP
: TCP is a connected mode, in which packets are sent in a reliable and ordered way. Everything sent at one end will eventually arrive in the same order at the other end, the underlying operating system takes care of retransmitting missing packets and reordering out-of-order ones.

UDP
: UDP is a non-connected mode. A packet sent on a UDP socket may or may not arrive at the other end. This is a much lighter protocol when reliability is not needed as the operating system does not have to use large buffers to reorder packets. Also, this generates less traffic as no acknowledgments need to be sent by the kernels.

The package `Sockets` defines a `Socket_FD` tagged type. An instance of this type (or of any of its descendants) represents an incoming or outgoing socket. Two different kinds of sockets can be created: unicast (one-to-one) and multicast (many-to-many).

3.3 Setting up multicast sockets

Multicast sockets are used for group communication over the Internet. To use multicast sockets, you must be connected to a multicast network and use a multicast-enabled operating system (such as most Unices, Linux or even recent Windows versions). Unless you are connected to the mbone (multicast backbone) or have setup a private multicast network, you will only be able to use multicast on your local network.

A multicast socket is somewhat similar to a UDP socket; in particular, packets may be lost or misordered.

You can create a multicast socket using the function `Create_Multicast_Socket` in the package `Sockets.Multicast`. This function returns a `Multicast_Socket_FD` object, which derives from `Socket_FD`.

`Create_Multicast_Socket` takes care of the whole setup of your multicast socket. You do not need to call any additionnal subprogram before using it. In particular, `Create_Multicast_Socket` will take care of registering the multicast socket to the operating system, so that the latter can tell the connected routers to propagate the subscription to the mbone as needed.

3.4 Sending and receiving data

In AdaSockets, data can be sent and received in three different ways: raw, string-oriented and stream-oriented.

3.4.1 Raw data manipulation

Raw data is manipulated using the predefined `Ada.Streams.Stream_Element_Array` Ada type. This corresponds to an array of bytes, or an `unsigned char *` in the C programming language.

3.4.2 String-oriented exchanges

String-oriented exchanges provides the programmer with `Ada.Text_IO` like subprograms. Most Internet protocols are line-oriented and those subprograms are perfectly suited to implement those.

When sending data, the classical `CR + LF` sequence will be sent at the end of each line. When receiving data, `CR` characters are discarded and `LF` is used as the end-of-line marker.

The `Get` function is tied to the size of the operating system buffer. It is better to use `Get_Line` to get a full line. The line can be as long as the length of the Ada buffer. The Ada buffer can be adjusted by using the `Set_Buffer` and `Unset_Buffer` procedures.

When using string-oriented exchanges with datagram protocols such as UDP, setting a buffer using `Set_Buffer` for the receiving socket is mandatory. If you don't, the receiving socket will loose data and will be unable to reconstitute the string.

4 Sockets package

The `Sockets` package contains all the definitions and subprograms needed to build a simple unicast client or server.

`type` **Socket_FD** `is tagged private;` Sockets.Socket_FD

The `Socket_FD` tagged type is the root type of all sockets. It gets initialized by calling [Socket (procedure)], page 17.

Accept_Socket (procedure)

Purpose Accept an incoming connection

Prototype

`procedure` **Accept_Socket** Sockets.Accept_Socket
(*Socket* : `Socket_FD`; *New_Socket* : `out Socket_FD`);

Parameters

Socket	in	Initialized
New_Socket	out	Incoming socket object

Description

This procedure creates a new socket corresponding to an incoming connection on TCP socket *Socket*. All the communications with the peer will take place on *New_Socket*, while the program can accept another connection on *Socket*.

New_Socket must not be initialized before calling this procedure, or must have been cleaned up by calling `Shutdown`, in order to avoid a file descriptors leak.

`Accept_Socket` will block until an incoming connection is ready to be accepted.

Example

```
declare
   Sock     : Socket_FD;
   Incoming : Socket_FD;
begin
   -- Create a TCP socket listening on local port 4161
   Socket (Sock, PF_INET, SOCK_STREAM);
   Bind (Sock, 4161);
   Listen (Sock, 3);
   -- One-connection-at-a-time server (3 may be pending)
   loop
      -- Wait for a new connection and accept it
      Accept_Socket (Sock, Incoming);
      -- Do some dialog with the remote host
      Do_Some_Dialog (Incoming);
      -- Close incoming socket and wait for next connection
      Shutdown (Incoming);
   end loop;
end;
```

SEE ALSO [Bind (procedure)], page 8,
[Listen (procedure)], page 12,
[Shutdown (procedure)], page 17,
[Socket (procedure)], page 17.

Bind (procedure)

PURPOSE Associate a local port to a socket

PROTOTYPE

procedure **Bind** (*Socket* : Socket_FD; Sockets.Bind
Port : Natural; *Host* : String := "");

PARAMETERS

Socket	in	Initialized socket object
Port	in	Local port to bind to
Host	in	Local interface to bind to

DESCRIPTION

This procedure requests a local port from the operating system. If 0 is given in *Port*, the system will assign a free port whose number can later be retrieved using [Get_Sock_Port (function)], page 23. Also, most operating systems require special privileges if you want to bind to ports below 1024.

If *Host* is not the empty string, it must contain the IP address of a local interface to bind to, or a name which resolves into such an address. If an empty string is given (the default), the socket will be bound to all the available interfaces.

EXCEPTIONS

Socket_Error	Requested port or interface not available

SEE ALSO [Listen (procedure)], page 12,
[Socket (procedure)], page 17.

Connect (procedure)

PURPOSE Connect a socket on a given host/port

PROTOTYPE

procedure **Connect** (*Socket* : Socket_FD; Sockets.Connect
Host : String; *Port* : Positive);

PARAMETERS

Socket	in	Initialized socket object
Host	in	Host to connect to
Port	in	Port to connect to

DESCRIPTION

This procedure connects an initialized socket to a given host on a given port. In the case of a TCP socket, a real connection is attempted. In the case of a UDP socket, no connection takes place but the endpoint coordinates are recorded.

EXCEPTIONS

`Connection_Refused`	The connection has been refused by the server
`Socket_Error`	Another error occurred during the connection

EXAMPLE

```
declare
   Sock : Socket_FD;
begin
   -- Create a TCP socket
   Socket (Sock, PF_INET, SOCK_STREAM);
   -- Connect it to rfc1149.net's mail server
   Connect (Sock, "mail.rfc1149.net", 25);
   -- Do a mail transaction then close the socket
   [...]
end;
```

SEE ALSO [Socket (procedure)], page 17.

Get (function)

PURPOSE Get a string from a remote host

PROTOTYPE

`function` **Get** (*Socket* : `Socket_FD'Class`) `return String;` Sockets.Get

PARAMETERS

Socket	in	Initialized and connected socket object

RETURN VALUE

Some characters that have been received

DESCRIPTION

This function receives some characters from a remote host. As soon that at least one character is available, the current reception buffer is returned.

There is usually little gain in using this function whose behaviour is comparable to the one of [Receive (function)], page 14. Other functions such as [Get_Char (function)], page 9, or [Get_Line (function)], page 11, allow more structured programming.

However, this function may be used to avoid loosing characters when calling [Unset_Buffer (procedure)], page 18, if, for some reason, the remote host may have sent some.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending any data

Get_Char (function)

PURPOSE Get a character from a remote host

PROTOTYPE

```
function Get_Char (Socket : Socket_FD'Class)
        return Character;
```

Sockets.Get_Char

PARAMETERS

Socket	in	Initialized and connected socket object

RETURN VALUE

One character sent by the remote host

DESCRIPTION

This function receives exactly one character from the remote host.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending the character

SEE ALSO [Get (function)], page 9,
[Get_Line (function)], page 11,
[Get_Line (procedure)], page 10,
[Receive (procedure)], page 14,
[Set_Buffer (procedure)], page 16.

Get_Line (procedure)

PURPOSE Get a whole line from a remote host

PROTOTYPE

```
procedure Get_Line (Socket : Socket_FD'Class;
        Str : in out String; Last : out Natural);
```

Sockets.Get_Line

PARAMETERS

Socket	in	Initialized and connected socket object
Str	in out	String to fill
Last	out	Last index used in the string

DESCRIPTION

This procedure receives one line from the remote host. A line consists into zero or more characters followed by an optional CR and by a LF. Those terminators are stripped before the line is returned.

This procedure blocks until one full line has been received.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending a whole line

SEE ALSO [Get (function)], page 9,
[Get_Char (function)], page 9,
[Get_Line (function)], page 11,
[Receive (procedure)], page 14,
[Set_Buffer (procedure)], page 16.

Get_Line (function)

PURPOSE Get a whole line from a remote host

PROTOTYPE

```
function Get_Line (Socket : Socket_FD'Class;          Sockets.Get_Line
         Max_Length : Positive := 2048)
         return String;
```

PARAMETERS

Socket	in	Initialized and connected socket object
Max_Length	in	Maximum returned line length

RETURN VALUE

A line without the CR and LF separators

DESCRIPTION

This function receives one line from the remote host. A line consists into zero or more characters followed by an optional CR and by a LF. Those terminators are stripped before the line is returned.

This function blocks until one full line has been received. The line length is limited with the value of the Max_Length argument, to avoid exhaustion of the secondary stack.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending a whole line

SEE ALSO [Get (function)], page 9,
[Get_Char (function)], page 9,
[Get_Line (procedure)], page 10,
[Receive (procedure)], page 14,
[Set_Buffer (procedure)], page 16.

Getsockopt (procedure)

PURPOSE Retrieve a socket option

PROTOTYPE

```
procedure Getsockopt (Socket : Socket_FD;          Sockets.Getsockopt
         Level : Socket_Level := SOL_SOCKET;
         Optname : Socket_Option; Optval : out Integer);
```

PARAMETERS

Socket	in	Initialized and bound socket object
Level	in	Protocol level
Optname	in	Option name
Optval	out	Option value

Description

This procedure retrieves options applicable to a socket. Please see your operating system manual for usable levels and options.

Two levels are defined: `SOL_SOCKET` (the default) and `IPPROTO_IP`. The options are `SO_REUSEADDR`, `SO_REUSEPORT`, `IP_MULTICAST_TTL`, `IP_ADD_MEMBERSHIP`, `IP_DROP_MEMBERSHIP`, `IP_MULTICAST_LOOP`, `SO_SNDBUF` and `IP_RCVBUF`.

Note that unlike their C language counterpart, `Getsockopt` and `Setsockopt` do not require an extra parameter representing the length in bytes of the option value. AdaSockets nows the right size for every option.

See also [Setsockopt (procedure)], page 16.

Listen (procedure)

Purpose Establish a listen queue

Prototype

```
procedure Listen (Socket : Socket_FD;                    Sockets.Listen
                  Queue_Size : Positive := 5);
```

Parameters

Socket	in	Initialized and bound socket object
Queue_Size	in	Requested slots in the listen queue

Description

This procedure establishes a listen queue after a TCP socket as been initialized and bound. Each slot in the queue can hold one incoming connection that has not been accepted yet. Note that most operating systems ignore queue sizes larger than five.

See also [Accept_Socket (procedure)], page 7,
[Bind (procedure)], page 8,
[Socket (procedure)], page 17.

New_Line (procedure)

Purpose Send a CR/LF to a remote host

Prototype

```
procedure New_Line                                        Sockets.New_Line
    (Socket : Socket_FD'Class; Count : Natural := 1);
```

Parameters

Socket	in	Initialized and connected socket object
Count	in	Number of CR/LF sequences to send

Description

This procedure sends one or more CR/LF combinations to the peer.

EXCEPTIONS

`Connection_Closed`	Peer has prematurely closed the connection

SEE ALSO [Put (procedure)], page 13,
[Put_Line (procedure)], page 13.

Put (procedure)

PURPOSE Send a string to a remote host

PROTOTYPE

```
procedure Put (Socket : Socket_FD'Class;              Sockets.Put
               Str : String);
```

PARAMETERS

Socket	in	Initialized and connected socket object
Str	in	String to send

DESCRIPTION
This procedure sends the content of *Str* over an outgoing or incoming socket.

EXCEPTIONS

`Connection_Closed`	Peer has prematurely closed the connection

SEE ALSO [New_Line (procedure)], page 12,
[Put_Line (procedure)], page 13,
[Send (procedure)], page 15.

Put_Line (procedure)

PURPOSE Send a CR/LF terminated string to a remote host

PROTOTYPE

```
procedure Put_Line (Socket : Socket_FD'Class;         Sockets.Put_Line
                    Str : String);
```

PARAMETERS

Socket	in	Initialized and connected socket object
Str	in	String to send

DESCRIPTION
This procedure sends the content of *Str* plus a CR/LF combination over an outgoing or incoming socket.

EXCEPTIONS

`Connection_Closed`	Peer has prematurely closed the connection

SEE ALSO [New_Line (procedure)], page 12,
[Put (procedure)], page 13,
[Send (procedure)], page 15.

Receive (procedure)

PURPOSE Receive raw data over a socket

PROTOTYPE

```
procedure Receive (Socket : Socket_FD'Class;          Sockets.Receive
         Data : out Ada.Streams.Stream_Element_Array);
```

PARAMETERS

Socket	in	Initialized and bound or connected socket object
Data	out	Incoming data buffer

DESCRIPTION

This procedure receives data from a bound UDP socket or a connected TCP socket. It will block until the *Data* reception buffer has been totally filled.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before `Data'Length` bytes were received

SEE ALSO [Get_Line (function)], page 11,
[Get_Line (procedure)], page 10,
[Receive (function)], page 14,
[Receive_Some (procedure)], page 15.

Receive (function)

PURPOSE Receive raw data over a socket

PROTOTYPE

```
function Receive (Socket : Socket_FD;          Sockets.Receive
         Max : Ada.Streams.Stream_Element_Count := 4096)
         return Ada.Streams.Stream_Element_Array;
```

PARAMETERS

Socket	in	Initialized and bound or connected socket object
Max	in	Maximum data length

RETURN VALUE

Received raw data

DESCRIPTION

This procedure receives data from a bound UDP socket or a connected TCP socket. Only one system call will be performed; this function will return whatever data has arrived. Note that in GNAT the secondary stack may be used to store the data and may result in stack storage exhaustion.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending any data

SEE ALSO [Receive (procedure)], page 14,
[Receive_Some (procedure)], page 15,
[Get_Line (function)], page 11,
[Get_Line (procedure)], page 10.

Receive_Some (procedure)

PURPOSE Receive raw data over a socket

PROTOTYPE

```
procedure Receive_Some                                  Sockets.Receive_Some
        (Socket : Socket_FD'Class;
         Data : out Ada.Streams.Stream_Element_Array;
         Last : out Ada.Streams.Stream_Element_Offset);
```

PARAMETERS

Socket	in	Initialized and bound or connected socket object
Data	out	Incoming data buffer
Last	out	Index of last element placed into *Data*

DESCRIPTION

This procedure receives data from a bound UDP socket or a connected TCP socket. As soon as at least one byte has been read, it returns with *Last* set to the index of the latest written element of *Data*.

EXCEPTIONS

`Connection_Closed`	Peer has closed the connection before sending any data

SEE ALSO [Get_Line (function)], page 11,
[Get_Line (procedure)], page 10,
[Receive (function)], page 14,
[Receive (procedure)], page 14.

Send (procedure)

PURPOSE Send raw data over a socket

PROTOTYPE

```
procedure Send (Socket : Socket_FD;                            Sockets.Send
        Data : out Ada.Streams.Stream_Element_Array);
```

PARAMETERS

Socket	in	Initialized and connected socket object
Data	out	Data to be sent

DESCRIPTION

This procedure sends data over a connected outgoing socket or over an incoming socket.

EXCEPTIONS

`Connection_Closed`	Peer has prematurely closed the connection

SEE ALSO [Put (procedure)], page 13,
[Put_Line (procedure)], page 13.

Set_Buffer (procedure)

PURPOSE Install a line-oriented buffer of the socket object

PROTOTYPE

procedure Set_Buffer Sockets.Set_Buffer
(*Socket* : Socket_FD'Class; *Length* : Positive := 1500);

PARAMETERS

Socket	in	Initialized and connected socket object
Length	in	Size in bytes of the newly installed buffer

DESCRIPTION

This procedure puts the socket object in buffered mode. If the socket was already buffered, the content of the previous buffer will be lost. The buffered mode only affects character- and line-oriented read operation such as [Get (function)], page 9, [Get_Char (function)], page 9, and [Get_Line (function)], page 11. Other reception subprograms will not function properly if buffered mode is used at the same time.

The size of the buffer has to be greater than the biggest possible packet sent by the remote host, otherwise data loss may occur.

SEE ALSO [Unset_Buffer (procedure)], page 18.

Setsockopt (procedure)

PURPOSE Set a socket option

PROTOTYPE

procedure Setsockopt (*Socket* : Socket_FD; Sockets.Setsockopt
Level : Socket_Level := SOL_SOCKET;
Optname : Socket_Option; *Optval* : Integer);

PARAMETERS

Socket	in	Initialized and bound socket object
Level	in	Protocol level
Optname	in	Option name
Optval	in	Option value

DESCRIPTION

This procedure sets options applicable to a socket. Please see your operating system manual for usable levels and options.

Two levels are defined: `SOL_SOCKET` (the default) and `IPPROTO_IP`. The options are `SO_REUSEADDR`, `SO_REUSEPORT`, `IP_MULTICAST_TTL`, `IP_ADD_MEMBERSHIP`, `IP_DROP_MEMBERSHIP`, `IP_MULTICAST_LOOP`, `SO_SNDBUF` and `IP_RCVBUF`.

Note that unlike their C language counterpart, `Getsockopt` and `Setsockopt` do not require an extra parameter representing the length in bytes of the option value. AdaSockets nows the right size for every option.

SEE ALSO [Getsockopt (procedure)], page 11.

Shutdown (procedure)

PURPOSE Shutdown a socket

PROTOTYPE

```
procedure Shutdown                                    Sockets.Shutdown
     (Socket : in out Socket_FD; How : Shutdown_Type := Both);
```

PARAMETERS

Socket	in out	Socket object to shutdown
How	in	Direction to shutdown

DESCRIPTION

This procedure shutdowns either direction of the socket. *How* can take the value '`Send`', '`Receive`' or '`Both`'.

SEE ALSO [Socket (procedure)], page 17.

Socket (procedure)

PURPOSE Create a socket of the given mode

PROTOTYPE

```
procedure Socket (Socket : out Socket_FD;             Sockets.Socket
     Domain : Socket_Domain := PF_INET;
     Typ : Socket_Type := SOCK_STREAM);
```

PARAMETERS

Socket	out	Socket object to initialize
Domain	in	Protocol family
Typ	in	Kind of sockets

DESCRIPTION

This procedure initializes a new socket object by reserving a file descriptor to the operating system. For backward compatibility with older versions of this library, `AF_INET` is still accepted as a value but should be replaced as soon as possible with the proper `PF_INET`. Using `SOCK_STREAM` for the *Typ* argument will create a TCP socket while a `SOCK_DGRAM` will create a UDP one.

EXAMPLE

```
declare
   Sock : Socket_FD;
begin
   -- Create a TCP socket
   Socket (Sock, PF_INET, SOCK_STREAM);
   -- Perform some operations on socket
   [...]
   -- Shutdown the socket in both directions
   Shutdown (Sock, Both);
end;
```

SEE ALSO [Shutdown (procedure)], page 17.

Unset_Buffer (procedure)

PURPOSE Deinstall the line-oriented buffer of the socket object

PROTOTYPE

`procedure` **Unset_Buffer** Sockets.Unset_Buffer
(*Socket* : `Socket_FD'Class`);

PARAMETERS

Socket	in	Initialized and connected socket object

DESCRIPTION

This procedure deinstalls the buffer previously installed by [Set_Buffer (procedure)], page 16. If any data is still present in the buffer, it will be lost. To avoid this situation, the buffer can be flushed by calling [Get (function)], page 9.

5 Sockets.Multicast package

The `Sockets.Multicast` allows the creation of IP multicast sockets.

`type` **Multicast_Socket_FD** `is new` Sockets.Multicast.Multicast_Socket_FD
`Socket_FD with private;`

The `Multicast_Socket_FD` tagged type derives from the `Socket_FD` type. It gets initialized by calling [Create_Multicast_Socket (function)], page 20.

Create_Multicast_Socket (function)

PURPOSE Create an IP multicast socket

PROTOTYPE

`function` Sockets.Multicast.Create_Multicast_Socket
Create_Multicast_Socket (*Group* : `String;`
Port : `Positive;` *TTL* : `Positive := 16;`
Self_Loop : `Boolean := True`)
`return Multicast_Socket_FD;`

PARAMETERS

Group	in	IP address of the multicast group to join
Port	in	Port of the multicast group to join
TTL	in	Time-to-live of sent packets
Self_Loop	in	Should the socket receive the packets sent from the local host?

RETURN VALUE

The new initialized multicast socket

DESCRIPTION

This function creates an IP multicast socket attached to a given group, identified by its class E IP address and port.

Be careful when choosing the TTL parameter of your IP multicast socket. Most IP multicast routers do implement threshold-based filtering and will not let IP multicast packets leave your organization if the TTL on the last router is smaller than 16.

EXAMPLE

```
declare
   Sock : Multicast_Socket_FD;
begin
   -- Create a multicast socket on group 224.1.2.3 port 8763
   Sock := Create_Multicast_Socket ("224.1.2.3", 8763);
   -- Perform some operations on socket
   [...]
   -- Shutdown the socket in both directions
   Shutdown (Sock, Both);
end;
```

SEE ALSO [Send (procedure)], page 15,
[Shutdown (procedure)], page 17.

Create_Multicast_Socket (function)

PURPOSE Create an IP multicast socket

PROTOTYPE

`function` Sockets.Multicast.Create_Multicast_Socket
Create_Multicast_Socket (*Group* : `String`;
Port : `Positive`; *Local_Port* : `Natural`; *TTL* : `Positive := 16`)
`return Multicast_Socket_FD;`

PARAMETERS

Group	in	IP address of the multicast group to join
Port	in	Port of the multicast group to join
Local_Port	in	Local port number to use
TTL	in	Time-to-live of sent packets

RETURN VALUE
The new initialized multicast socket

DESCRIPTION
This function creates an IP multicast socket attached to a given group, identified by its class E IP address and port. If Local_Port is 0, a free port will automatically be chosen by your operating system.

This function should be used when you want to send packets to a multicast group without receiving any packet yourself.

EXAMPLE

```
declare
   Sock : Multicast_Socket_FD;
begin
   -- Create a multicast socket on group 224.1.2.3 port 8763
   Sock := Create_Multicast_Socket ("224.1.2.3", 8763);
   -- Perform some operations on socket
   [...]
   -- Shutdown the socket in both directions
   Shutdown (Sock, Both);
end;
```

SEE ALSO [Send (procedure)], page 15,
[Shutdown (procedure)], page 17.

6 Sockets.Naming package

The `Sockets.Naming` package contains types and helper functions needed to manipulate Internet host names and addresses.

```
type Address is record                                  Sockets.Naming.Address
      H1, H2, H3, H4 : Address_Component;
     end record;
```

This type represents an IPv4 address with `H1` being the first octet and `H4` the last one. For example, 137.194.161.2 is represented by `H1=137, H2=194, H3=161, H4=2`.

```
type Address_Array is array (Positive range       Sockets.Naming.Address_Array
     <>) of Address;
```

Helper type

```
type Address_Component is Natural             Sockets.Naming.Address_Component
     range 0 .. 255;
```

Helper type

```
type Host_Entry (N_Aliases, N_Addresses :          Sockets.Naming.Host_Entry
     Natural) is new Ada.Finalization.Controlled with record
       Name : String_Access;
       Aliases : String_Array (1 .. N_Aliases);
       Addresses : Address_Array (1 .. N_Addresses);
     end record;
```

The `Host_Entry` type holds a set of names and IP addresses associated with a host. Each host can have several IP address as well as several aliases.

```
type String_Access is access String;              Sockets.Naming.String_Access
```

Helper type

```
type String_Array is array (Positive range <>)     Sockets.Naming.String_Array
     of String_Access;
```

Helper type

Address_Of (function)

PURPOSE Get the IP address of a host

PROTOTYPE

```
function Address_Of                                  Sockets.Naming.Address_Of
       (Something : String)
       return Address;
```

PARAMETERS

Something	in	Host name or IP address

RETURN VALUE

IPv4 address

EXCEPTIONS

`Naming_Error`	No information available for this name or address

SEE ALSO [Name_Of (function)], page 25.

Any_Address (function)

PURPOSE Special address representing any address on the local host

PROTOTYPE

```
function Any_Address                              Sockets.Naming.Any_Address
```

RETURN VALUE

Equivalent to `INADDR_ANY` in the C programming language

Get_Peer_Addr (function)

PURPOSE Retrieve IP address of remote host

PROTOTYPE

```
function Get_Peer_Addr                            Sockets.Naming.Get_Peer_Addr
        (Socket : Socket_FD)
        return Address;
```

PARAMETERS

Socket	in	Connected socket object

RETURN VALUE

Peer address

SEE ALSO [Get_Peer_Port (function)], page 22,
[Get_Sock_Addr (function)], page 23.

Get_Peer_Port (function)

PURPOSE Retrieve port used by remote host

PROTOTYPE

```
function Get_Peer_Port                            Sockets.Naming.Get_Peer_Port
        (Socket : Socket_FD)
        return Positive;
```

PARAMETERS

Socket	in	Connected socket object

RETURN VALUE

Port used on the remote host

SEE ALSO [Get_Sock_Port (function)], page 23, [Get_Peer_Addr (function)], page 22.

Get_Sock_Addr (function)

PURPOSE Retrieve IP address of local host

PROTOTYPE

```
function Get_Sock_Addr                    Sockets.Naming.Get_Sock_Addr
    (Socket : Socket_FD)
    return Address;
```

PARAMETERS

Socket	in	Connected socket object

RETURN VALUE

Address of local interface used

SEE ALSO [Get_Sock_Port (function)], page 23, [Get_Peer_Addr (function)], page 22.

Get_Sock_Port (function)

PURPOSE Retrieve port used by local host

PROTOTYPE

```
function Get_Sock_Port                    Sockets.Naming.Get_Sock_Port
    (Socket : Socket_FD)
    return Positive;
```

PARAMETERS

Socket	in	Connected socket object

RETURN VALUE

Port used on the local host

SEE ALSO [Get_Peer_Port (function)], page 22, [Get_Sock_Addr (function)], page 23.

Host_Name (function)

Purpose Get the name of the current host

Prototype

`function` **Host_Name** Sockets.Naming.Host_Name

Return value
Name of the current host

Description
This function returns the name of the current host. Depending on the local configuration, it may or may not be a fully qualified domain name (FQDN).

Image (function)

Purpose Make a string from an address

Prototype

`function` **Image** (*Add* : `Address`) Sockets.Naming.Image
`return String;`

Parameters

Add	in	IP address

Return value
String representation of the IP address

See also [Value (function)], page 25.

Info_Of_Name_Or_IP (function)

Purpose Get addresses and names of a host

Prototype

`function` Sockets.Naming.Info_Of_Name_Or_IP
Info_Of_Name_Or_IP (*Something* : `String`)
`return Host_Entry;`

Parameters

Something	in	Host name or IP address

Return value
Corresponding host entry

Description
This function extracts all the names and addresses from the naming service.

Exceptions

`Naming_Error`	No information available for this name or address

Is_IP_Address (function)

PURPOSE Check if given string is a valid IP address

PROTOTYPE

```
function Is_IP_Address                Sockets.Naming.Is_IP_Address
    (Something : String)
    return Boolean;
```

PARAMETERS

Something	in	String to check

RETURN VALUE

'True' if *Something* is an IP address

Name_Of (function)

PURPOSE Official name of the host

PROTOTYPE

```
function Name_Of (Something : String)       Sockets.Naming.Name_Of
    return String;
```

PARAMETERS

Something	in	Host name or IP address

RETURN VALUE

Name of the host

EXCEPTIONS

`Naming_Error`	No information available for this name or address

SEE ALSO [Address_Of (function)], page 21.

Value (function)

PURPOSE Transform a string into an address

PROTOTYPE

```
function Value (Add : String)                Sockets.Naming.Value
    return Address;
```

PARAMETERS

Add	in	Textual representation of an IP address

RETURN VALUE

Corresponding Address

SEE ALSO [Image (function)], page 24.

Appendix A Contributors

AdaSockets has been originally developped by Samuel Tardieu who still maintains it. However, the following people have made crucial contributions to AdaSockets, be they new code, bug fixes or porting to new operating systems:

- Dmitriy Anisimkov (`anisimkov@yahoo.com`)
- Alan Barnes (`barnesa@aston.ac.uk`)
- Juanma Barranquero (`lektu@terra.es`)
- Bobby D. Bryant (`bdbryant@mail.utexas.edu`)
- Sander Cox (`sander.cox@philips.com`)
- Sune Falk (`sune.falck@telia.com`)
- Guillaume Foliard (`guifo@wanadoo.fr`)
- Laurent Guerby (`guerby@club-internet.fr`)
- David J. Kristola (`David95037@aol.com`)
- Dominik Madon (`dominik@acm.org`)
- Pascal Obry (`p.obry@wanadoo.fr`)
- Nicolas Ollinger (`Nicolas.Ollinger@ens-lyon.fr`)
- Stphane Patureau (`spaturea@meletu.univ-valenciennes.fr`)
- Thomas Quinot (`thomas@cuivre.fr.eu.org`)
- Preben Randhol (`randhol@pvv.org`)
- Maxim Reznik (`max1@mbank.com.ua`)
- Samuel Tardieu (`sam@rfc1149.net`)

If you feel that you have been forgotten, please send me a mail so that I can fix it in the next version.

See Appendix B [Resources on the Internet], page 29, for how to contribute.

Appendix B Resources on the Internet

The latest version of AdaSockets can always be found at:

`http://www.rfc1149.net/devel/adasockets`

There is a mailing-list that you can use to discuss problem or suggestions at:

`http://www.rfc1149.net/lists/info/adasockets`

Please use the mailing-list for questions and discussions. However, you can send me patches by private mail (`sam@rfc1149.net`).

Index

P

R

S

T

U

V