ELF Communications
Macro Reference

This manual was produced using Applixware.

Printed: June 2010

axnet Macros

The following is a brief synopis of axnet commands:

AXNET_CLOSE_CHANNEL@
Closes a channel.

AXNET_OPEN_CHANNEL@
Opens a channel.

AXNET_READ@
Reads information into an ELF array.

AXNET_RPC@
Convenience function that opens a channel, sends data to the channel, and then closes the channel. If data is returned from the server, it will also be returned using this function.

AXNET_RPC_CHANNEL@
Invokes a function executing in a different task. This task can be an ELF or a C task and the task can be running on your machine or a different machine.

AXNET_SERVICES_LIST@
Returns a list of services registered with axnet.

AXNET_SERVICE_REGISTER@
Registers a service with axnet.

AXNET_SERVICE_UNREGISTER@
Removes knowledge of a service from an axnet server.

AXNET_START_SERVICE@
Starts a server on a machine.

Using axnet

axnet provides the backbone of a communications system for ELF-to-ELF and C-to-ELF RPC services. It does this by providing connection and network naming service for ELF and Applix*ware* tasks.
axnet is managed by the axnet process. All connections to tasks are made through this server. Connections are made to named tasks on the Applix*ware* network.
The axnet server is accessed from a standard Berkeley TCP/IP port using a machine name.

See Chapter 6, "Applix*ware* Communications," of the *System Administrator's Guide* for more information.

The AXNET_RPC_CHANNEL@ macro allows remote programs to directly call ELF macros. These macros can pass complex information to the server on the axnet machine and, likewise receive complex results from the server. All ELF-to-ELF RPCs must use axnet. Similarly, all RPCs sent from outside Unix programs to ELF will also use axnet.

Most ELF-to-C RPC's will not use axnet. Instead, they will use the more direct, RPC service.

What's in a name?

Within Applix*ware*, a task can use the AXNET_SERVICE_REGISTER@ function to register itself as an available server. This function sends an identification string to its machine's axnet server. While this name is completely arbitrary, only one task at a time can use this name. The task can also specify a password string required of all client tasks that wish to use the server.

The AXNET_SERVICES_LIST@ macro allows all programs to connect to any axnet server and obtain a list of the services registered to it. The only exception is that it will not list services that have been made private to a user.

Making Connections

Connections made through axnet are pipelined through a process to which axnet forks. (Use the AXNET_OPEN_CHANNEL@ macro to establish a connection to the server.)

When a client task (or program) connects to an axnet server task (or program), a connection is first made to the axnet server. The server then forks and the child, the *connection server*, connects to the server task. All messages between the client and server task are routed through the connection server. These messages are sent using AXNET_RPC_CHANNEL@. When either end drops the connection, the connection server goes away.

For More Information

axnet Functions
axnet Example
axnet Menu

axnet Functions

Two categories of axnet functions exist:

· Functions that manipulate the service list. These functions include:

  AXNET_SERVICES_LIST@
  AXNET_SERVICE_REGISTER@
  AXNET_SERVICE_UNREGISTER@
  AXNET_START_SERVICE@
  The first functions are performed by the server-side ELF application. The last one is performed by a client application.

· Functions that are used in communicating with the server application. These functions include:

[AXNET_CLOSE_CHANNEL@](#)
[AXNET_OPEN_CHANNEL@](#)
[AXNET_RPC@](#)
[AXNET_RPC_CHANNEL@](#)

After is a service is registered with axnet and is running on a host, the typical interaction pattern within the client is as follows:

```
AXNET_OPEN_CHANNEL@...
AXNET_RPC_CHANNEL@...
AXNET_RPC_CHANNEL@...
AXNET_RPC_CHANNEL@...
. . .
AXNET_CLOSE_CHANNEL@...
```

AXNET_CLOSE_CHANNEL@

Closes a channel

Format AXNET_CLOSE_CHANNEL@(channel)

Arguments

channel	The number of a channel returned to your program by the AXNET_OPEN_CHANNEL@ macro.

Description Closes a channel previously opened using AXNET_OPEN_CHANNEL@. After closing a channel, you will no longer be able to access information from it. The next time you invoke AXNET_OPEN_CHANNEL@ for this service, a different channel number will probably be used.

See Also [AXNET_OPEN_CHANNEL@](#)

[axnet Menu](#)

[Using axnet](#)

[axnet Example](#)

AXNET_OPEN_CHANNEL@

Opens a channel

Format num = AXNET_OPEN_CHANNEL@(hostname, service)

Arguments

hostname	The name of the machine upon which the program identified by service is executing.

	service	The name of the service as it was registered with AXNET_SERVICE_REGISTER@.

Description Opens a channel between the currently executing ELF task and another task. This other task can be a C language or ELF task. It can be executing on the same machine as your task or on a different machine.

After your program finishes interacting with service, it should use AXNET_CLOSE_CHANNEL@ to free the system resources being used to maintain the connection.

See Also AXNET_CLOSE_CHANNEL@

axnet Menu

Using axnet

axnet Example

AXNET_READ@

Reads a buffer of information

Format elfArray = AXNET_READ@(channel)

Arguments channel — The number of a channel returned to your program by the AXNET_OPEN_CHANNEL@ macro.

Description Reads a buffer of information from a channel into an ELF array.

See Also axnet Menu

Using axnet

axnet Example

AXNET_RPC@

Directs a function to be executed within a service

Format [value=]AXNET_RPC@(hostname, service, cmdCode, arg1[, arg2[, ...arg10]]])

hostname	The machine upon which the service is registered.
service	The name of the service as it was registered using AXNET_SERVICE_REGISTER@.

cmdCode	Either the name of an ELF function if the service is an ELF program, or a function constant within a C program's **AxDispatch** function.
arg1..arg10	An optional list of values used by the function named in cmdCode.

Description The AXNET_RPC@ macro causes an action to be performed within the task named by service. If that macro returns a value, it is also returned by this macro.

AXNET_RPC@ performs the work of the following three macro:

- **AXNET_OPEN_CHANNEL@**
- **AXNET_RPC_CHANNEL@**
- **AXNET_CLOSE_CHANNEL@**

You would use this macro instead of the individual macros if you only need to connect with the service for one operation. (As there is a performance penalty for repeatedly opening and closing the service, you would not want to use this macro to perform a series of activities.)

If your ELF program is communicating with another ELF program, you would name the ELF function within the service using the cmdCode argument. However, if your ELF program is communicating with a C language program, cmdCode is an integer constant that is used within the AxDispatch function to determine what function should be executed.

See Also **AXNET_RPC_CHANNEL@**

axnet Menu

Using axnet

axnet Example

AXNET_RPC_CHANNEL@

Directs a function to be executed

Format [value=] AXNET_RPC_CHANNEL@(channel, cmdCode[, arg1[, arg2[, ...arg10]]])

Arguments

channel	A channel previously opened using AXNET_OPEN_CHANNEL@.
cmdCode	Either the name of an ELF function if the service is an ELF program, or a function constant within a C program's **AxDispatch** function.
arg1..arg10	An optional list of values used by the function named in cmdCode.

Description Tells a function within an axnet service that it should execute. If the function requires arguments, they are provided in the arguments following cmdCode.

Before using this macro, you must open a channel to the task using the AXNET_OPEN_CHANNEL@ macro.

This macro and AXNET_RPC@ perform similar functions. This macro should be used when your ELF program will be requiring the service to perform a number of activities. In contrast, use AXNET_RPC@ when the service is only performing one activity. While AXNET_RPC@ can be easier to use, it has the additional overhead of opening and closing the channel.

If your ELF program is communicating with another ELF program, you would name the ELF function within the service using the cmdCode argument. However, if your ELF program is communicating with a C language program, cmdCode is an integer constant that is used within the AxDispatch function to determine what function should be executed.

See Also AXNET_RPC@

axnet Menu

Using axnet

axnet Example

AXNET_SERVICES_LIST@

Returns a list of services

Format array = AXNET_SERVICES_LIST@(hostname)

Arguments hostname The name of a server accessible by your machine.

Description Returns a list of the services that are currently registered with an axnet server. As services can be added and removed dynamically, this list will change with time.

See Also AXNET_SERVICE_REGISTER@

axnet Menu

Using axnet

axnet Example

AXNET_SERVICE_REGISTER@

Adds a service to axnet

Format AXNET_SERVICE_REGISTER@(serviceName, macroNames[, isPrivateFlag[, passwd[, prefix]]])

Arguments

serviceName	An arbitrary name assigned by your program to a group of macros within an ELF program.
macroNames	An array of macro names or function define constants.
isPrivateFlag	An optional true/false value indicating if the macros and functions within the service can be used by other people. TRUE means that the functions are private.
passwd	An optional value that specifies the password that must be used when using the service. (This option is not supported at this time.)
prefix	An optional value which can be placed in front of each element in the macroNames array.

Description Registers a service name with the axnet daemon running on your system. (Remote registration is not supported.) *Registering* a service means that you have associated a series of functions with a name and are making the name available so that other tasks can access these functions.

The macros being registered are macros within an ELF program. Only the functions listed here can be invoked remotely.

A service can be registered even if the tasks associated with the service's task are not running.

See Also **AXNET_SERVICE_UNREGISTER@**

axnet Menu

Using axnet

axnet Example

AXNET_SERVICE_UNREGISTER@

Removes information about a service

Format AXNET_SERVICE_UNREGISTER@(passedName)

Arguments passedName

The name of a service that is currently registered with an axnet daemon.

Description This function removes the information stored with the axnet server that describes one service. This should be one of the last actions performed by your ELF server program as it shuts down.

See Also **AXNET_SERVICE_REGISTER@**

axnet Menu

Using axnet

axnet Example

AXNET_START_SERVICE@

Starts a server program

Format AXNET_START_SERVICE@(hostname, service, pathname[, argumentList])

Arguments

hostname	The name of the machine upon which the service resides.
service	The name of the service. If the service is not registered, this macro will register it.
pathname	The pathname of the program containing the functions registered with the server.
argumentList	Arguments required by program pathname when it starts.

Description The AXNET_START_SERVICE@ macro instructs machine hostname that it should begin executing the specified service. It is assumed that this program sits and waits for a command from a client process.

This command is usually followed by an AXNET_OPEN_CHANNEL@ macro.

See Also **AXNET_SERVICE_REGISTER@**

axnet Menu

using axnet

axnet Example

AXNET_START_SLAVED_PROCESS@

Starts a private server

Format channel = AXNET_START_SLAVED_PROCESS@ (hostName, pathName[, argList])

Arguments		
	hostName	The machine upon which the service is registered.
	pathName	The pathname of the program containing the functions registered with the server. (If the program is in the same directory, you can pass *name instead.)
	argList	An optional list of arguments that are required by program pathName.

Description Starts a private server. This macro should be used instead of the following sequence:

AXNET_START_SERVICE@(hostname, service, pathname, "-slave")
AXNET_OPEN_CHANNEL@(hostname, service)

The returned channel number can be used as the channel value in the RPC functions. In many ways, this macro is analogous to RPC_START_SERVER@.

AXNET_TEST@

Tests to see if Axnet is running on the target server

Format channel = AXNET_TEST@ (hostName)

Arguments hostName The machine to query for a running axnet process.

Description Tests the target hostName for a running axnet process. If axnet is running on the target machine, a message box appears saying that axnet is running.

This macro is used in situations where axnet is used to launch another process. If the process is not starting properly, you can use this test to see if axnet is running on the remote machine. If axnet is running correctly, the problem may be with another piece of software.

For example, suppose you are using axnet to launch a database server. If you receive the message "cannot launch gateway on server," then you should run axnet_test@ on the server to see if axnet is operational. If so, the problem may be with the gateway server software, or the database server.

Axnet Example

```
/*****************************************************************
 * AXNET_SERVER
 *       Create server "demo", respond to simple messages
 *       Note the real service name is "<username>:demo".
 *****************************************************************/
macro AXNET_SERVER
        var info

        info = "demo_macro","rpc_pong"
        AXNET_SERVICE_REGISTER@("demo",info)
endmacro

/*****************************************************************
 * AXNET_DROP_SERVER
 *       Convenience function for dropping the server
 *****************************************************************/
macro AXNET_DROP_SERVER
        AXNET_SERVICE_UNREGISTER@("demo")
endmacro


/*****************************************************************
 * DEMO_MACRO
 *       The Server side function. This function will
 *       respond to one of three "messages":
 *               ping, pong, or file.
 *       Anything else is an error.
 *****************************************************************/
macro DEMO_MACRO(cmd,data1,data2)
        if cmd = "ping"
                return
        if cmd = "pong"
                return(data1)
        if cmd = "file"
        {
                WRITE_BINARY_FILE@(data1,data2)
                return(data2)
        }
        ERROR@(1,"Not a known demo command",cmd)
endmacro
```

```
/*****************************************************************
 * AXNET_CLIENT
 *       Send messages to the demo server
 ****************************************************************/
macro AXNET_CLIENT(hostname,username)
        var service
        var stuff
        var result

        service = username ++ ":demo"
'
' Send message "ping". There is no response.
'
        AXNET_RPC@(hostname,service,"demo_macro","ping")

'
' Send message "pong". The result should be the name
' of the service.
'
        result = AXNET_RPC@(hostname, service,
                        "demo_macro","pong",service)
        if result <> service
                error@("pong failed")

'
' Send message "pong". The result should be the string
' sent as the fourth argument.
'
        result = AXNET_RPC@(hostname,service,
                        "rpc_pong","Hello World!")
        if result <> "Hello World!"
                error@("rpc_pong failed")

'
' Send message "file". The file should be created.
' The information written to the file should be
' returned.
'
        stuff = READ_BINARY_FILE@("/etc/passwd")
        result = AXNET_RPC@(hostname,service,
                        "demo_macro","file",
                        "/tmp/zztest",stuff)
        if result <> stuff
```

```
                error@("file xfer failed")

'
' Send message "make_error". An error message should be
' thrown.
'
        stuff = AXNET_RPC@(hostname, service,
                        "demo_macro","make_error")
endmacro

/****************************************************************
 * RPC_PONG
 *      Simply returns the string sent to it
 ****************************************************************/
macro RPC_PONG(str)
        return(str)
endmacro
```

For more information see:

axnet Menu

RPC_CALL@

Invokes an RPC function

Format [value=] RPC_CALL@(programName, cmd, data [, socketName[, hostMachine]])

Arguments

programName	The name of the C language program containing the AxDispatch function.
cmd	An integer number used to identify the function within programName being invoked.
data	The information required by the function identified by cmd. data can be an array if more than one piece of information is required.
socketName	An optional port name or a path name.
hostMachine	The optional name of the machine upon which programName is executing.

Description Sends information to a server and, if data is sent back, it is returned by RPC_CALL@. This function first performs an **RPC_CONNECT@**. (Go to that topic to see how the connection arguments are used.) After connecting, two kinds of information are sent from the client process to the server process, which is assumed to be a program written in the C language. The first is an integer command code (cmd). This code is used by case statement within a function that must be named AxDispatch within the C program.

The channel remains open after the RPC_CALL@ macro executes. Whenever an RPC_CALL@ executes, it checks to see if the channel is open.

Here is how AxDispatch is declared:

```
elfData AxDispatch (funcId, funcData)
int funcId;
elfData funcData;
```

The case statement would be something like:

```
switch (funcID)
{
 case FUNCTION1:
       function1(funcData);
       break;
case FUNCTION2:
       function2(funcData);
       break;
}
```

See Also **RPC_CONNECT@**

RPC_DISCONNECT@

RPC_CHANNEL_CALL@

Invokes an RPC function

Format [value=] RPC_CHANNEL_CALL@(channel, cmd, data)

Arguments

channel	The channel number returned by **RPC_CONNECT@**.
cmd	An integer number used to identify the function within programName being invoked.
data	The information required by the function identified by cmd. data can be an array if more than one piece of information is required.

Description Sends information to a server and, if data is sent back, it is returned as value. The function to which this information is sent is identified by the cmd integer value. (This command code is used by case statement within a function that must be named AxDispatch within the C program. For more information, see RPC_CALL@.) If this function requires data to be sent to it, this data is sent as data.

This command requires that you open a channel using RPC_CONNECT@ and that the channel is open.

This command differs from RPC_CALL@ in that RPC_CALL@ begins by opening (or reestablish a connection) to a channel. Because the channel is already open, RPC_CHANNEL_CALL@ is faster than RPC_CALL@.

RPC_CHANNEL_MASTER@

Sets channel existence state when a task goes away

Format RPC_CHANNEL_MASTER@(channel, taskID)

Arguments

channel	The channel number returned by RPC_CONNECT@.
taskID	The ID of an executing task.

Description Indicates that the channel is disconnected when a task stops executing. The exception is when you set taskID to 0, which couples it to no task. In this case, the channel will remain even if the task goes away. Normally, you would only use this macro when you want a channel to a running server process to remain open at all times.

RPC_CHANNEL_USE_HOURGLASS@

Displays hourglass for a task

Format RPC_CHANNEL_USE_HOURGLASS@(channel, hourglassFlag)

Arguments

channel	The channel number returned by RPC_CONNECT@.
hourglassFlag	A Boolean value which if set to TRUE tells ELF that it should display an hourglass.

Description Displays an hourglass for the task. Normally, you would use this function for slow channels that are performing channel I/O.

RPC_CONNECT@

Connects to a running process

Format channel = RPC_CONNECT@(programName[, socketName[, hostMachine]])

Arguments

programName	The name of the C language program containing the AxDispatch function.
socketName	An optional portname or a path name.
hostMachine	The optional name of the machine upon which programName is executing.

Description Either creates the initial connection between a client and a server process or just returns the channel of an existing connection. The arguments are used as follows:

Program Name	Socket Name	Machine	Meaning
NULL	NULL	NULL	Invalid
String	NULL	NULL	Name is a program name. Socket is slave on current machine
String	String	NULL	Name is any handle. Socket is a server.
String	NULL	String	Uses Applix socket name. Socket is a server
String	String	String	Uses Customer socket name. Socket is a server

The channel number is returned.

All RPC channels (including those used by the SQL macros) disappear when the creating task goes away.

See Also RPC_CALL@

RPC_DISCONNECT@

RPC_DISCONNECT@

Disconnects from a running process

Format RPC_DISCONNECT@(programName)

Arguments

programName	The name of a program to which a connection was made.

Description Removes the connection made previously to programName.

See Also RPC_CALL@
RPC_CONNECT@

RPC_START_SERVER@

Starts a server process

Format channel = RPC_START_SERVER@(programName[, socketName[, machine]])

Arguments

programName	The name of the C language program containing the AxDispatch function.
socketName	A portname or a path name.
machine	The name of the machine upon which the program is executing.

Description Either creates the initial connection to a server or returns the channel of an existing connection. While this macro is similar to RPC_CONNECT@, it differs in that it will always obtain a new server. (RPC_CONNECT@ will connect to an existing server.)

For information on this macro's arguments, see RPC_CONNECT@.

SOCKET_CLOSE@

Closes the specified TCP/IP socket on a server

Format SOCKET_CLOSE@(num)

Arguments

num	The TCP/IP socket ID as assigned to the socket when it was opened.

Description Closes the socket specified by num. SOCKET_CLOSE@ should be used to close a socket on a server, not a client socket. Any attempts to communicate to a socket that has been closed will throw an error.

SOCKET_CLOSE_CHANNEL@

Closes the specified TCP/IP network channel

Format SOCKET_CLOSE_CHANNEL@(channel)

Arguments	channel	The channel ID to close. The channel must have been previously opened using SOCKET_OPEN_CLIENT@ or SOCKET_OPEN_SERVER@.

Description Closes the socket channel specified by channel. Once channel is closed, any attempts to communicate over channel will throw an error.

SOCKET_OPEN_CLIENT@

Opens a socket on a TCP/IP network

Format SOCKET_OPEN_CLIENT@ (num, timeout, hostname)

Arguments	num	The socket ID of the socket to which to connect.
	timeout	Reserved for future use. You must specify -1 for this argument.
	hostname	The server name to which client messages should be sent.

Description Opens a socket on the machine from which it is called. SOCKET_OPEN_CLIENT@ opens a channel to the server identified by hostname and returns the channel ID. The channel remains open until it is closed using SOCKET_CLOSE_CHANNEL@. If host-name cannot be found, an error is thrown.

A system designated as a client uses socket communications to send messages to the system designated as server.

See also SOCKET_OPEN_SERVER@

SOCKET_CLOSE_CHANNEL@

SOCKET_OPEN_SERVER@

Opens a socket on a TCP/IP network

Format channelID = SOCKET_OPEN_SERVER@ (channelName, timeout[, phase])

Arguments	channelName	The TCP/IP socket ID, which is a unique number assigned to the socket being opened. Check the /etc/services directory for socket numbers already used by the system. If you use the name of a file instead of a number, a named pipe is created rather than a TCP/IP socket. In this case, you must use an absolute path name.
	timeout	Reserved for future use. You must specify -1 for this argument.

phase	Indicates how the macro waits for the socket to be created. Use one of the following two values:
	1 Creates the socket. After the macro returns, other UNIX processes can connect to this socket.
	2 Specifies that the task will pend, waiting for a connection.

Description Opens a socket on the machine from which it is called. A socket opened by SOCKET_-OPEN_SERVER@ pends until a client socket is opened, at which time SOCKET_-OPEN_SERVER@ returns a channel ID. The system designated as the server uses socket communications to receive messages from client machines rather than for sending messages.

A channel remains open until it is closed using SOCKET_CLOSE_CHANNEL@.

A typical code sequence used when creating real-time macros is as follows:

```
channel = SOCKET_OPEN_SERVER@
              (channelName, -1, 1)
stuff = channeLName, readerID

RPC_CHANNEL_CALL@
              (rtChannel, RT_OPEN_CHANNEL_, stuff)
channel = SOCKET_OPEN_SERVER@
              (channelName, -1, 2)
```

See also SOCKET_OPEN_CLIENT@

SOCKET_CLOSE_CHANNEL@

SOCKET_READ@

Reads the ELF array sent over a TCP/IP network channel

Format array = SOCKET_READ@(channel, timeout)

Arguments

channel	The socket channel that is to receive the array. The channel must have been previously opened using SOCKET_OPEN_CLIENT@ or SOCKET_OPEN_SERVER@.
timeout	Reserved for future use. You must specify -1 for this argument.

Description Used to read the ELF string array sent over a TCP/IP network. Typically, SOCKET_READ@ is used by a server on the network to read a message sent by a client machine on the network. SOCKET_READ@ returns the ELF array received over the channel.

SOCKET_READ_BINARY@

Receives an ELF binary object over a TCP/IP network channel

Format data = SOCKET_READ_BINARY@(channel, timer, maxbytes)

Arguments

channel	The channel over which to receive the binary object. The channel must have been previously opened using **SOCKET_OPEN_CLIENT@** or **SOCKET_OPEN_SERVER@**.
timer	A number indicating the amount of seconds SOCKET_READ_BINARY@ should read data. -1 indicates that no time limit is set for a data read.
maxbytes	The maximum number of bytes to be read by SOCKET_READ_BINARY@.

Description Receives an ELF binary object sent over a TCP/IP network channel by **SOCKET_WRITE_BINARY@**. SOCKET_READ_BINARY@ returns a binary object. The size of this object equals the amount of bytes read. If no bytes were read, NULL is returned.

SOCKET_READ_BINARY@ reads data until one of the following conditions is met:

· The number of seconds specified by timer is reached.

· The maximum number of bytes specified by maxbytes is read.

Typically, a timer is used when you are not sure of the number of bytes that will be received.

SOCKET_RPC_READ@

Reads a message from a socket into an ELF array

Format SOCKET_RPC_READ@(uid, timer)

Arguments

channel	The channel over which to receive the binary object. The channel must have been previously opened using **SOCKET_OPEN_CLIENT@** or **SOCKET_OPEN_SERVER@**.
timer	A number indicating the amount of seconds SOCKET_READ_BINARY@ should read data. -1 indicates that no time limit is set for a data read. -2 indicates that the entire buffer is read as a message

Description Reads a message from a socket into an ELF array. This data must be built as an ELF array within your C language program. The data is placed on the socket using standard C socket calls.

This macro, while supported, is obsolete.

SOCKET_WRITE@

Sends an ELF array over the specified TCP/IP network channel

Format SOCKET_WRITE@(channel, array)

Arguments

channel	The channel over which to send the array. The channel must have been previously opened using SOCKET_OPEN_CLIENT@ or SOCKET_OPEN_SERVER@.
array	An ELF string array.

Description Sends the specified ELF string array over a TCP/IP network channel. Typically, SOCKET_WRITE@ is used by a client machine on the network to send a message to the network server.

SOCKET_WRITE_ASCII@

Sends ASCII data over a TCP/IP network channel

Format SOCKET_WRITE_ASCII@(channel, string)

Arguments

channel	The channel over which to send the ASCII data. The channel must have been previously opened using one of the following: · SOCKET_OPEN_CLIENT@ · SOCKET_OPEN_SERVER@
string	The ASCII string to send.

Description Sends a string of ASCII bytes over a TCP/IP network channel.

SOCKET_WRITE_BINARY@

Sends an ELF binary object over a TCP/IP network channel

Format SOCKET_WRITE_BINARY@(channel, object)

Arguments	channel	The channel over which to send the binary object. The channel must have been previously opened using one of the following: · **SOCKET_OPEN_CLIENT@** · **SOCKET_OPEN_SERVER@**
	object	The binary object to send.

Description Sends an ELF binary object over a TCP/IP network channel.