SIEMENS

SIMATIC NET

SOFTNET-S7/UNIX

User Manual V2.0

August 1999 **Update:** C79000-B8976-C058/01

Note

We would point out that the contents of this product documentation shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statement contained in this documentation do not create new warranties or restrict the existing warranty.

We would further point out, that for reasons of clarity, these operating instructions cannot deal with every possible problem arising from the use of this device. Should you require further information or if any special problems arise which are not sufficiently dealt with in the operating instructions, please contact your local Siemens representative.

General

This device is electrically operated. In operation, certain parts of this device carry a dangerously high voltage.

WARNING !

!

Failure to heed warnings may result in serious physical injury and/or material damage.

Only appropriately qualified personnel may operate this equipment or work in its vicinity. Personnel must be thoroughly familiar with all warnings and maintenance measures in accordance with these operating instructions.

Correct and safe operation of this equipment requires proper transport, storage and assembly as well as careful operator control and maintenance.

Personnel qualification requirements

Qualified personnel as referred to in the operating instructions or in the warning notes are defined as persons who are familiar with the installation, assembly, startup and operation of this product and who possess the relevant qualifications for their work, e.g.:

- Training in or authorization for connecting up, grounding or labeling circuits and devices or systems in accordance with current standards in safety technology;
- Training in or authorization for the maintenance and use of suitable safety equipment in accordance with current standards in safety technology;
- First Aid qualification.

Contents

1 Introduction to SOFTNET-S7/UNIX

In industrial production, there are numerous hardware platforms with a wide variety of properties, for example, numeric controls, robot controls, programmable logic controllers, production controllers etc. These devices are networked with powerful network components and suitable communications hardware and software.

SOFTNET-S7/UNIX provides the S7 programming interface for UNIX computer systems.

The software is currently available for the following operating systems:

- Solaris Sparc
- Solaris X86
- SCO UNIX
- HP-UX
- LINUX

☞ **You will find a detailed description of the basic ideas and concepts of the S7 programming interface in the “S7 Programming Interface" manual. If you are a first time user, we strongly recommend that you read this section of the documentation.**

SOFTNET Architecture

SOFTNET-S7/UNIX consists of a total of three software components:

➢ SAPI-S7	SOFTNET-S7/UNIX Programming Interface
➢ CMX	Communication Manager UNIX, access to communication layer 4, addressing using logical names
➢ CCP-TP4	Transport provider with connectionless network

The following diagram is an overview of the product structure of SOFTNET-S7/UNIX.

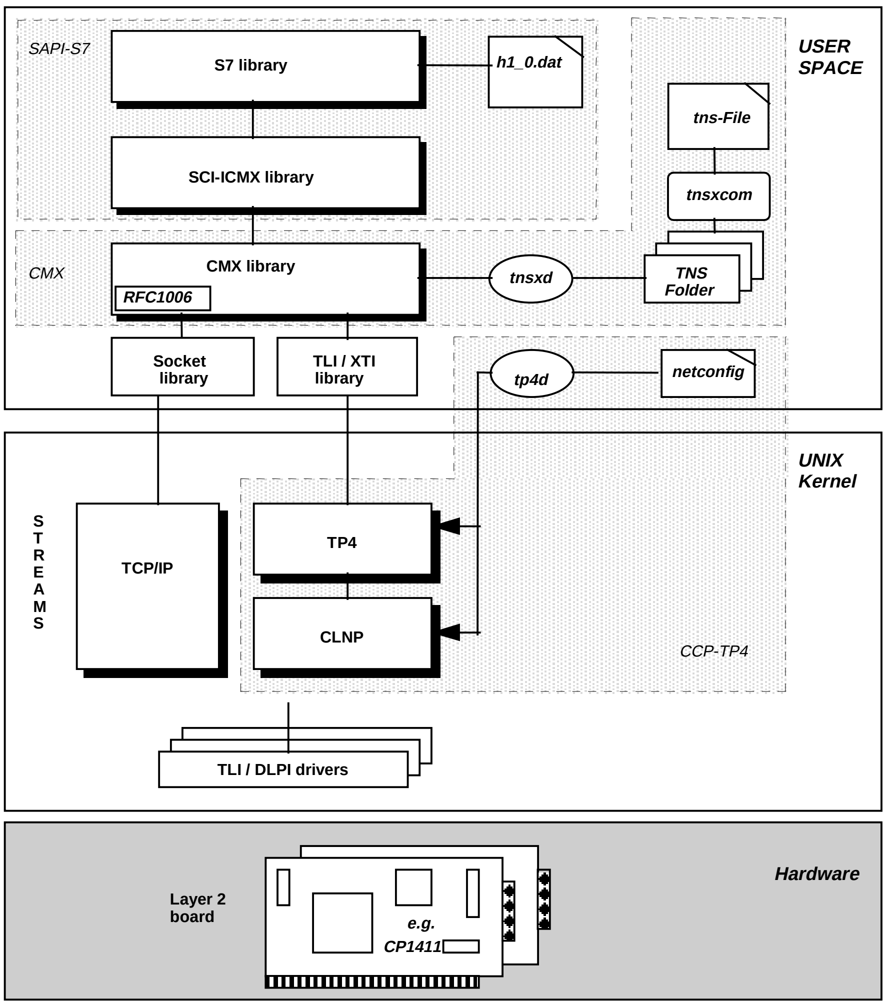

Figure 1.1: SOFTNET-S7/UNIX Architecture

SAPI-S7

The SAPI-S7 software package provides the SOFTNET-S7 library containing the coding for S7 function calls that you can use via the S7 programming interface (C programming language).

The SOFTNET-S7 library accesses the configuration files h1_x.dat. . Where “x" stands for <0> to the <number of communication boards being used -1>. This means that there is a configuration file for each communication board. The assignment of the application association names to the connection endpoints is set in these configuration files. You should bear in mind that the end points are logical names in the TNS directory and that the real addresses must be assigned to the names using the tnsxcom tool (configuration).

Access to the transport system (CMX) is via an SCI-CMX converter that maps the SCI calls of the SOFTNET-S7 library on suitable CMX calls.

Relevant components

Module	Function
/usr/lib/libs7h1.so or libs7h1.sl /usr/lib/libs7h1.a	S7 programming interface
/usr/lib/libsci_cmx.so or libs7h1.sl /usr/lib/libsci_cmx.a	SCI-CMX converter library
/usr/include/s7/sapi_s7.h	Definition of the S7 programming interface
/usr/s7/example/h1_0.dat	Sample S7 configuration file
/usr/s7/example/tns_inp.dat	Sample TNS configuration file
/usr/s7/example/xs7_bsp.c	Sample S7 client application
/usr/s7/example/makefile	Sample makefile for S7 client application
tping	Tool for testing transport connections

CMX

The CMX communication manager contains the CMX library. The library provides calls for layer 4 communication in which the stations can be addressed using logical names. During operation, the logical names are assigned real addresses (T selector, MAC address) and communication takes place based on the TLI/XTI interface.

The assignment of the logical names to the real addresses is handled by the background process *tnsxd* in the TNS directory (Transport Name Service). The assignment can be made during installation and startup by the user editing an ASCII file (tns file) that is then transferred to the *tnsxcom* tool. *tnsxcom* runs a syntax and consistency check and enters the assignments in the TNS directory.

Relevant components

Module	Function
/usr/lib/libcmx.so or libcmx.sl tnsxd tnsxcom tnsxchk cmxl	CMX library TNS daemon TNS compiler Check TNS directory Trace evaluation of CMX (Activated using the environment variable CMXTRACE)

CCP-TP4

The CCP-TP4 software package provides the protocol implementations of the transport (layer 4) and network (layer 3) layers according to the 7-layer ISO/OSI reference model. Internally, it uses interfaces such as LLI/DLPI and TLI/XTI CCP-TP4 that were defined by AT&T. Some of these interfaces were included in the Xopen standard and are available for almost all UNIX systems. CCP-TP4 consists of the STREAMS multiplexers TP4 (Transport Provider Class 4) and CLNP (ConnectionLess Network Protocol) and the background process *tpd4*.

TP4 contains the protocol implementation for the transport layer, class 4. Since TP 4 is implemented as a STREAMS multiplexer, it is linked into the UNIX kernel as a separate entity. As its upper access, it has the TPI (Transport Provider Interface) and can therefore be addressed via the standard interfaces available in the UNIX system, TLI (Transport Layer Interface)/XTI (Xopen Transport Interface).

TP4 uses the network provider CLNP via the NPI (Network Provider Interface). CLNP is also a STREAMS multiplexer. It contains the protocol implementation of the network layer of which SOFTNET-S7/Unix uses the inactive mode.

The CLNP accesses the network via the LLI (Logical Link Provider Interface) / DLPI (Data Link Provider Interface), the layer 2 access to the network adapters of the system. The Ethernet interfaces existing in the system are also addressed via the LLI/DLPI interface. Although, in contrast to TCP/IP, the CLNP uses the LLI/DLPI interface not in the Ethernet but rather in the ISO 802.3 mode, parallel operation of both protocols via the same hardware interface is possible.

The background process tp4d creates the STREAM for the communications protocol consisting of CLNP and TP4. Based on the configuration file netconfig, it also sets communications parameters that are independent of the particular connection. "netconfig" contains the protocol profile for SIMATIC NET that is matched to the protocol conventions of SOFTNET-S7/UNIX and must not be modified.

Relevant components

Module	Function
tp4d tp4stat	TP4 daemon Provides statistical information

2 Creating Applications

2.1 Installation

How to install SOFTNET-S7/UNIX is described in detail in the product information for the operating system you use. When installing SOFTNET-S7/UNIX, follow the instructions in the product information.

Parallel Operation of SOFTNET-S7/UNIX and SOFTNET-TF/UNIX

Parallel operation of SOFTNET-S7/UNIX and SOFTNET-TF/UNIX is possible. If you intend to run these packages simultaneously, only install the CMX and CCP-TP4 packages once.

2.2 Programming Environment

The functions of the S7 programming interface are provided as C calls in the library *libs7h1.so*.

The special definitions for the S7 programming interface are located in header files that the application must include using the *#include* instruction. The following header files are relevant:

Header Files	/usr/include/s7/sapi_s7.h	This header file contains the prototypes of the S7 function calls. It must be included in all modules that contain the S7 programming interface.

Deselecting the Prototype Test

The SOFTNET-S7/UNIX include file sapi_s7.h lists all the prototypes of the functions of the S7 programming interface. If this include file is linked, the function calls you use in your SOFTNET-S7/UNIX application are compared with the function prototypes contained in this file. Since not every C compiler supports function prototypes, the function prototypes in the include file can be suppressed with the following switch:

```
-DS7_NOPROTOTYPES
```

If this switch is set when you compile your SOFTNET-S7/UNIX application, the function prototypes are not checked.

Linking the Applications

The SOFTNET-S7/UNIX user interface calls are located as separate modules in libraries. Each application links these library modules automatically into the executable program.

For more detailed information about link instructions, refer to the product information for the operating system you are using.

Example of an Application

After you have installed SOFTNET-S7/UNIX, there is an example of a client application and configuration files and a makefile in the following directory:

/usr/s7/example

2.3 Differences Compared with the S7 Programming Interface

The S7 programming interface is defined without reference to a particular system and is described in detail in the “S7 Programming Interface" manual. For more detailed information, refer to this manual.

SOFTNET-S7/UNIX differs from the description in the “S7 Programming Interface" manual in the points listed below.

For details of other supplementary functions or restrictions, please refer to the SOFTNET-S7/UNIX product information for the operating system you are using.

s7_init()

The name of the CP-specific file from which the configuration information is read during the logon, does not end as described in the “S7 Programming Interface" manual with the extension “.LDB". Here, the extension “.dat" is used. Refer also to Section 3.7 of this document “Configuring VFDs and the Connection Names Assigned to Them" in which the configuration of the CP-specific configuration files is described.

The name of the CP-specific configuration file can be assigned freely using environment variables. These environment variables are also described in Section 3.7 of this document “Configuring VFDs and the Connection Names Assigned to Them".

Important: If you use more than one VFD in one or more communications processors at the same time, please read Section 3 of this document completely “Installing and Starting Up Applications", in particular Section 3.8 “Notes on Configuration".

Byte Alignment

The SAPI-S7 programming interface can only operate when the variables are stored byte-aligned in memory. This means that no padding bytes can be inserted, for example between individual components of a structure. This is achieved by appropriate compiler options or by pragmas.

Network Representation of the Variable Values

The SAPI-S7 library supplies the read variable values and expects the variable values to be written in network representation (see also the “S7 Programming Interface" manual Section 7.2 “Representation of S7 Variables".

Depending on the host CPU you are using and the operating system, the variable values must be converted from the host to the network representation before they are sent and from the network to the host representation after they are received, as follows:

HPUX:	Host representation corresponds to network representation. No conversion necessary.
Solaris SPARC:	Host representation corresponds to network representation. No conversion necessary.
Solaris X86:	Host representation corresponds to the Intel CPU format. Conversion of the host representation to the network representation and vice-versa is necessary.
SCO:	Host representation corresponds to the Intel CPU format. Conversion of the host representation to the network representation and vice-versa is necessary.

2.4 Additional Function "s7_getfds()"

New S7 Function s7_getfds()

The *s7_getfds()* call returns the file descriptors used by the S7 programming interface. These can be used in the *poll()* system call. This allows user programs to wait for further external events in addition to communications events. If the *poll()* system call recognizes events on file descriptors, *s7_receive()* must then be called. Remember that the *s7_receive()* call can return S7_NO_MSG.

The file descriptor assigned to an S7 connection is only known after the S7 connection has been established. This means that poll() can only be used to receive events on S7 connections that are already established at the time of the s7_getfds() call (in other words the file descriptor is known).

The file descriptors used by the S7 programming interface must be checked before every *poll()* since they can change dynamically.

int s7_getfds (struct pollfd **poll_fdes, int *nfdes)

Parameter: poll_fdes: In **poll_fdes**, the caller transfers a pointer containing a pointer to a structure **poll_fdes**.

After successful execution, the caller receives a pointer to an array of structures of the type pollfd. This array contains all the file descriptors currently being used by SOFTNET-S7/UNIX. The array contains an element of the pollfd structure for every file descriptor used by SOFTNET-S7/UNIX. The number of elements of this array is returned in nfdes.

The pollfd structure contains the following elements:

```
int fd;              /* file descriptor */
short events;        /* requested events */
short revents;       /* returned events */
```

For further information, refer to the UNIX Programmer's Reference Manual description of the UNIX system call poll().

nfdes: Pointer to an integer variable.

After successful execution, this integer variable contains the number of elements in the poll_fdes array. The number of elements in the poll_fdes array corresponds to the number of file descriptors currently being used by SOFTNET-S7/UNIX.

Return:	0:	Call is ok: the file descriptors have not changed since the last call
	1:	Call is ok: the file descriptors have changed since the last call.
	-1:	Error occurred, repeat the call, if necessary several times.

If you do not want the user program to wait for additional events, the return values of *s7_getfds()* can be transferred directly to the *poll()* system call.

Example of s7_getfds()

```
struct pollfd     *fdes;
int               nfdes;
int               timeout = 5000; /*5 seconds */
int               ret;


...

if (s7_getfds (&fdes, &nfdes) >= 0)
{
      /* preparation before calling poll */

      ...

      ret = poll(fdes, nfdes, timeout);

      if (ret > 0)
      {
            /* event received  */
            /* call s7_receive */

            ...
      }
      else if (ret = 0)
      {
            /* timeout when calling poll */

            ...
      }
      else
      {
            /* poll returned an error */

            ...
      }
}

...
```

For a detailed description of the system call poll(), please refer to the UNIX Programmer´s Reference Manual.

In the example above, additional file descriptions may be contained in the fdes array after the s7_getfds() call and before the poll() call with which, for example, entries made by a user on the user interface can be received.

With s7_getfds() and poll() you can therefore implement a common waiting point within a UNIX process for all events that can be received in the particular UNIX process.

2.5 Difference Compared with the "S7 Programming Interface" Manual

Field-Oriented Services

Functions s7_bsend_req, s7_brcv_init, s7_brcv_stop :

The "r_id" parameter has a value range of 1 to FFFF FFFF and not 0 to FFFF FFFF as described.

3 Installing and Starting Up Applications

General

SOFTNET-S7/UNIX applications use the functions of the S7 programming interface for communication with remote systems (for example S7 programmable controllers). Communication is based on the following concept: The S7 programming interface uses the term VFD (Virtual Field Device) (see also /1/ “S7 Programming Interface"). An S7 application logs on at one or more VFDs. A list of connections is available for the logon at a VFD. On the S7 programming interface, freely selectable connection names are used. Each individual connection name describes exactly one S7 connection between two end systems. This connection name is used to access the required address information contained in the transport name service (TNS) of CMX. The assignment of the connection names to the actual address parameters of the partner applications is then made while the program is running.

To operate SOFTNET-S7/UNIX applications, two types of configuration file are therefore required:

- ➢ Configuration files containing a list of configured VFD names. These configuration files also contain the connection names assigned to a VFD name.
- ➢ Configuration files that contain the physical addresses of the S7 connection names: these are in files containing the TNS entries. Using these files, the address information is written to the transport name service and then can be accessed by a SOFTNET-S7/UNIX application while it is running.

3.1 The Transport Name Service TNS

Transport Name Service (TNS)

Each network and each transport system demands special address information to be able to address a communications partner. With SOFTNET-S7/UNIX, the ***Transport Name Service TNS*** is used with which the names and addresses of SOFTNET-S7/UNIX applications can be managed.

TNS reads the address information from a directory, the ***TS directory*** (Transport System Directory). This TS directory contains the address information of every SOFTNET-S7/UNIX application under a symbolic name known as the ***global name***. The TS directory must contain information about all SOFTNET-S7/UNIX applications resident on the local system and about the potential communication partners on remote systems.

A SOFTNET-S7/UNIX application operates on the programming interface using symbolic names for the S7 connections. When, for example, the s7_get_cref() function is called, the parameter “conn_name" is used as the input parameter for a symbolic name of an S7 connection (refer to the “S7 programming interface" manual). The ***global name*** of an entry in the TS directory corresponds to the symbolic name of an S7 connection. The global name of an S7 connection in the TS directory contains both the local address information and the address information for the remote communications partner.

TNS Daemon tnsxd

With all inquiries to the transport name service TNSX, the TNSX daemon ***tnsxd*** is the central server process. A SOFTNET-S7/UNIX application accesses the TS directory using the TNSX daemon to obtain the address information belonging to a global name.

This procedure is based on the following principle: a SOFTNET-S7/UNIX application sends an inquiry about a global name to the TNSX daemon. The TNSX daemon then accesses the TS directory, reads the required address information and sends this back to the inquiring SOFTNET-S7/UNIX application.

The TNS daemon *tnsxd* is started when the operating system is started. It can also be started at any time with the following command:

/opt/etc/tnsxd &

TNS Compiler tnsxcom

Entries in the TS directory are created using the TNSX compiler ***tnsxcom*** with which they can also be read and modified. The TNS entries made in the TS directory are transferred in the form of a file when the TNSX compiler is called. The TNSX compiler expects the TNS entries in this file in a fixed format. Once the format has been checked, the TNSX compiler compiles the entries into the format of the TS directory and writes the entries to the TS directory.

The TNSX compiler ***tnsxcom*** is called as follows:

/opt/bin/tnsxcom [-D] [-u] [filename]

-D	Dump mode: tnsxcom outputs all the TNS entries in the TS directory to the “*filename*" file. This option excludes the “-u" option.
-u	Update mode: tnsxcom reads the TNS entries from the “*filename*" file, runs a syntax check and writes syntactically correct entries to the TS directory. Any existing entries are updated This option excludes the “-D" option. Calling tnsxcom with the “-u" option is only possible under the root ID.
filename	Name of the file to which the TNS entries will be written (Option “-D") or from which the TNS entries will be read (Option “-u").

TNS Entries

The TNS entries transferred to the TNS compiler tnsxcom in the input file must have the syntax described below for SOFTNET-S7/UNIX.

The protocol architectures (transport systems) supported by SOFTNET-S7/UNIX are used depending on the address format in the TNS entries. The following transport systems are supported by SOFTNET-S7/UNIX:

- OSI transport protocol with inactive network layer (null Internet).
- RFC 1006 via TCP/IP

3.2 TNS Configuration for the OSI Transport Protocol

TNS Entry for the OSI Transport Protocol

If a TNS entry has the following format, the OSI transport protocol with inactive network layer (null Internet) is selected as the transport system for a SOFTNET-S7/UNIX connection.

```
<S7 connection name>\
        TSEL LANSBKA <local_tsel>
        TA LANSBKA <subnet_id> <ethernet_adr> <remote_tsel>
```

<S7 connection name>

Symbolic S7 connection name. The entry is made in the TS directory with this global name.

Character string with maximum 30 ASCII characters.

The name must be terminated with the “\" character.

There must be an entry in the TS directory for each symbolic S7 connection name used in your SOFTNET-S7/UNIX application.

TSEL

This keyword is fixed. The entries following the keyword in this line contain the local address information of the symbolic S7 connection name.

LANSBKA

This keyword is fixed. It indicates the use of the OSI transport protocol with inactive network layer (null Internet) as the transport system.

<local_tsel>

Indicates the local transport selector (local TSEL). The local SOFTNET-S7/UNIX application is obtainable on the network using this transport selector. If active connection establishment to a remote SOFTNET-S7/UNIX application is selected, this transport selector is used as the calling TSEL.

The following formats are possible for <local_tsel>:

A'<string>'	<string> character string with a maximum of 8 ASCII characters
X'<string>'	<string> character string with a maximum of 8 hexadecimal digits

TA

This keyword is fixed. The entries following it in this line contain the address information of the remote SOFTNET-S7/UNIX application identified by this symbolic S7 connection name.

LANSBKA

This keyword is fixed. It indicates the use of the OSI transport protocol with inactive network layer (null Internet) as the transport system.

<subnet_id>

When using the OSI transport protocol, SOFTNET-S7/UNIX supports operation with more than one Ethernet board. This parameter is used to address the Ethernet board used in active connection establishment to the remote SOFTNET-S7/UNIX application.

Caution: *The <subnet_id> parameter depends on the particular operating system. Please refer to the SOFTNET-S7/UNIX product information for the operating system you are using to check the correct value.*

The following format is mandatory for <subnet_id>:

X'<string>'	<string> character string with exactly 4 hexadecimal digits

<ethernet_addr>

This is the Ethernet address via which the remote SOFTNET-S7/UNIX application can be obtained on the network.

The following format is mandatory for <ethernet_addr>:

Character string with exactly 12 hexadecimal characters without separators.

<remote_tsel>

This is the remote transport selector (also known as the “called TSEL"). The remote SOFTNET-S7/UNIX application is addressed on the network using this transport selector.

The following formats are possible for <remote_tsel>:

A’<string>’ <string> character string with a maximum of 8 ASCII characters

X’<string>’ <string> character string with a maximum of 8 hexadecimal octets

Example of the OSI Transport Protocol

```
S7_Conn_1\
        TSEL   LANSBKA X’010F’
        TA     LANSBKA X’<subnet_id>’ 080014151588  A’Softnet1’
```

S7_Conn_1

The example above shows a TNS entry stored under the global name “S7_Conn_1" in the TS directory. The global name corresponds to the symbolic S7 connection name used in the S7 programming manual. The name is terminated with the “\" character.

LANSBKA

The “LANSBKA" parameter indicates the use of the OSI transport layer.

TSEL LANSBKA X’010F’

A TNS entry contains the entire addressing information required for a SOFTNET-S7/UNIX connection. When using the OSI transport layer this includes the local transport selector TSEL, with which the SOFTNET-S7/UNIX application logs on at the local transport system and the Ethernet address of the partner system and the transport selector of the remote SOFTNET-S7/UNIX application.

In the example above, the transport selector X'010F' is used to logon at the local transport system. The parameter “X" stands for hexadecimal; in other words the transport selector “010F" must be specified in hexadecimal format. The local SOFTNET-S7/UNIX application is obtainable on the network using this transport selector. The “LANSBKA" parameter indicates the use of the OSI transport layer with inactive network layer (null Internet).

TA LANSBKA X’<subnet_id>’ 080014151588 A’Softnet1’

If a connection is established to the remote SOFTNET-S7/UNIX application, the Ethernet address 080014151588 and the transport selector A'Softnet1' are used. The parameter “A" stands for ASCII, in other words the “Softnet1" transport selector is interpreted as an ASCII string. The

“LANSBKA" parameter indicates the use of the OSI transport layer with inactive network layer (null Internet).

X’<subnet_id>’

If the OSI transport layer is used, SOFTNET-S7/UNIX supports operation with more than one Ethernet board. The parameter X'<subnet_id>' specifies the Ethernet board that is used for active establishment of the connection to the remote SOFTNET-S7/UNIX application. In other words, the particular Ethernet board is addressed using the <subnet-id> parameter. Since this parameter depends on the operating system being used, it is not possible to indicate a generally valid value for all operating systems supported by SOFTNET-S7/UNIX in the above example. Please refer to the SOFTNET-S7/UNIX product information for the operating system you are using to check the correct value.

Client and Server Locally via the OSI Transport Protocol

Due to the fact that server services are not yet implemented in SOFTNET-S7/UNIX, the operation of a client application with a server application purely locally on one computer is not possible.

3.3 TNS Configuration for RFC1006 via TCP/IP

TNS Entry for RFC1006 via TCP/IP

The format described below for a TNS entry selects RFC1006 via TCP/IP as the transport system for a SOFTNET-S7/UNIX connection.

```
<S7 connection name>\
        TSEL RFC1006 <local tsel>
        TA RFC1006 <ip-addr> <remote tsel>
```

<S7 connection name>

Symbolic S7 connection name. The entry is made in the TS directory with this global name.

Character string with maximum 30 ASCII characters.

The name must be terminated with the “\" character.

There must be an entry in the TS directory for each symbolic S7 connection name used in your SOFTNET-S7/UNIX application.

TSEL

This keyword is fixed. The entries following the keyword in this line contain the local address information of the symbolic S7 connection name.

RFC1006

This keyword is fixed. It indicates use of RFC1006 via TCP/IP.

<local_tsel>

Indicates the local transport selector (local TSEL). The local SOFTNET-S7/UNIX application can be obtained on the network using this transport selector. With active connection establishment to a remote SOFTNET-S7/UNIX application, this transport selector is used as the calling TSEL.

The following formats are possible for <local_tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal digits

TA

This keyword is fixed. The entries following it in this line contain the address information of the remote SOFTNET-S7/UNIX application identified by this symbolic S7 connection name.

RFC1006

This keyword is fixed. It indicates use of RFC1006 via TCP/IP.

<ip-addr>

This is the Internet address via which the remote SOFTNET-S7/UNIX application can be obtained on the network.

The following format is mandatory for <ip-addr>:

<number>.<number>.<number>.<number> where 0 <= number < 256

<remote tsel>

This is the remote transport selector (also known as the “called TSEL"). The remote SOFTNET-S7/UNIX application is addressed on the network using this transport selector.

The following formats are possible for <remote tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal octets

Example of RFC1006 via TCP/IP

```
S7_Conn_1\
        TSEL RFC1006 A'1212'
        TA RFC1006 192.33.21.23 A'4277'
```

S7_Conn_1

The example above shows a TNS entry stored under the global name “S7_Conn_1" in the TS directory. The global name corresponds to the symbolic S7 connection name used in the S7 programming manual. The name is terminated with the “\" character.

RFC1006

The “RFC1006" parameter indicates the use of RFC1006 via TCP/IP.

TSEL RFC1006 A'1212'

A TNS entry contains the entire addressing information required for a SOFTNET-S7/UNIX connection. When using RFC1006 via TCP/IP, these include the local transport selector with which the SOFTNET-S7/UNIX application logs on at the local transport system, the Internet address of the partner system and the transport selector of the remote SOFTNET-S7/UNIX application.

In the example above, the transport selector A'1212' is used to log on at the local transport system. The parameter “A" stands for ASCII, in other words the port number “1212" must be specified in decimal format. The local SOFTNET-S7/UNIX application is obtainable on the network using this transport selector. The “RFC1006" parameter indicates the use of RFC1006 via TCP/IP.

TA RFC1006 192.33.21.23 A'4277'

When the connection is established to the remote SOFTNET-S7/UNIX application, the Internet address 192.33.21.23 and the transport selector A'4277' are used. The parameter “A" stands for ASCII, in other words the transport selector “4277" must be specified in ASCII format and is interpreted as a decimal value. The “RFC1006" parameter indicates the use of RFC1006 via TCP/IP.

☞ **The TCP port 102 is reserved for applications that allow an RFC1006 implementation. This port is used by RFC1006 applications to receive connection establishment requests from other communication partners (which must be possible with the SAPI-S7 protocol).**
This port is known as a privileged port on UNIX systems; in other words it must only be used for applications operated under the root ID.
This means that SAPI-S7 applications operating with RFC1006 via TCP/IP must be operated under the root ID.

Client and Server Locally with RFC1006 via TCP/IP

Due to the fact that server services are not yet implemented in SOFTNET-S7/UNIX, the operation of a client application with a server application purely locally on one computer is not possible.

3.4 Restrictions for Names

Restrictions for Names in TNS Entries

The **logical name of a TNS entry** can be a maximum of 30 characters long. It must start with a letter or the underscore character, the remaining 29 characters can be letters, underscores or numbers.

The **transport selectors** can be either in ASCII format when preceded by an "A" or in hexadecimal format when preceded by an "X".

When using the **ASCII format**, a maximum of 8 characters are possible for the **transport selector**. Permitted characters are letters, numbers or the underscore character.

When using the **hexadecimal format** for the **transport selector**, an even number of a maximum 16 hexadecimal digits can be used. Hexadecimal numbers are made up of the numbers 0 to 9 and the letter a to f or A to F.

☞ **Please note that there may be restrictions on the partner systems that go beyond the restrictions explained here.**

3.5 S7 Connections with the SAPI-S7 Protocol

Remote TSAP

TSAP of the S7 partner station

☞ **The remote TSAP cannot be freely selected but is fixed mainly by the S7 hardware configuration.**

It consists of two groups each with two hexadecimal characters.

First group:	Device ID, used internally by the SIMATIC S7 PLC.

Possible IDs:

01 = PG , PC (to be used in Softnet)
02 = OS
03 = Other

Second group: Addressing of the SIMATIC CPU, as follows:
(bit 7..5) = rack
(bit 4..0) = slot

To allow the resources of the S7 CPU to be processed (data blocks, inputs/outputs etc.) with the SAPI-S7 protocol, the position of the CPU in the PLC is specified here.

For an S7-400 with only one rack and the CPU in slot 3, this results in a TSAP of **0103** :

The PC communicates directly with the SIMATIC CPU in rack 0, slot 3.

For all SAPI-S7 connections, the same remote TSAP must always be used, the local TSAPs must be different!

The corresponding TNS entries could be as follows:

```
conn_1\
        TA      LANSBKA  X'<subnet_id>' <Ethernet_addr.>  X'0103'
        TSEL    LANSBKA  A'loc1'
conn_2\
        TA      LANSBKA  X'<subnet_id>' <Ethernet_addr.>  X'0103'
        TSEL    LANSBKA  A'loc2'
```

Field-Oriented Services

In contrast to the variable services, when using the field-oriented services, the connection must be configured at both ends (i.e. also on the S7 PLC).

☞ **STEP 7 proposes TSAPs for the connections, and only some of these can be modified by the user. The configuration on the UNIX system must match up with the S7 end; in other words, the local TSAP on the S7 = remote TSAP on UNIX and vice versa.**

3.6 Working with the TNSX Compiler

Generating TNS Entries

The TNS entries are generated using the TNSX compiler *tnsxcom*. The address information is edited in an ASCII file whose name is transferred to the TNS compiler as a parameter.

➘ Calling the TNS compiler to generate TNS entries:

```
/opt/bin/tnsxcom -u <ASCIIfile>
```

This call must be made as superuser.

The TNS entries in the specified file are included in the TS directory.

- Checking the newly generated entries:

```
/opt/bin/tnsxcom -D ASCIIfile
```

This call does not need to be made as superuser.

All the TNS entries contained in the TS directory are written to the specified file.

Modifying Existing TNS Entries

If you need to change the address information of existing TNS entries, follow the procedure below:

- Write the current TNS entries to an ASCII file:

```
/opt/bin/tnsxcom -D ASCIIfile
```

This call does not need to be made as superuser.

All the TNS entries contained in the TS directory are written to the specified file.

- Change the address information in the ASCII file:

Refer to the previous sections explaining TNS configuration.

- Update the TNS entries by calling the TNS compiler:

```
/opt/bin/tnsxcom -u ASCIIfile
```

This call must be made as a super user.

The TNS entries in the specified file are included in the TS directory or entries already in the TS directory are updated.

- Check the modifications made:

```
/opt/bin/tnsxcom -D ASCIIfile
```

This call does not need to be made as superuser.

The TNS entries currently in the TS directory are written to the specified file.

Deleting Existing TNS Entries

You delete an existing TNS entry as follows:

- Generate an ASCII file with the following entry:

```
TNSname DEL
```

Where TNSname = the TNS entry to be deleted

➘ Call the TNS compiler:

```
/opt/bin/tnsxcom -u ASCIIfile
```

This call must be made as superuser.

All the TNS entries in the specified file are removed from the TS directory.

➘ Check the deletion:

```
/opt/bin/tnsxcom -D ASCIIfile
```

This call does not need to be made as superuser.

The entries remaining in the TS directory are written to the specified file.

Outputting the Address Information of all TNS Entries

The current address information of all existing TNS entries can be output using the following command. The output can be redirected to an ASCII file as follows:

```
/opt/bin/tnsxprop > ASCIIfile
```

Where ASCIIfile = any file name

All the entries currently in the TS directory are written to the file.

The format of the TNS entries output by tnsxprop is easier to read than the format output by calling tnsxcom.

3.7 Configuring the VFDs and the Assigned Connection Name

General

Using the s7_init() call, a SOFTNET-S7/UNIX application logs on at a VFD of an underlying communication system. As input parameters, the CP name and the VFD name are specified. With this logon, the SOFTNET-S7/UNIX application has a list of connections available that can be used for communication with remote systems.

The following sections describe how the VFD names assigned to a communications processor are stored in configuration files. These configuration files also contain the connection names assigned to a VFD name.

Name Conventions

A configuration file *<CP_name>.dat* must exist for each communications processor used by a SOFTNET-S7/UNIX application. This contains all the VFDs used and the connection names that can be used by a SOFTNET-S7/UNIX application.

The names of the CPs are as follows:

h1_0	CP name for the first board
h1_1	CP name for the second board
...	...
h1_x	CP name for the nth board, where x=n-1

Within the SAPI-S7 library, the TP4 configuration file */opt/lib/tp4/netconf* is read to determine the CP names. The CP names are assigned according to the number of communications processors configured for SOFTNET-S7/UNIX. The order in which the communications boards occur in this file determines the order of the CP names.

The name of the configuration file for the individual boards consists of the CP name and the extension "*.dat*", as follows:

<CP_Name>.dat

As a result, the configuration files for the individual CPs have the following names:

h1_0.dat	configuration file for the first board
h1_1.dat	configuration file for the second board
...	...
h1_x.dat	configuration file for the nth board, where x=n-1

Location of the Configuration Files <CP_name>.dat

Initially, these configuration files must be located in the current directory from which the SOFTNET-S7/UNIX application is started. Using UNIX environment variables, however, each individual configuration file can be moved to any directory. To do this, the following UNIX environment variables are used:

h1_0_s7h1	environment variable for the configuration file for the first board
h1_1_s7h1	environment variable for the configuration file for the second board
...	...

h1_x_s7h1	environment variable for the configuration file for the nth board, where x=n-1

Example

If the environment variable h1=0_s7h1 is set as shown in the following example, the configuration file h1_0.dat will be expected in the /usr/tmp directory:

```
csh:    setenv h1_0_s7h1 /usr/tmp/h1.cfg
sh:     h1_0_s7h1=/usr/tmp/h1.cfg; export h1_0_s7h1
```

Structure of the Configuration Files <CP_name>.dat

The entries in the configuration files for a specific CP are as follows:

```
appl_assoc_name    =     <s7_conn_name>
vfd_name           =     <s7_vfd_name>
<empty line>
```

The contents must have the following syntax:

appl_assoc_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following S7 connection name.
<s7_conn_name>	Symbolic S7 connection name. Maximum 30 characters. A TNS entry with the same name must also exist for this symbolic S7 connection name (global TNS name = symbolic S7 connection name).
vfd_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following VFD name.
<s7_vfd_name>	Name of a VFD. Maximum 32 characters. The symbolic S7 connection name in the line before is assigned to this VFD.
<empty line>	An empty line used to separate this entry from the next.

The entries above can be repeated as often as required. However there must be at least one entry under appl_assoc_name and vfd_name.

Example of the Configuration File "h1_0.dat"

An example of the configuration file "*h1_0.dat*" is shown below:

```
      appl_assoc_name = V1
      vfd_name = VFD1

      appl_assoc_name = V2
      vfd_name = VFD1

      appl_assoc_name = V3
      vfd_name = VFD1

      appl_assoc_name = V4
      vfd_name = VFD2

      appl_assoc_name = V5
      vfd_name = VFD2
```

In this example, two VFDs are defined for CP "h1_0": VFD1 and VFD2. In this example, two VFDs are defined for CP "h1_0":

3.8 Distributing the Address Information on Two TNS Entries

In Sections 3.2 "TNS Configuration for the OSI Transport Protocol" or 3.3 "TNS Configuration for RFC1006 via TCP/IP", the address information for an S7 connection at the local end (see entry TSEL) and for the remote end (see entry TA) is written in a single TNS entry.

The address information for an S7 connection can also be written in two TNS entries. One TNS entry contains the address information for the local end, the other contains information for the remote end.

The TNS entries are then assigned to an S7 connection in the configuration files for the specific CP <CP_name>.dat.

Structure of the Configuration Files <CP_name>.dat

The entries in the configuration files for the specific CPs are structured as follows:

```
      appl_assoc_name      =      <s7_conn_name>
      vfd_name             =      <s7_vfd_name>
      local_name           =      <local_tns_name>
      remote_name          =      <remote_tns_name>
      <empty line>
```

The parameters must have the following syntax:

appl_assoc_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following S7 connection name.
<s7_conn_name>	Symbolic S7 connection name. Maximum 30 characters. If the parameters “local_name" and “remote_name" below are specified, the S7 connection name entered in “appl_assoc_name" is not used for the TNS access but rather the names specified in “local_name" and “remote_name".
vfd_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following VFD name.
<s7_vfd_name>	Name of a VFD. Maximum 32 characters. The symbolic S7 connection name in the line before is assigned to this VFD.
local_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following TNS name for the local connection endpoint.
<local_tns_name>	Name of the TNS entry for the local connection endpoint. The assigned TNS entry contains the address information for the local end of the S7 connection (parameter TSEL).
remote_name =	Keyword followed by the “=" character that is interpreted as a delimiter from the following TNS name for the remote connection endpoint.
<remote_tns_name>	Name of the TNS entry for the remote connection endpoint. The assigned TNS entry contains the address information for the remote end of the S7 connection (parameter TA).
<empty line>	An empty line used to separate this entry from the next.

The entries above can be repeated as often as required. However there must be at least one entry under appl_assoc_name and vfd_name.

Example of the Configuration File "h1_0.dat"

An example of the configuration file "*h1_0.dat*" is shown below:

```
      appl_assoc_name = V1
      vfd_name = VFD1
      local_name = local_1
      remote_name = remote_0

      appl_assoc_name = V2
      vfd_name = VFD1
      local_name = local_2
      remote_name = remote_0

      appl_assoc_name = V3
      vfd_name = VFD1
      local_name = local_3
      remote_name = remote_0
```

In the example of the configuration file "h1_0.dat" above, three S7 connections "V1", "V2", and "V3" are entered for the VFD "VFD1". The address information for the local end of these connections is in the TNS entries "local_1", "local_2", and "local_3". All three S7 connections use the TNS entry "remote_0" for the remote end.

Corresponding TNS Configuration

These are the TNS entries for the example above:

```
local_1\
      TSEL   LANSBKA X'0001'

local_2\
      TSEL   LANSBKA X'0002'

local_3\
      TSEL   LANSBKA X'0003'

remote_0\
      TA     LANSBKA  X'<subnet_id>' 080014151588  X'0103'
```

The TNS entries "local_1", "local_2" and "local_3" contain address information for the local end (parameter TSEL).

The TNS entry "remote_0" that is used by all three S7 connections in the example above, contains the address information for the remote end (parameter TA).

The parameters for the entry TSEL for the address information of the local connection endpoint and the entry TA for the address information of the remote connection endpoint are described in Section 3.2 "TNS

Configuration for the OSI Transport Protocol" and in Section 3.3 “TNS Configuration for RFC1006 via TCP/IP".

3.9 Notes on Configuration

Mapping the VFD on the CP

The VFDs that are valid on a CP are defined in the configuration files for the specific CP.

The mapping of the VFD names on the CP is unique, a CP can be assigned several VFDs,

however the VFD can only be assigned to one CP.

Mapping the VFD S7 Connection on a VFD

In addition to VFDs, the configuration files for a specific CP also contain the valid S7 connections.

The mapping of the connection name on the VFD name is unique, this means that a VFD name can be assigned one or more connection names,

however a connection name can only be assigned to one VFD name.

Working with More Than One Ethernet Board

The Ethernet board used to establish a connection to the remote SOFTNET-S7 application is selected in the TNS entries using the <subnet_id> parameter. This means that it is not necessarily the CP in whose configuration file the symbolic S7 connection name is entered that is selected for connection establishment. The Ethernet board is selected only using the <subnet_id> parameter in the corresponding TNS entry.

Recommended Configuration

The concept of assigning VFD and S7 connection names in the configuration files for the specific CPs is part of the PC-based version of the SAPI-S7 programming interface under Microsoft Windows.

his concept was adopted for SOFTNET-S7/UNIX to ensure compatibility with the PC version.

If there are no compelling reasons to the contrary, such as compatibility with the PC version, a single configuration file “h1_0.dat" should be used with SOFTNET-S7/UNIX applications even if you are working with more than one Ethernet board. To simplify matters, this one configuration file should contain all the VFDs used and the corresponding connection names. This avoids unnecessary confusion during configuration.

To recapitulate: With SOFTNET-S7/UNIX, the Ethernet board used is selected not using the name of the configuration file for the specific CP but always using the <subnet_id> in the TNS entry for an S7 connection.

4 Error Diagnostics

The following sections describe the tools and logging mechanisms for diagnosing error groups. The errors that can occur when using SOFTNET-S7/UNIX applications can be divided into two groups, as follows:

Communication Errors

The main causes and effects of communication problems are as follows:

- Application associations cannot be established. The most common causes are problems with the underlying transport system (for example transport name service not started).
- Individual application associations cannot be established. The cause is usually in the configuration or the limit values of the transport system (for example maximum number of connections) have been exceeded.
- Application associations break down during operation. These problems can be caused by longer network failures, incorrect response of the partner or by the application not calling *s7_receive()* often enough.
- Problems during the data exchange between applications. The cause is usually an incorrect response on the partner or that *s7_receive()* is not called often enough.

Programming Errors

Common programming errors are as follows:

- Incorrect parameters are transferred to the S7 functions.
- Rare external events (errors, exceptional situations) have not been taken into account.
- The *s7_receive()* S7 function is not called often enough.

4.1 Error Diagnostics for Communication

CCP-TP4 Transport System

The STREAM that forms the protocol stack is maintained by the *tp4d* background process.

Using the UNIX command *ps*, you can check whether the tp4d process is active.

The *tp4stat* command is also available with which you can fetch and analyze statistical information from the transport system. Use the following command:

```
/opt/bin/tp4stat
```

Transport Name Service (TNS)

In the TNS, the background process *tnsxd* manages the address information of the applications involved in communication as TNS entries in the TNS directory.

With the UNIX command *ps* you can check whether the *tnsxd* process is active.

The existing TNS entries of the TNS directory can be read into the ASCII file with the name ASCIIfile using the command

```
tnsxcom -D ASCIIfile
```

and checked with the command

```
tnsxcom -s ASCIIfile
```

CMX Library Trace

The trace of the CMX library is controlled by the environment variable *CMXTRACE*. By supplying the environment variable with a value, the trace is activated and the scope of the information to be collected is specified.

The trace entries of a process are collected as compact binary data in a dynamically created buffer and periodically written to temporary files. These binary files are edited separately with the *cmxl* tool. The binary files are saved in the directory */usr/tmp*. The file names consist of the prefix *CMXLa* or *CMXMa* and the process identification number *pid*.

cmxl reads the entries generated by the trace from the temporary file. The scope of the analysis is decided by the options selected for *cmxl*.

CMXTRACE: Controlling the Trace

The options specified in CMXTRACE control the trace. The options *s*, *S*, and *D* determine what is logged. The options *p*, *r* control the buffering and (wrap) writing of the file:

```
CMXTRACE = "[ -s] [-S] [ -D] [ -p fac] [ -r wrap] [ -f directory]"
```

-s	The CMX calls, their arguments, the options and user data are logged normally.
-S	The calls, their arguments, the contents of any options, the user data in their full length are logged. The options -s and -S exclude each other.
-D	The calls with additional information about system calls are logged in detail. This option is only available in addition to -s or -S.
-p *fac*	The decimal digit *fac* determines the buffer factor. The amount of buffering is determined according to *fac* * BUFSIZ where BUFSIZ is determined by <stdio.h>. If you specify *fac* = 0, each trace entry is written to the file immediately (with no buffering). *fac* = 0..8 Default: *fac* = 1
-r *wrap*	The decimal number *wrap* specifies that after *wrap* * BUFSIZ bytes (BUFSIZ according to <stdio.h>) logging continues in the second temporary file <directory>/CMXMa<pid>. This second file handles the trace in exactly the same way as CMXLa<pid>. After *wrap* * BUFSIZ bytes, the trace switches between the two files. Following this switch over the content of the file is overwritten. Default: *wrap* = 128
-f *directory*	The trace files are written to the specified directory. Default: *directory*: /usr/tmp or /var/tmp

cmxl:
How the Trace Trace

cmxl reads the entries generated by the trace from the temporary *file*, processes the entries according to the selected options and outputs the result to *stdout*.

The following options specify which trace entries from the *file* are processed. It is possible to specify more than one of the values described below per *cmxl* call. Only the options *v* and *x* exclude each other. If no options are specified, *cdex* is used as the default.

```
cmxl [ -c] [-d] [ -e] [ -t] [ -x] [ -D] file
```

-c	The CMX calls for logging on and off the TS application with CMX and for establishing and terminating the connection are processed.

-d	The CMX calls for data exchange and flow control are processed.
-e	The CMX calls for handling events are processed.
-t	In addition to logging the error messages, the t_error() calls are processed explicitly. Error messages are always logged even if this option is not specified.
-v	The CMX calls, their arguments, the options and the user data are processed in detail. The extent of processing depends on the options specified for CMXTRACE.
-x	The calls and their arguments are processed without options and user data.
-D	This option selects detailed processing with additional information about system calls.
file	Name of one or more files containing trace entries to be processed.

Example of Activating and Evaluating the CMX Library Trace

Example of a configuration for activating the CMX library trace:

```
csh:    setenv CMXTRACE "-SD -p 0 -r 64 -f ."
sh:     CMXTRACE="-SD -p 0 -r 64 -f ."; export CMXTRACE
```

Example of a configuration for editing the trace files:

```
cmxl    -Dv CMXLa<pid> > file_name
```

It is advisable to redirect the data to a file, otherwise they are output to *stdout*.

☞ **CMX error codes and a brief description can be found in the CMX header file */usr/include/cmx.h*.**

SOFTNET-S7/UNIX Trace

The trace of the S7 library can be controlled by a total of three environment variables, as follows:

S7_TRACE_SELECT

S7_TRACE_DEPTH

S7_TRACE_TARGET

Using these environment variables, you can set the service classes for which the entries will be made in the trace, the trace depth and the trace target (refer to the sapi_s7.h file).

Sample configuration for activating the CMX library trace:

```
sh:     S7_TRACE_TARGET=1; export S7_TRACE_TARGET
        S7_TRACE_SELECT=65535; export S7_TRACE_SELECT
        S7_TRACE_DEPTH=104; export S7_TRACE_DEPTH

csh:    setenv S7_TRACE_TARGET 1
        setenv S7_TRACE_SELECT 65535
        setenv S7_TRACE_DEPTH 104
```

The existence and values of the environment variables are checked when the trace is initialized. Trace settings made previously are then overwritten.

It is advisable to set the trace using the environment variables listed above and not using mini DB calls. This means that the default values can be overwritten for debugging without modifying the application and having to recompile it.

Testing the Transport Connection (tping)

Communication at the transport layer can be checked with the *tping* program. The program uses the transport name service.

```
tping [-o tnsname1] [-p tnsname2]
```

Parameter:	***-o tnsname***	global name for the TNS entry for the logon at the local transport system; if the parameter is not specified, the default *tping* applies.
	-p tnsname2	TNS entry with which it is attempted to establish a transport connection; if the parameter is not specified, *tnsname1* is used.

The program logs on at the local transport system with the TNS entry *tnsname1* and attempts to establish a transport connection with the TNS entry *tnsname2*. Three reactions to the connection request are possible:

- **Connection request accepted**
  Message from tfping:

```
"T_CONF received after <time> seconds.
 Connection to remote system established!!!"
```

The connection request was confirmed with Connect Confirm by the partner. This proves that the transport system is functioning and that the parameters of the transport layer are correctly set. *tping* terminates the connection again with DISCONNECT and logs off at the transport system.

➢ **Connection request rejected**
Message from tfping:

```
„T_DISIN received after <time> seconds... returned code:
 <error number>: <error description>"
```

The connection request was rejected with Disconnect by the partner. Communication via the transport system is functioning. If a transport connection to this partner is required, parameters must be adapted to the transport layer.

➢ **No reaction**
Message from tfping:

```
„T_DISIN received after <timeout> seconds... returned code:
 <error no.>: Connection cannot be set up because partner does not
             respond to CONRQ"
```

Possible causes include:

- There is not partner in the network with this network address or the partner is not operational.

- The partner is configured so that it does not react to incorrect transport layer parameters (for example TSAP).

The functionality of the transport system can be checked using a LAN analyzer. The transport system is functioning when the LAN analyzer records the appropriate connect request PDU.

☞ **The CMX error codes (*reason*) and a short description can be found in the CMX header file */usr/include/cmx.h*.**

☞ **Errors can be investigated in greater detail with the CMX trace tool.**

5 Sample Program

A sample program is available in the directory */usr/s7/example*. The sample program contains a client application, configuration files and a makefile. These sample files will help you to develop a SOFTNET-S7/UNIX application. To allow communication with a server application, a server application must be configured and the address data in the configuration files of the client application must be adapted to match the data of the server application.

The example programs consist of the following files:

xs7_bsp.c	SOFTNET-S7 client application
makefile	Makefile for client application

With the supplied *makefile*, you can compile the programs and link the libraries.

To initialize the communication system, the directory also contains the two configuration files

h1_0.dat	CP-specific configuration file
tns_inp.data	TNS entries

The configuration files match the supplied client application. The TNS entry must still be adapted to your server application.

The variables may still need to be adapted to the situation in the partner station.

6 Who to Contact

Contacts for Technical Questions

If you have questions about using this product, please call our SIMATIC NET hotline in Nuremberg:

Siemens AG
Customer Support

Telephone: ++49-911-895-7000
Telefax: ++49-911-895-7001

7 References

/1/ SINEC - S7 Programming Interface
Manual, Release 02, 1996
Order number: C79000-B8976-C077/02