COSMOS
Programmer Guide

ND-60.164.3 EN

COSMOS
Programmer Guide

ND-60.164.3 EN

PRINTING RECORD

Printing	Notes
11/84	Version 1 EN
10/85	Version 2 EN
05/86	Version 3 EN

COSMOS Programmer Guide
Publ.No. ND-60.164.3 EN

Norsk Data

Norsk Data A.S
Graphic Center
P.O.Box 25, Bogerud
0621 Oslo 6, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the Customer Support Information (CSI) and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:

THE PRODUCT

This manual documents the COSMOS programmer library which includes the following libraries:

- Use of XMSG from PLANC (XMP)
- Use of XMSG from FORTRAN (XMF)
- Use of RR-LIB from PLANC (RRP)
- Use of TLIB from PLANC (TLP)
- Use of TLIB from FORTRAN (TLF)

These versions are all based on XMSG version J.

The product number is ND-10609B.

THE READER

This manual is written for the programmer who needs to write data communication software based on XMSG or RR-LIB.

PREREQUISITE KNOWLEDGE

The reader should have a general understanding of data communication, knowledge of the SINTRAN III operating system, and programming experience in PLANC or FORTRAN. Some of the advanced features will require a general understanding of NPL (Nord Programming Language).

RELATED MANUALS

SINTRAN III Communication Guide	ND-60.134
COSMOS Operator Guide	ND-30.025

CHANGES FROM THE PREVIOUS VERSION

The manual is completely reorganized and most parts are expanded or rewritten. More examples are included and the manual now covers all programming libraries of datacommunications available through COSMOS:

TABLE OF CONTENTS

Section Page

Section Page

APPENDIX

Section Page

Section Page

Section Page

CHAPTER 1

XMSG - TASK TO TASK MESSAGE SYSTEM

1 XMSG - TASK TO TASK MESSAGE SYSTEM

1.1 Synopsis

This chapter explains the general concepts of XMSG and should be read by all users who are using XMSG.

1.2 General Concepts

XMSG is a system made for communication between different tasks. The communicating tasks may reside in the same system or in different systems.

1.2.1 Task

Many applications require the division of a program system into separate, asynchronous processes or tasks, that communicate by sending messages. This separation may be motivated by:

- Security considerations like separation of work areas or definition of interface points.
- Hardware design (tasks may run in separate systems).
- Address space limitations.
- Simplicity of program development.

We will use the word task to mean a driver, a direct task, or an RT (foreground or background) program, all being processes running on ND-500/PIOC. The XMSG system allows tasks to send messages to each other, including the handling of memory allocation, queueing, and task synchronization. Each user of XMSG has an XT-block (task block). The XT-block is automatically allocated by XMSG.

RT-programs that call XMSG operate as two independent tasks as seen by XMSG: a user task and a system task. Separation between these two tasks is done by using the XFSYS (system mode) option, when one wants to act as system task. This is only allowed when running on ring 2. It is used by SINTRAN when using XMSG on behalf of the user (e.g. in COSMOS remote file access).

1.2.2 System

A system is a processing unit that runs an independent XMSG kernel. An ND-100 CPU is a system, but a PIOC or an ND-500 is not. These are seen as part of an ND-100, since every PIOC or ND-500 process which uses XMSG has a 'shadow' task in the ND-100.

1.2.3 Port

A task can open ports through which it can send and receive messages. All ports belong to XMSG in the sense that they are tables residing in XMSG space in the physical memory. A task becomes the owner of a port when it executes an open port call. When a port is opened, it remains owned by the task until the task decides to close the port or until the task terminates. A task may have several open ports at the same time. When a task opens a port, it is identified locally with a unique port number (like a file number).

A task identifies other tasks' ports using a 32-bit magic number which is comprised of the port number, the system number, and a random part. The random part makes it extremely unlikely that a port, when closed and then reopened, will have the same identifier.

There is an abbreviated version of the magic number which only requires 16 bits. This is referred to as hashed magic number, and it is almost unique.

A port may also be given a name.

A port may have one message or a queue of messages attached to it.

XMSG allocates a port list to your task. This is done so that XMSG can administrate all opened ports.

When you open a port, XMSG inserts that port on top of the port list. When you close a port, XMSG takes that port out of the list. If a port number is specified as zero, XMSG will use the first port in the port list. This port is referred to as the most recently opened port (or the default port).

1.2.4 Message Buffer

Message buffers are variable length areas which can be reserved. They belong to XMSG in the sense that they normally reside in XMSG space in the physical memory. A task can 'own' a message buffer, meaning that the message buffer is assigned to the task. The message buffer is identified by a message identifier. A task may own several message buffers, at the same time.

Before a task can transfer the contents of its own internal buffer into the message buffer, the task has to own that buffer. The user data must be put into a message buffer before you can transfer (send) the user data to another task.

When assigned to a task, the message buffer remains reserved for that task until it decides to release it or send it to another task. At that point either of the following will happen: If the tasks reside in the same system, the ownership of the buffer is transferred to the receiving task; otherwise the content of the buffer is transported to

the other system and the buffer goes back to the XMSG buffer pool. Having read the data, the receiving task may then either release the received buffer back to the pool, or use it for storing a message to be sent back to the first, or any other, task.

Note that in many of the calls to XMSG, there are no parameters that specify the message identifier, because a current (default) message buffer is assumed.

There are two types of current messages:

1) When a task refers to a message buffer, the message becomes task current. When a task receives a message buffer, it also becomes task current.

2) When a task receives a secure message on a port, the message in addition to becoming task current, the message also becomes port current on that particular port.

When there is no message parameter in a call which involves a message buffer, the default message is:

- For calls of port reference: port current if one exists, otherwise task current.

- For other types of calls: task current.

A message identifier value of -1 implies the current message. Sending or releasing a message buffer leads to its currency being lost. The task may also change the value of the current message, with the 'set current message' call.

Messages cannot be released, read from or written to by tasks other than the current owner, or while queued to a port. In the latter case the message must be received first.

Whenever a message buffer is read from or written to, we are dealing with user data. The programmer will not see the XMSG headers. Usually, the communicating tasks will follow some protocol on top of XMSG, which implies headers within the user data. Whenever an XMSG call refers to a 'header', this refers to the user data and should not be confused with XMSG's own internal headers.

A message may be sent secure. Such a message will be returned to the sending port if:

- It cannot be delivered.

- The receiving port gets closed while the message is current for that port or queued on that port.

Messages are by default nonsecure. Nonsecure messages are discarded by XMSG if they cannot be delivered, or are queued on a part that gets closed, and their message buffers are relased.

If you perform several read/write operations on the same message buffer, you may have trouble with odd displacements. This is because XMSG rounds the message displacement to an even number of bytes, so a 'garbage byte' may appear in a message. You will have no such problems if you consistently specify an even message displacement (or read/write an even number of bytes).

1.3 XROUT

XROUT is a special program that allows tasks to find each other initially, by providing a port naming scheme and a message routing facility.

It can be considered to be equivalent to the 'directory enquiries' service provided by a public telephone company, but with the following restriction:

XROUT will never give you somebody else's magic number. It may however give him a message sent by you, together with your magic number. The destination task can then notice your message and ring you back, if he wants to, and thereby give you his magic number. In this special case your message is called a letter.

The source task will in this case have to know the port name of a port owned by the destination task. The destination task must have opened and named a port beforehand, e.g., by using the combined call 'open and name port'. An alternative to simply naming a port is to declare it as a connection port. This allows XROUT to control the number of users that a port can handle simultaneously, and even distribute users among server parts.

1.4 Privileged XMSG Functions and XROUT Services

Some calls can only be executed by privileged tasks. In order to become privileged for XMSG, a task must successfully execute the 'make calling task privileged' call. In order to do this, the caller must be either a driver, a direct task, a foreground program, or a background program logged in as user SYSTEM.

1.5 Types of Routines

The following types of routines exist in the PLANC and FORTRAN XMSG libraries:

1) Routines executing only XMSG functions.

2) Routines involving XROUT services using XMSG functions.

3) Routines which do formatting of buffers according to the XROUT conventions.

1.6 Sequences of XMSG Calls

For a task receiving data, a typical sequence is as follows:

1) A named port is opened ('open and name a port').

2) The task waits for a message to arrive ('receive message' with the 'wait' option set).

3) The user data in the message buffer is transferred into a buffer internal to the task ('read message').

This is a typical sequence for a sending task:

1) An unnamed port is opened ('open port').

2) A message buffer is reserved ('get message').

3) The user data is transferred into the message buffer ('write into message buffer').

4) Data is sent to the receiving task's port ('send letter').

1.7 General Information about the Routines

The routine implementation in PLANC gives the status as an outvalue:

ROUTINE VOID,INTEGER (parameters.....). Example of a call:

```
xmpfrel(0,msgIdentifier) =: returnStatus
```

In FORTRAN the routines are implemented as functions. Example of a call:

```
returnStatus = xmffrel(0,msgIdentifier)
```

Normal return status is zero. If you include the appropriate :DEFS file in your source code, you may use the symbol XMOK for the zero status. The file is called XMP:DEFS for PLANC and XMF:DEFS for FORTRAN. If a call is not terminated, the return status is XMXENTM (e.g., if no message is waiting when XMPFRCV is called without the XFWTF option,XMXENTM). Other return statuses are error codes. A list of the error codes plus their corresponding symbols is provided in appendix D. However, note that the routines used for buffer formatting will, if an error is encountered, return -1 as the error code.

In the description of the PLANC calls, an R is used to denote a read parameter and W stands for write.

In the description of the FORTRAN calls, an I is used to denote an input parameter and O stands for output.

1.7.1 Options

Certain options may be permitted for each call. Which ones you may use are specified, under each call, later in this manual. How to use them is specified in the same places.

The options are implemented as flags. This means that certain bits have to be set in the 'flags' parameter. The bit position may be referred to by a symbolic name, e.g. XFWTF or XFWAK. These symbols are included in the XMP:DEFS file for PLANC and in the XMF:DEFS file for FORTRAN. Be sure to do a $INCLUDE on the relevant file if you wish to use one of these symbols.

The flags parameter is included in all calls even if there are currently no options implemented for a particular call. This is to allow for possible future options. If no options are specified, this parameter should be set to zero to ensure compatibility with future versions.

Note that, as a general rule, options not described under a specific call should not be set when the routine is called. However, an RT-program that wants to call XMSG as a system task must specify the XFSYS option (see description on page 3).

For the use of one single option in a call, please see the XMPFGST or the XMFFGST routine. For using several options simultaneously in one call we refer you to XMPFSND or XMFFSND.

1.7.2 Message Types

Some of the calls return a message type to the calling task. These types are explained under the respective calls. The message types are referred to by symbols, e.g. XMTNO. The symbols are included in the XMP:DEFS file for PLANC and in the XMF:DEFS file for FORTRAN. Be sure to do a $INCLUDE on the relevant file if you wish to use one of these symbols.

1.7.3 Parameter Types in the PLANC Interface

When you call XMSG from PLANC, you will notice in the reference section that some special data types are used. An example is Xmmsgidentifier in the XMPFRCV routine. These special data types are defined in the XMP:IMPT file. Make sure you do a $INCLUDE on this file.

1.7.4 The XMSG Command Program - A Debugging Tool

The XMSG-COMMAND program may be useful as a debugging tool, for example, the commands LIST-MESSAGES, LIST-NAMES and in the tracing of XMSG calls. For a description of this program please consult the COSMOS Operator Guide.

1.8 Summary of the Different Routines

The different PLANC routines are listed in the following table. Notice that each routine name starts with XMP. The corresponding FORTRAN routine names start with XMF. The FORTRAN routines are identical to the PLANC routines.

In the next chapters, the routines are described alphabetically.

Routine name	Purpose	Category
XMPBADB	Append double integer	Buffer
XMPBAIN	Append integer	Buffer
XMPBAST	Append string	Buffer
XMPBINI	Build header in buffer	Buffer
XMPBLET	Format letter header	Buffer
XMPBLOC	Locate parameter in buffer	Buffer
XMPBRDY	Buffer ready	Buffer
XMPCLNM	Clear port name and close port	XROUT
XMPCONF	Return the current configuration	XMSG
XMPFABR	Absolute read from physical memory	XMSG
XMPFALM	Allocate message buffers	XMSG
XMPFCLS	Close port	XMSG
XMPFCPV	Check system and user privileges	XMSG
XMPFCRD	Define a driver for XMSG	XMSG
XMPFDBK	Define a bank number for drivers	XMSG
XMPFDCT	Disconnect from XMSG	XMSG
XMPFDMM	Define maximum memory	XMSG
XMPFDUB	Define a user buffer	XMSG
XMPFDUM	Dummy call	XMSG
XMPFFRM	Free allocated message buffers	XMSG
XMPFGET	Get message buffer	XMSG
XMPFGST	General status	XMSG
XMPFLMP	List messages and ports	XMSG
XMPFM2P	Convert magno. to port and/or sys. no.	XMSG
XMPFMST	Message status	XMSG
XMPFOPN	Open port	XMSG
XMPFP2M	Convert port number to magic number	XMSG
XMPFPRV	Make calling task privileged	XMSG
XMPFPST	Port status	XMSG
XMPFRCV	Receive message	XMSG
XMPFREA	Read message	XMSG

Routine name	Purpose	Category
XMPFREL	Release message buffer	XMSG
XMPFRHD	Read header	XMSG
XMPFRRE	Receive and read message	XMSG
XMPFRRH	Receive and read header	XMSG
XMPFRTN	Return message	XMSG
XMPFSCM	Set current message	XMSG
XMPFSIN	Initialize for system functions	XMSG
XMPFSMC	Start multi call	XMSG
XMPFSND	Send message	XMSG
XMPFSTD	Start driver	XMSG
XMPFWDF	Define wake-up context	XMSG
XMPFWHD	Write header	XMSG
XMPFWRI	Write message	XMSG
XMPREAD	Read message, not necessarily current	XMSG
XMPSEND	Send message, not necessarily current	XMSG
XMPWRHD	Write header, not necessarily current	XMSG
XMPWRTE	Write message, not necessarily current	XMSG
XMPINFC	Increment free connection count	XROUT
XMPOPCN	Create connection port	XROUT
XMPOPNM	Open and name a port	XROUT
XMPROUT	Send a message to or via XROUT	XROUT

CHAPTER 2

XMSG/PLANC REFERENCE GUIDE

2 XMSG/PLANC REFERENCE GUIDE

Routine name : XMPBADB			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	outBuffer Bytes	R	Local user buffer.
2	offSet Integer	RW	Current number of bytes used in outBuffer.
3	paramValue Integer4	R	32-bit value to be coded.
4	paramNumber Integer	R	Parameter number.

FUNCTION. . : Appends a 32-bit value as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append paramValue as the next integer parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and update the offSet parameter accordingly.

paramValue will be put into the buffer as an integer4 parameter, or, if (and only if) this is valid, as an integer2 parameter.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMPBINI.
Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : The example formats a user buffer containing only one 32-bit value coded as an integer parameter.

```
% First, we initialize the user buffer
  xmpbini(myBuff,lengthBuffer,offSet) =: returnStatus
% Check returnStatus, and if Ok,
% append a 32-bit value as parameter 1
  xmpbadb(myBuff,offSet,magicNumb,1) =: returnStatus
% Check returnStatus, and if Ok, put in
% the serial number and the service number
  xmpbrdy(myBuff,XSGNM,serialnumber) =: returnStatus
```

Routine name : XMPBAIN			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	outBuffer Bytes	R	Local user buffer.
2	offSet Integer	RW	Current number of bytes used Buffer.
3	paramValue Integer2	R	16-bit value to be coded.
4	paramNumber Integer	R	Parameter number.

FUNCTION. . : Appends a 16-bit value as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append paramValue as the next integer parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and it will update the offSet parameter accordingly.

paramValue will be put into the buffer as an integer2 parameter.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMPBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats a user buffer containing only one 16-bit value coded as an integer parameter.

```
% First, we initialize the user buffer
  xmpbini(myBuff,lengthBuffer,offSet) =: returnStatus
% Check returnStatus, and if Ok,
% append a 16-bit value as parameter 1
  xmpbain(myBuff,offSet,noOfServic,1) =: returnStatus
% Check returnStatus, and if Ok, put in
% the serial number and the service number
  xmpbrdy(myBuff,XSNSP,serialnumber) =: returnStatus
```

Routine name : XMPBAST			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	outBuffer Bytes	R	Local user buffer.
2	offSet Integer	RW	Current number of bytes used in outBuffer.
3	string Bytes	R	String to be coded.
4	paramNumber Integer	R	Parameter number.

FUNCTION. . : Appends a string as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append the specified string as the next string parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and it will update the offSet parameter accordingly.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMPBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats a user buffer containing only one string parameter.

```
% First, we initialize the user buffer
  xmpbini(myBuff,lengthBuffer,offSet) =: returnStatus
% Check returnStatus, and if Ok,
% append the string as parameter 1
  xmpbast(myBuff,offSet,portName,1) =: returnStatus
% Check returnStatus, and if Ok, put in
% the serial number and the service number
  xmpbrdy(myBuff,XSGIN,serialnumber) =: returnStatus
```

Routine name : XMPBINI			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	outBuffer Bytes	R	Local user buffer.
2	lengthBuffer Integer	R	Total length of the buffer in bytes.
3	offSet Integer	W	Number of bytes used in outBuffer after initializing.

FUNCTION. . : Initializes the user buffer. The buffer is initialized according to the XROUT message format described in appendix B.

EXPLANATION : When a task sends a service request to XROUT, the request (and the response from XROUT) must be coded according to the XROUT message format.

This routine will build and initialize the XROUT header in the user buffer specified by outBuffer (and lengthBuffer) for repeated use of the other buffer formatting routines. On return from the routine, the offSet parameter contains the lenght in bytes used for the XROUT header descriptor, i.e., the space left in outBuffer for coding of parameters using XMPBADB, XMPBAIN and XMPBAST is equal to lengthBuffer-offSet. Thus so make sure that the buffer length is big enough to contain the parameter(s).

Note that outBuffer must start on an even byte boundary and that lengthBuffer must be big enough to contain the XROUT header. If one of these checks fails, -1 will be returned as error code in returnStatus.

Continued on next page.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example below formats a buffer containing two parameters.

```
% Specify the total buffer length in bytes.
  100 =: lengthBuffer
% Initialize the user buffer
  xmpbini(myBuff,lengthBuffer,offSet) =: returnStatus
% offSet no. of bytes are used for the XROUT header.
% Check returnStatus, and if Ok, append parameters.
  xmpbast(myBuff,offSet,systName,1) =: returnStatus
% Check returnStatus, and if Ok, we have used offSet
% no of bytes for the XROUT header and parameter 1.
  xmpbain(myBuff,offSet,systNumb,2) =: returnStatus
% Check returnStatus, and if Ok, we have used offSet
% no. of bytes for the XROUT header and the two
% parameters (i.e., no of bytes not yet used in
% outBuffer equals 100-offSet).
```

Routine name : XMPBLET			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	headerBuffer Bytes	R	Local user buffer.
2	lengthBuffer Integer	R	Total length of the buffer in bytes.
3	offSet Integer	W	Number of bytes used in headerBuffer after formatting.
4	serialNumber Integer	R	Reference number.
5	systemName Bytes	R	Name of destination system.
6	portName Bytes	R	Name of remote port.

FUNCTION. . : Formats and codes a header for the XROUT 'Send Letter' service (XSLET). The letter is created according to the XROUT message format described in appendix B.

EXPLANATION : The routine will create the XROUT header that is required when a task wants to send a letter (service XSLET) to XROUT.

The header will be created and formatted in the user buffer specified by headerBuffer. serialNumber is put into byte 0 of the letter to allow the user task, which may have more than one request outstanding at the same time, to distinguish this letter from other messages. The port name specified by portName is appended as parameter 1, and the system name specified by systemName is appended as parameter 2 in the header.

systemName is the name of the system the letter will be sent to, whereas portName is the name of a (remote server) port in systemName that you want to contact. If the length of the name specified by systemName is 0, the local system is assumed.

Note that this routine just prepares the letter in a local buffer (headerBuffer). It does not copy the header into an XMSG message buffer, nor does it send anything. The data copying and sending must be done using routines such as XMPFWRI and XMPROUT.

The XMSG system provides task to task communication within the same system and between tasks running in different systems. When a task wants to send a message directly to another task, the sending task must know the magic number of a port belonging to the receiving task.

Since the magic number of a port (and the port number) is allocated by XMSG when a task opens a port, the sending task must obtain the magic number of the remote (receiving) port via XROUT (routing task).

When a task (usually a server task) has opened and named a port (e.g., using XMPOPCN or XMPOPNM), another task can now send a message from one of its own ports, via XROUT, to the named (remote) port. The header of the message sent via XROUT must contain a 'Send Letter' (XSLET) service request to XROUT. The remainder of the message can contain user data for the receiving (server) task (e.g., protocol information, user name, password etc.). The remainder of the message will not be looked at by XROUT (i.e., XROUT will only look at the header of the message - the letter), thus the user data can be (coded) in any format legible to the receiving (server) task.

When XROUT receives the letter, XROUT will look at systemName, and if systemName has been defined as a (remote) system name, XROUT will forward the letter (message) to the XROUT in the specified system.

The destination XROUT will look up the specified portName in its name table. If portName is a normal named port (i.e., a port named using XMPOPNM), then XROUT will forward the whole message to portName. If portName is a connection port (i.e., a port named using XMPOPCN), XROUT will look at the free connection counter for portName and if this is greater than zero, XROUT will decrement the counter and forward the whole message to portName. If there are no free connections, XROUT tries to find another port with the same name (portName) that have a free connection, and if found, XROUT will decrement the free connection counter and forward the message to that port.

When the (server) task receives the message on portName, it can, and it normally will, check that the sending task is allowed to use the server before it sends a (positive) reply to the requester, and thereby givies away its own magic number. If the server task does not want to give away its own magic number, it can do so by sending a (negative) reply with the XFFWD (forward message) option.

When the requesting task receives the (positive) reply from the (server) task, it can then use the XMPFMST routine to extract the magic number of the remote (server) port (portName), and direct communication with the remote (server) port can begin.

Note that if XMSG is unable to send the letter to XROUT in the specified system, or if the destination XROUT does not know the name of the destination port (i.e., if portName does not match any named port in systemName),

or if portName is a connection port with no free connections, then XROUT will return the letter (message) to the sending port with an error status. The requesting (sending) task can test whether the remote (server) task returned a reply or XROUT returned the message by checking the message type (which is returned as a result from a call to a 'receive message' or a call to a 'port status' routin).

Note that headerBuffer must start on an even byte boundary and that lengthBuffer must be big enough to contain the formatted XROUT 'Send Letter' service header. If one of these checks fails, -1 will be returned as error code in returnStatus.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats the header for an XROUT letter in our local buffer.

```
% We are going to send to a port named 's-port' in the
% system named 'scholar'.
  's-port' =: portName
  'scholar' =: systemName
% To recognize the message, we put in a reference no.
  111 =: serialNumber
% Give xmpblet more than enough space for formatting.
  60 =: lengthBuffer
% Let xmpblet create the necessary header.
  xmpblet(myBuffer,lengthBuffer,offSet,serialNumber,&
          systemName,portName) =: returnStatus
% Check returnStatus, and if Ok, then offSet
% number of bytes have been used for the XROUT
% header. (In this example, offSet=22.)
% (If 'scholar' is the name of our own system, we
% can of course create the letter specifying a
% system name length of zero instead.)
%  xmpblet(headerBuffer,lengthBuffer,offSet,&
%        serialNumber,systemName(0: -1),portName)
% We can now copy the service header from our
% local buffer into the header of a message, fill
% the remainder of the message with data for the
% receiving task, and send the message via XROUT
% using the appropriate routines.
% (If we need to send data to the receiving task, we
% must make sure that these data are written into the
% message after the XROUT header.)
```

Type: Buffer formatting

Routine name : XMPBLOC

No:	Parameter Name/ Type:	R/W	Explanation:
1	localBuffer Bytes	R	Local user buffer.
2	paramNumber Integer	R	Parameter number of the parameter that is to be found.
3	startOfParam Integer	W	Displacement within localBuffer in bytes.
4	paramType	W	Indicates the parameter type of paramNumber. The parameter type is returned as INTEGER or STRING.
5	paramLength Integer	W	Length of parameter in bytes.

FUNCTION. . : Locates a parameter within a buffer coded according to the XROUT message format described in appendix B.

EXPLANATION : Since the parameters in localBuffer may have been put into the buffer in a random order, this routine can be used to locate a specified parameter.

The routine will search through localBuffer for the parameter number specified by paramNumber. If the parameter is found, paramType indicates the parameter type of the located parameter, startOfParam indicates the position of the first significant byte in the parameter, and paramLength gives the number of significant bytes in the parameter. If no parameter with the requested parameter number is found, -1 is returned as error code in returnStatus.

Note that localBuffer must start on an even byte boundary. If not, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been coded according to the XROUT message format.
Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : The example locates parameter number 1 (which has been coded as a string parameter containing only the two characters H and I) in a user buffer containing:

```
Byte 0   = 123  Serial number
     1   = 0    Status from XROUT (0=Ok)
     2-3 = 32   Length of remainder of mess in bytes
     4   = -1   Param. no. 1 (negative means string)
     5   = 2    Length of parameter 1 in bytes
     6   = 72   First byte of parameter 1
     7   = 73   Second byte of parameter 1
     8-9 = 14   Length of parameter 2 in bytes
     etc.
```

```
% Specify and locate parameter number 1.
  1 =: paramNumber
  xmpbloc(localBuffer,paramNumber,startOfParam,&
          paramType,paramLength) =: returnStatus
% Check returnStatus, and if Ok, startOfParam=6,
% paramType=STRING and paramlength=2.
```

Routine name : XMPBRDY			Type: Buffer formatting
No:	Parameter Name/ Type:	R/W	Explanation:
1	outBuffer Bytes	R	Local user buffer.
2	serviceNumber Integer	R	XROUT service number.
3	serialNumber Integer	R	Reference number.

FUNCTION. . : Puts the serial number and the service number into a buffer which has been coded according to the XROUT message format described in appendix B.

EXPLANATION : When all the necessary parameters have been appended to outBuffer, this routine can be used to insert the serial number and the service number in the buffer. serialNumber will be put into byte 0 and serviceNumber will be put into byte 1 of the buffer.

Note that since this routine overwrites the first two bytes of the XROUT header descriptor with the serial number and the service number, this should be the last buffer preparation routine called, before the user buffer is written into a message and sent to XROUT. Note that this routine does not copy the local user buffer into any message; this must be done using routines such as XMPFWRI.

Note that outBuffer must start on an even byte boundary. If not, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMPBINI.
Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : The example build a buffer for the 'Send Letter' (XSLET) service request. It is similar to the behaviour of routine XMPBLET.

```
% Initialize the user buffer
  xmpbini(myBuff,lengthBuffer,offSet) =: returnStatus
% Check returnStatus, and if Ok, append
% the port name as parameter 1
  xmpbast(myBuff,offSet,portName,1) =: returnStatus
% Check returnStatus, and if Ok, append
% the system name as parameter 2
  xmpbast(myBuff,offSet,systemName,2) =: returnStatus
% Check returnStatus, and if Ok, we will insert
% the serial number and the service number XSLET
  xmpbrdy(myBuff,XSLET,serialnumber) =: returnStatus
% Check returnStatus, and if Ok, we can now write
% the buffer into a message and send the request
% to XROUT.
```

Routine name : XMPCLNM			Type: XROUT Service
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portName Bytes	R	Name of the port.
3	portNumber Integer	R	Number of port to be closed.

FUNCTION. . : Closes a port and clears the port name.

EXPLANATION : This routine can be used to close a named port that has been opened and named using XMPOPNM or XMPOPCN. The port specified by portNumber will be closed and the name assigned to portNumber will be cleared (i.e., the name will be removed from XROUT's name table).

When portNumber is closed, all nonsecure messages currently queued for that port are released, while all secure messages (as well as the 'port current' message, if any) are set nonsecure and returned to the sender.

The specified portNumber should be a port number returned from a call to XMPOPNM or XMPOPCN, and the specified portName should be the port name declared when the port was opened and named.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : This simple example will close a port and clear the name of a previously opened and named port.

```
% Specify the port's name and number, thus
  'xx-server' =: portName
  4 =: portNumber
% Now, close the port and remove its name
  xmpclnm(0,portName,portNumber) =: returnStatus
```

Routine name : XMPCONF			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	XMSGpassword Integer	W	XMSG password (the same as XMSG version code).
3	configMask Integer	W	Configuration mask. See below.
4	XMSGrestartCnt Integer	W	XMSG restart count.

FUNCTION. . : Gets information about the running XMSG system.

EXPLANATION : On return, XMSGpassword contains the password which is needed in order to become a privileged XMSG task (see routine XMPFPRV). XMSGrestartCnt returns the number of times that XMSG has been (re)started since the last warmstart.

The bits currently defined and returned in configMask are:

```
bit 0 : set if inter-system XMSG
    1 :  "  "  generated with tracing
    2 :  "  "  generated for ND-100
    3 :  "  "  file server for file transfer is incl.
    4 : is not used
    5 : set if running on page table 3
    6 :  "  "  generated for ND-100/CX instruction set
    7 :  "  "  generated with gateway software for
               network servers
```

Note that this bit mask, which is based on XMSG version J, will most certainly be extended in later XMSG versions.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpconf(0,XMSGpassword,configMask,&
                XMSGrestartCnt) =: returnStatus
```

Type: XMSG Function

Routine name : XMPFABR

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	userBuffer Xmuseraddress	R	Address of the user buffer.
3	userDisp Integer	R	Displacement within userBuffer in bytes. See note below.
4	readLength Integer	R	Number of bytes to read.
5	memoryDisp Integer	R	Address within bankNumber.
6	bankNumber Integer	R	The seven least significant bits (bit 0-6) specifies bank number.

FUNCTION. . : is absolute reading from the part of the physical memory used by XMSG.

EXPLANATION : This call allows a task to read a block of data from the physical memory into its user area specified by userBuffer (and userDisp).

If bankNumber is zero, a value for bankNumber equal to the bank in which the XMSG kernel code has been fixed is assumed. The data is read from the specified bank, starting from the address specified by memoryDisp, into the user buffer.

Note that on an ND-100, the userDisp is always rounded down to the previous even byte, if an odd displacement is specified.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in an ND-500.

EXAMPLE . . :
```
xmpfabr(0,userBuffer,userDisp,readLength,&
        memoryDisp,bankNumber) =: returnStatus
```

Routine name : XMPFALM			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	messageSize Integer	R	Message size in bytes.
3	numberOfMsgs Integer	R	Number of messages to allocate.

FUNCTION. . : Allocates message buffers to a task.

EXPLANATION : Normal message buffers that have been reserved using the XMPFGET call, lose their association with the task that got them when they are sent to another task. This implies that the sending task has no guarantee that it will be able to get space later.

By allocating message buffers, a task can indicate to XMSG its long-term buffer requirements. Allocated messages are removed from the free space pool, and marked as allocated to the original caller. They do change owners when sent within a system, but when released, or sent out of the local system, the message buffer is put back on the original allocator task's 'Available Allocated Message List (AAML)'.

All allocated messages for a given task must be of the same size. When an XMPFGET is executed by that task for a buffer of that size, XMSG will first look at the task's AAML and take a message buffer from it, if one is available. Similarly, when a message of that size comes into the system from another system, XMSG will first look at the AAML for the receiving task and take a message buffer from it if, one is available.

Note that if the routine call fails, due to lack of buffer space, no messages are allocated.

Continued on next page.

OPTIONS . . : XFEXC - Exclusive buffers. If set, the message buffers are allocated and set aside for exclusive use by the task, i.e., these message buffers will not be used by XMSG when a message of messageSize is received from another system. Buffers allocated with XFEXC do not have to be of the same size as buffers allocated without this option set. However, all exclusive buffers must be of the same size. To reserve one of these exclusive buffers, the task must call the XMPFGET routine with the XFEXC flag set.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : `xmpfalm(0,messageSize,numberOfMsgs) =: returnStatus`

Routine name : XMPFCLS			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options
2	portNumber Integer	R	Number of port to be closed

FUNCTION. . : Close(s) port(s).

EXPLANATION : Closes the specified local port. If portNumber is less than zero, all ports owned by the calling task will be closed. If portNumber is zero, the most recently opened port (i.e., the default port) will be closed.

When a port is closed, all nonsecure messages currently queued for that port are released, while all secure messages (as well as the 'port current' message, if any) are set nonsecure and returned to the sender. If the port had a name, the name is cleared (i.e., the name is removed from XROUT's name table).

See also the disconnect call, XMPFDCT.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : This will close all ports owned by this task:

```
-1 =: portNumber
xmpfcls(0,portNumber) =: returnStatus
```

Routine name : XMPFCPV			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	The identifier of a received message.
3	accessInfo Integer	W	Access information. See below.
4	additionalInfo Integer	W	Additional information. See below.

FUNCTION. . : Checks system and user privileges.

EXPLANATION : This call allows a task, when a message has been received, to check the privileges of the sender. If msgidentifier is not -1, the specified message becomes the 'task current' message.

If the sending task is allowed to update the routing tables (i.e., execute the privileged XROUT services XSDRN and XSDSY) on this system, then accessInfo = 1. If the message is sent from a task within the local system, then additionalInfo = 0. If the message is sent from a task in another system, then additionalInfo = 1.

If the sending task is not allowed to update the routing tables, then accessInfo = 0 and additionalInfo contains the reason:

additionalInfo = 0 implies that the sending task, as well as the source system are nonprivileged.
additionalInfo = 1 implies that the source system is privileged, but the sending task is not.
additionalInfo = 2 implies that the sending task is privileged, but the source system is not.
additionalInfo = 3 if the specified message is a returned message (it could not be delivered).

An nonprivileged task is a task which has not (yet) successfully executed the XMPFPRV call. An nonprivileged system is a remote system which has not (yet) been defined as a friend to your system, see XROUT service XSDAT.

Continued on next page.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfcpv(0,msgIdentifier,accessInfo,&
additionalInfo) =: returnStatus

Routine name : XMPFCRD			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	interruptLevel Integer	R	The interrupt level that the driver should run on.
3	registerBlock Xmuseraddress	R	Address of register block (an 8-word buffer).
4	XTblockAddress Integer	W	Address of the XT-block allocated to the driver.

FUNCTION. . : Defines a driver for XMSG.

EXPLANATION : This call is used to define an already existing driver, with a context as specified by registerBlock. The buffer must contain the register block that the driver will be started with, in the order required for the Load Register Block (LRB) hardware instruction. XMSG will allocate a task block (XT-block) to the driver and return its address in XTblockAddress.

OPTIONS . . : XFPON - Paging on. This must be set if the driver is running with paging on.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in ND-500.

EXAMPLE . . :
```
2**XFPON =: flags
xmpfcrd(flags,interruptLevel,registerBlock,&
          XTblockAddress) =: returnStatus;
```

Routine name : XMPFDBK			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	bankNo Integer	R	Bank number.

FUNCTION. . : Defines a bank number for drivers.

EXPLANATION : When calling routines which transfer data between a user area and an XMSG buffer (e.g., XMPFREA, XMPFWRI or XMPFSMC), drivers specify a physical address as user buffer (the userAddress parameter). This is in bank 0, unless they have previously defined a bank number using the XMPFDBK call.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.
Not permitted for RT-programs.
Not available for tasks running in ND-500.

EXAMPLE . . : xmpfdbk(0,bankNo) =: returnStatus

Routine name : XMPFDCT			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.

FUNCTION. . : Disconnects from XMSG.

EXPLANATION : Releases all XMSG resources. All ports opened by the task are closed and all XMSG space belonging to the current caller is released. Special action is taken in the case of current messages, and messages waiting on the input queue (see XMPFSND, XMPFRCV and XMPFCLS).

Note that the following automatic disconnects are executed by SINTRAN:

User disconnect:
- On return to the SINTRAN command processor
- On log out or RT program termination

System mode disconnect:
- On log out or RT program termination

There is no return from a driver call to XMPFDCT (as the driver context is released by the call).

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfdct(0) =: returnStatus;

Routine name : XMPFDMM			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	requestedTaskSp Integer	R	Requested task space in bytes.

FUNCTION. . : Defines maximum limit of memory usage.

EXPLANATION : When a new task is defined in XMSG, its maximum buffer space is set to a predefined value (defined when the XMSG system is generated). This can be changed for privileged tasks using this call. requestedTaskSp will be set equal to the maximum number of bytes of message space that can be owned by the task at one time.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.

EXAMPLE . . : xmpfdmm(0,requestedTaskSp) =: returnStatus

Type: XMSG Function

Routine name : XMPFDUB

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	bufferAddress Integer4	R	Address of physical memory buffer.
3	bufferSize Integer	R	Buffer size in bytes.

FUNCTION. . : Defines a user buffer.

EXPLANATION : This is a privileged call that allows a task to associate a physical memory buffer with a message descriptor previously obtained by XMPFGET with sizeBuffer = 0. All XMSG calls then operate on that message, as the buffer space was part of the general XMSG buffer pool, except that XMPFREL only releases the message descriptor and not the buffer area.

This allows special systems or drivers to fully control their memory allocation procedures.

This call acts on the 'task current' message.

Buffers that have been defined in this way cannot be sent to other systems.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not available for tasks running in an ND-500.

EXAMPLE . . : xmpfdub(0,bufferAddress,bufferSize) =: returnStatus

Routine name : XMPFDUM			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.

FUNCTION. . : Dummy call.

EXPLANATION : This call may be useful if the programmer wants to check that XMSG is up and running. It is also useful for benchmarking.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfdum(0) =: returnStatus

Type: XMSG Function

Routine name : XMPFFRM

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	noOfMsgToFree Integer	R	Number of allocated message buffers to free.
3	noOfMsgFreed Integer	W	Number of message buffers actually freed.

FUNCTION. . : Frees allocated message buffers.

EXPLANATION : This call frees message buffers which have been allocated by XMPFALM.

OPTIONS . . : XFEXC - Exclusive buffers. If set, only those message buffers which have been allocated with the XFEXC option set will be freed. If not set, only those message buffers which have been allocated without the XFEXC option will be freed.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFEXC =: flags
 xmpffrm(flags,noOfMsgToFree,noOfMsgFreed)&
      =: returnStatus
```

Routine name : XMPFGET			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	sizeBuffer Integer	R	Number of bytes requested.
3	msgIdentifier Xmmsgidentifier	W	Message identifier.

FUNCTION. . : Reserves a message buffer.

EXPLANATION : msgIdentifier is returned to the caller for possible use in subsequent routine calls. Each buffer size has a maximum, system dependent size defined when the XMSG system is generated. The total XMSG buffer space owned by a task cannot exceed another limit, which is initially set to a value also defined at XMSG generation time. It may be changed, however, by privileged tasks using the Define Maximum Memory (XMPFDMM) routine.

Only the current owner of a message is allowed to read or write in it, give it to someone else, or release it.

Specifying a buffer size of 0 bytes implies that only a message descriptor will be reserved. Privileged tasks can then associate a physical memory area with that message descriptor using the XMPFDUB routine. It is not allowed to send a buffer of size 0 to another system.

OPTIONS . . : XFWTF - Wait flag. If no message buffer of the requested size is available, the task will be suspended. Execution resumes when a buffer becomes available.

XFEXC - Exclusive buffer. If set, it implies that the caller wants to reserve exclusively a message buffer allocated using the XMPFALM. If no such allocated message buffer is available, an error status is returned.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
20 =: sizeBuffer
xmpfget(0,sizeBuffer,msgIdent) =: returnStatus
```

Routine name : XMPFGST			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	startScanPort Integer	R	Last port to be scanned.
3	portNumber Integer	W	Port number where the message is waiting.

FUNCTION. . : A task may have many open ports. It does not always know on which one the next message is arriving. XMPFGST allows the task to check all ports belonging to the task, i.e., it allows the task to find out whether any messages are waiting on any port.

EXPLANATION : The parameter startScanPort specifies the last port to be searched. If startScanPort is zero, this implies the most recently opened port (i.e., the default port). Note that the search will begin with the next port (if any) after that specified, and then follow the task's port list (see example below).

On return, the parameter portNumber contains the port number where the message, if any, is waiting. If no messages are waiting and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

For example, if the task has opened four ports and have got the port numbers 15 (from the 1st XMPFOPN), 4 (from the 2nd XMPFOPN), 6 (from the 3rd XMPFOPN) and 19 (from the 4th XMPFOPN), then the port list comprises the ports 19-6-4-15 (in that order!). Port number 19 (the first port in the list) is the task's default port (i.e., if startScanPort is zero, this port is assumed). If the task has just handled a message received on port 6, it can, when it wants to have a 'round-robin' scheduling of requests, call XMPFGST with startScanPort=6. Port 6 will then be the last port to be looked at by XMSG. XMSG will start looking at port 4 to see if a message is waiting. If no message is waiting on port 4, XMSG will look at port 15. If no message is waiting on port 15, XMSG will look at port 19, and, if no message is waiting on port 19, XMSG will finally look at port 6.

Note that calling this routine, when a message is waiting on one of the ports will lead, to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on the returned portNumber. If no message

is waiting on any of the ports, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on all ports opened by the task before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMPFGST call, and the 'wake up' bit on the individual ports may have been set as a result from previously executed XMPFPST, XMPFRCV, XMPFRRH or XMPFRRE calls.)

OPTIONS . . : XFWTF - Wait flag. If set, the task is suspended if no messages are waiting. Execution resumes when a message arrives on one of the ports.

XFWAK - General wake up. Unless a message is already waiting on one of the ports, a 'general wake up' bit will be set for this task. When 'general wake up' is set, the next transmission to any of the ports opened by the task will lead to a wake up of the receiver task, and clearing of the 'general wake up' bit for that task.

However, be aware that if the task is in XMSG wait position (for example, if sending a secure message with wait), i.e. when the task should have been woken up as a result of a message being sent to one of its ports, the 'general wake up' bit will be cleared but, the task will not (and cannot) be woken up.

XFHIP - High priority message. Allows a task to check the arrival of high priority messages. If a high priority message is waiting on one of the ports and XFHIP is set, the port number where the high priority message is waiting is returned in portNumber. If no high priority message is waiting on any of the ports and XFHIP is set, and XFWTF is not set, the base value of the error codes (XMXENTM) is returned as returnStatus, and when the next message of any type is sent to a port opened by the task, the task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFWTF =: flags
4 =: startScanPort
xmpfgst(flags,startScanPort,portNumber) =: returnStatus
```

Type: XMSG Function

Routine name : XMPFLMP

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier or 0.
3	portNo Integer	R	Port number or 0.
4	msgIdentFound Xmmsgidentifier	W	Message ID for the first message found equal to or greater than requested.
5	msgSize Integer	W	Message size of msgIdentFound in bytes.
6	portNoFound Integer	W	Port no. of the first port found equal to or greater than requested.

FUNCTION. . : Lists messages and ports.

EXPLANATION : This call allows a task to obtain information about its own open ports and its own messages.

msgSize contains the size of the returned msgIdentFound in bytes. msgIdentFound has either been reserved using the XMPGET call, using the XMPFALM, or been received as a result of the message being sent from another task.

If there is no message identifier found equal to or greater than the requested msgIdentifier, then msgIdentFound is 0.

If there is no port found equal to or greater than the requested portNo, then portNoFound is 0.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpflmp(0,msgIdentifier,portNo,msgIdentFound,&
        msgSize,portNoFound) =: returnStatus
```

Routine name : XMPFM2P			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	magicNumber Integer4	R	Magic number.
3	portNumber Integer	W	Port number.
4	systemNumber Integer	W	System number.
5	RTinOrOtherInfo Integer	W	RT index or other information. See below.
6	additionalInfo Integer	W	Additional information. See below.

FUNCTION. . : Converts magic number to a port and system number.

EXPLANATION : This call allows a task to convert the magic number to a port number and a system number.

RTinOrOtherInfo may contain additional information about the port owner task. If the magic number was that of a system or that of a remote port, then RTinOrOtherInfo = -1. If the magic number was that of a local port and the port owner task is a driver, then RTinOrOtherInfo = -2. If the magic number was that of a local port and the port owner task is an RT-program, then RTinOrOtherInfo equals the SINTRAN RT-index of the RT-program.

additionalInfo may contain additional information about the specified magic number. If the magic number was that of a system, then additionalInfo = 3. If the magic number was that of a local port and the port owner task is privileged, then additionalInfo = 2. If the magic number was that of a remote port, or that of a local port and the port owner task is nonprivileged, then additionalInfo = 1.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfm2p(0,magicNumber,portNumber,systemNumber,&
        RTinOrOtherInfo,additionalInfo) =: returnStatus
```

Type: XMSG Function

Routine name : XMPFMST

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	msgType Integer	W	Message type, see explanation below.
4	remoteMagicNum Integer4	W	Magic number of sending port.
5	msgLength Integer	W	Message length in bytes.

FUNCTION. . : Obtains message status.

EXPLANATION : This call allows a task to extract the sender's magic number, and get the length and type of a received message. If msgIdentifier is not -1, the specified message becomes the 'task current' message.

Message type: XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent secure, but could not be delivered)

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfmst(0,msgIdentifier,msgType,remoteMagicNum,&
        msgLength) =: returnStatus
```

Routine name : XMPFOPN			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portNumber Integer	W	Port number.

FUNCTION. . : Opens a port.

EXPLANATION : A port is opened and the port number (i.e., the port identifier) is returned in portNumber.

The opened port becomes the task's default port. When this port later is closed, the previously opened port, if any, becomes the task's default port.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfopn(0,portNumber) =: returnStatus

Routine name : XMPFP2M			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portNumber Integer	R	Port number.
3	magicNumber Integer4	W	Magic number.

FUNCTION. . : Converts port number to magic number.

EXPLANATION : This call allows a task to convert a local port number to a magic number. Any task may obtain the magic number of its own ports. Privileged tasks can obtain the magic number of a port owned by another local task.

Note that this routine will only return the magic number of ports opened by tasks in the local system.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfp2m(0,portNumber,magicNumber) =: returnStatus

Routine name : XMPFPRV			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	XMSGPassword Integer	R	XMSG password

FUNCTION. . : Make the calling task privileged.

EXPLANATION : Some of the routine calls can only be executed by privileged XMSG tasks. In order to become privileged (for XMSG), a task must successfully execute the XMPFPRV call. When the task stops being privileged, the same call should be used, but with XMSGPassword equal to zero. The reason for specifying the XMSG password is to ensure that privileged programs, that base themselves on accessing XMSG table structures, have been updated to the current XMSG table definitions.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : The caller must be either a driver, a direct task, a foreground program, or a background program logged in as user system. Besides this, the program must also specify the current XMSG password, which can be obtained using the XMPCONF routine.

EXAMPLE . . :
```
xmpfprv(0,XMSGPassword) =: returnStatus
```

Routine name : XMPFPST			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	localPort Integer	R	Port number to be checked.
3	msgType Integer	W	Message type, see explanation below.
4	remotePort Integer	W	Hashed magic number of remote port.
5	msgIdentifier Xmmsgidentifier	W	Message identifier.
6	queueLength Integer	W	Number of messages queued for localPort.

FUNCTION. . : Checks a port to see if any message is waiting.

EXPLANATION : If localPort is zero, the most recently opened port (i.e. the default port) is assumed.

On return from the routine, msgType indicates the message type of the first message queued to localPort. If no message is waiting, msgType is zero. If a message is waiting, the remotePort, msgIdentifier and queueLength parameters will contain the hashed magic number of the sending port, the message address and the number of messages chained to localPort.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

Note that calling this routine when a message is waiting on localPort will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on localPort. If no message is waiting on localPort, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on localPort before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMPFGST call, and the 'wake up' bit on localPort may have been set as a result from a previously executed XMPFPST, XMPFRCV, XMPFRRH or XMPFRRE call.)

Continued on next page.

Message type: XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent secure but, could not be delivered)

OPTIONS . . : XFWTF - Wait flag. If no message is waiting on localPort, the task is suspended. Execution resumes when a message arrives on localPort.

XFWAK - Wake up. Unless a message is already waiting on localPort, a 'wake up' bit will be set on localPort. When 'wake up' is set on localPort, the next transmission to this port will lead to a wake up of the receiver task, and clearing of the 'wake up' bit on localPort. This option can be enabled on more than one port at a time.

However, be aware that if the task is in XMSG wait position (for example, sending a secure message with wait), when the task should have been woken up as a result of a message being sent to localPort, the 'wake up' bit will be cleared but the task will not (and cannot) be woken up.

XFHIP - High priority message. Allows a task to check the arrival of high priority messages. If a high priority message is waiting and XFHIP is set, the message type XMTHI is returned in msgType. If no high priority message is waiting, and XFHIP is set and XFWTF is not set, a zero (0) is returned in msgType. When the next message of any type is sent to localPort, the receiving task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfpst(flags,localPort,msgType,remotePort,&
        msgIdentifier,queueLength) =: returnStatus
```

Type: XMSG Function

Routine name : XMPFRCV

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	localPort Integer	R	Number of the receiving port.
3	msgType Integer	W	Message type, see explanation below.
4	remotePort Integer	W	Hashed magic number of remote port.
5	msgIdentifier Xmmsgidentifier	W	Message identifier.
6	msgLengthOrStat Integer	W	Message length in bytes. If msgType is XMTRE, msgLengthorStat contains the error status.

FUNCTION. . : Receives a message when it is queued for a port.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue) and its address returned in msgIdentifier. msgType indicates the message type of the received message, msgLengthOrStat gives the message length and remotePort contains the hashed magic number of the sending port. If the message type is XMTRE (returned message), then msgLengthOrStat contains the reason for return.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

A successful receiving causes the received message to become the 'task current' message. In addition, if it is a secure message (i.e., a message sent with option XFSEC set), it becomes the 'port current' message for localPort. If the task aborts or localPort is closed while the message is 'port current', the message will be returned to the sender with 'return' status.

The 'task current' message is cleared by releasing/sending it to someone else, or receiving another message. The 'port current' message is cleared by releasing/sending it to someone else, or receiving another secure message. A task may also change the value of the current message using the XMPFSCM routine.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

Note that calling this routine when a message is waiting on localPort will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on localPort. If no message is waiting on localPort, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on localPort before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMPFGST call, and the 'wake up' bit on localPort may have been set as a result from a previously executed XMPFPST, XMPFRCV, XMPFRRH or XMPFRRE call.)

Message type: XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent secure but could not be delivered)

OPTIONS . . : XFWTF - Wait flag. If no message is waiting on localPort, the task is suspended. Execution resumes when a message arrives on localPort.

XFWAK - Wake up. Unless a message is already waiting on localPort, a 'wake up' bit will be set on localPort. When 'wake up' is set on localPort, the next transmission to this port will lead to a wake up of the receiver task, and clearing of the 'wake up' bit on localPort.

When the wake up is done, the message is not received, and so the receiving must be repeated. This option can be enabled on more than one port at a time.

However, be aware that if the task is in XMSG wait position(for example, sending a secure message with wait), when the task should have been woken up as a result of a message being sent to localPort, the 'wake up' bit will be cleared, but the task will not (and cannot) be woken up.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFWTF =: flags
 XMPFRCV(flags,localPort,msgType,remotePort,&
       msgIdent,msgLengthOrStat) =: returnStatus
```

Type: XMSG Function

Routine name : **XMPFREA**

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgDisp Integer	R	Displacement within message in bytes.
3	userAddress Xmuseraddress	R	Address of user buffer.
4	userDisp Integer	R	Displacement within userAddress in bytes. See note below.
5	userLength Integer	R	Number of bytes you want to read.
6	readLength Integer	W	Number of bytes actually read.

FUNCTION. . : Reads user data from a message buffer.

EXPLANATION : The data is read from the 'task current' message, starting with displacement msgDisp (rounded up to the next even byte), into the user buffer specified by userAddress (and userDisp). readLength is returned to indicate the actual number of bytes read. If msgDisp is -1, the reading of the message is resumed from the current message displacement.

Note that the displacement within the message is always rounded up to the next even byte and, on a ND-100, that userDisp is always rounded down to the previous even byte before the data is read.

On return, if the last byte in the message is read, the current message displacement is set to 0, and the 'whole-message-read' flag is set, so that the next 'write message' call (e.g., XMPFWRI or XMPFWHD, will reset the current message length to zero. Otherwise, except when readLength is zero, the current message displacement is set to msgDisp+readLength, where msgDisp is the specified displacement (rounded up if necessary) and readLength is the actual number of bytes transferred. If readLength is zero, the current message displacement is not updated.

Continued on next page.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfrea(0,msgDisp,&
        addr(inbuffer(0)) force Xmuseraddress,userDisp&
        ,userLength,readLength) = : returnStatus
```

Routine name : XMPFREL			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.

FUNCTION. . : Releases a message buffer.

EXPLANATION : This call is used to release a message buffer reserved by the task. A message buffer is reserved when the task issues the XMPFGET call or when a message is sent to it from another task. In the latter case the message must be received before it can be released.

At any particular time, the total message buffer space owned by a task cannot exceed a limit defined when the XMSG system is generated. Therefor, as, a general rule for a task should be its message buffer release, as soon as the task is through with it.

A msgIdentifier parameter of -1, will release the 'task current' message.

If the specified message is an allocated message (i.e., a message allocated using the XMPFALM call), the message will be put back on the original task's 'Available Allocated Message List' (AAML), see the routine XMPFALM.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfrel(0,msgIdentifier) =: returnStatus

Type: XMSG Function

Routine name : XMPFRHD

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	bytes0To1 Integer2	W	Bytes 0 and 1 of message header.
4	bytes2To3 Integer2	W	Bytes 2 and 3 of message header.
5	bytes4To5 Integer2	W	Bytes 4 and 5 of message header.

FUNCTION. . : Reads only the header of a message buffer.

EXPLANATION : The first 6 bytes of a message buffer are read and returned in bytes0To1, bytes2To3 and bytes4To5, and then the current message displacement is set to 6.

If msgIdentifier equals -1, the data will be read from the 'task current' message. If not -1, the specified message becomes the 'task current' message before the data is read.

If the message size is less than 6 bytes, an error return occurs.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfrhd(0,msgIdentifier,bytes0To1,bytes2To3,&
        bytes4To5) =: returnStatus
```

Routine name : XMPFRRE			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	localPort Integer	R	Number of the receiving port.
3	userAddress Xmuseraddress	R	Address of user buffer.
4	userDisp Integer	R	Displacement within userAddress in bytes. See note below.
5	userLength Integer	R	Number of bytes you want to read.
6	msgType Integer	W	Message type, see explanation below.
7	remotePort Integer	W	Hashed magic number of the remote port.
8	msgIdentifier Xmmsgidentifier	W	Message identifier.
9	msgLengthOrStat Integer	W	Message length in bytes. If msgType is XMTRE, msgLengthorStat contains the error status.

FUNCTION. . : Receives a message queued for a port and read from the message buffer.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue) and then userLength number of bytes will be read from the first byte in the message buffer into the user buffer specified by userAddress (and userDisp). If the last byte in the message is read, the current message displacement is set to 0, and the 'whole-message-read' flag is set. Thus the next 'write message' call will reset the current message length to zero. Otherwise, if the last byte is not read, the current message displacement is set to the actual number of bytes read.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed. If userLength is greater than the message length, only msgLength OrStat number of bytes will be read into the user buffer.

Note that on an (ND-100, the userDisp (displacement within the user buffer) is always rounded down to the previous even byte.

Note that when the message is received, both the 'task current' message and the 'port current' message will be set as described under routine XMPFRCV. Note also that the handling of flags (options) in this routine is identical to the handling described under XMPFRCV. Also, the return parameters from the routine are identical to the return parameters from the XMPFRCV call.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

You should note that this routine act as if both XMPFRCV and XMPFREA had been called. Calling this routine, instead of the other two routines, however eliminates the overhead associated with each routine and XMSG call.

Message type: As for XMPFRCV.

OPTIONS . . : As for XMPFRCV.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFWTF =: flags
 xmpfrre(flags,localPort,useraddress,userDisp,&
     userLength,msgType,remotePort,msgIdent,&
     msgLengthOrStat) =: returnStatus
```

Type: XMSG Function

Routine name : **XMPFRRH**

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	localPort Integer	R	Number of the receiving port.
3	msgType Integer	W	Message type, see explanation below.
4	remotePort Integer	W	Hashed magic number of the remote port.
5	msgIdentifier Xmmsgidentifier	W	Message identifier.
6	bytesOTolorStat Integer2	W	Normally first 2 bytes of message. If msgType is XMTRE, bytesOTolorStat contains the error status.

FUNCTION. . : Receives a message and reads the header.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue). Then the first two bytes of the message buffer are read and returned in the bytesOTolorStat parameter.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

Note that when the message is received, both the 'task current' message and the 'port current' message will be set as described under routine XMPFRCV. Note also that the handling of flags (options) in this routine is identical to the handling described under XMPFRCV. The return parameters from the routine are identical to the return parameters from the XMPFRCV call, except that the first two bytes of user data is returned instead of the message length.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

Message type: As for XMPFRCV.

Continued on next page.

OPTIONS . . : As for XMPFRCV.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFWTF =: flags
xmpfrrh(flags,localPort,msgType,remotePort,&
      msgIdentifier,bytesOTolorStat) =: returnStatus
```

Type: XMSG Function

Routine name : XMPFRTN

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	localPort Integer	R	Number of the sending port.
4	bytesOTol Integer2	R	First 2 bytes of the message header.

FUNCTION. . : Returnes a message to the port from which it came.

EXPLANATION : The user often needs to write a return status into a message and send it back to the port from which it came (e.g., replying to a transaction). This call leads to msgIdentifier being set as the 'task current' message and the 'port current' message for localPort, bytesOTol, being written into the first two bytes of the message buffer. Then the message is being returned to the port from which it was last sent.

If msgIdentifier is -1, the current message is assumed to be either the 'port current' message, if one exists, or, if none, the 'task current' message.

The localPort parameter specifies the port from which the message will be sent. If localPort is zero, the most recently opened port (i.e. the default port) is assumed.

OPTIONS . . : XFWTF - Wait flag. This is only significant when sending a secure (XFSEC) message to a task in another system.

If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port).

If not set, secure messages that cannot be delivered will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered or if the receiving port is closed (e.g., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFHIP - High priority message. The message will be chained to the head of the receiver's queue instead of the tail, following any other high priority messages already queued.

XFFWD - Forwarding message. The sender information in the message will not be updated. To the receiver, it will appear that the message was sent directly from the previous sending port.

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., the routines XMPFRCV, XMPFRRH or XMPFRRE), which would have led to this message being received, it will instead be returned to the sender.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : `xmpfrtn(0,msgIdentifier,localPort,data0) =: returnStatus`

Routine name : XMPFSCM			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portNumber Integer	R	Port number.
3	msgIdentifier Xmmsgidentifier	R	Message identifier.

FUNCTION. . : Sets the current message.

EXPLANATION : Since many routines implicitly operate on the current message, it is useful to be able to set the latter. This call sets the specified message as the 'task current' message. If portNumber is >=0, the message is also set as 'port current' for the specified port. If portNumber is zero, the most recently opened port (i.e., the default port) is assumed.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfscm(0,portNumber,msgIdentifier) =:returnStatus

Routine name : XMPFSIN			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	XMSGbase Integer	W	XMSG base field address.

FUNCTION. . : Gets XMSG's base field address.

EXPLANATION : This call returns the base field address of the XMSG system in the memory bank, where the XMSG kernel code has been fixed. This address is needed in order to be able to access XMSG tables.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in an ND-500.

EXAMPLE . . : xmpfsin(0,XMSGbase) =: returnStatus

Type: XMSG Function

Routine name : XMPFSMC

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	noOfCalls Integer	R	Number of XMSG functions to be executed
3	userAddress Xmuseraddress	R	Address of user buffer containing the parameters.
4	userDisp Integer	R	Displacement within userAddress in bytes. See note below.
5	Treg Integer	W	The content of the T register.
6	Areg Integer	W	The content of the A register.
7	Dreg Integer	W	The content of the D register.
8	Xreg Integer	W	The content of the X register.

FUNCTION. . : Starts multi call.

EXPLANATION : This call allows a task to execute a set of XMSG functions issuing only one routine call. This eliminates the overhead associated with each routine call (and XMSG monitor call).

noOfCalls is the number of XMSG functions to be executed and userAddress is the address of a buffer containing the parameters for the functions. Each set of parameters comprise 4 words (T, A, D and X registers), so the buffer length should be 8*noOfCalls bytes long. noOfCalls has a maximum, system dependent size defined when the XMSG system is generated. If noOfCalls is 0 (or -1), then the previously executed multi call request will be re-executed.

Note that on an ND-100, the userDisp is always rounded down to the previous even byte.

The meaning of the T, A, D and X registers depend on the particular XMSG function. A documentation of the XMSG functions is provided in appendix A.

XMPFSMC returns as soon as an XMSG function terminates with status less than or equal to zero (or when all the functions have been executed). The return parameters (Treg, Areg, Dreg and Xreg) are set according to the return registers from the last XMSG function executed. Completion status is also returned in the returnStatus,

XMOK, if the multi call has been successfully executed, in XMXENTM, if one of the XMSG functions in the multi call was not terminated, otherwise returnStatus contains an error code.

You should be aware of the fact that if an XMSG disconnect function is specified (and executed) as one of the functions in the multi call, the succeeding functions in the multi call will not be executed, as the task context (XT-block) is released by the disconnect (XFDCT) function.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks. Not available for tasks running in an ND-500.

EXAMPLE . . :
```
xmpfsmc(0,noOfcalls,userAddress,userDisp,&
        Treg,Areg,Dreg,Xreg) =: returnStatus
```

Routine name : XMPFSND			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	localPort Integer	R	Number of the sending port.
3	remoteMagicNum Integer4	R	Magic number of the receiving port.

FUNCTION. . : Sends the current message to another task.

EXPLANATION : When a task wants to send a message to another task, it must know the magic number of a port of the other task. For a description of how to obtain the magic number, see under the XMPBLET call, and in the example in appendix G. A remoteMagicNum parameter of -1 will direct the message back to the port from wich it was last sent.

The localPort parameter specifies the port from which the message will be sent. If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

Note that there is no parameter specifying the message that is to be sent, for the reason that the current (default) message buffer is assumed, namely the 'port current' message if one exists, or, if none, the 'task current' message.

OPTIONS . . : XFWTF - Wait flag. This is only significant when sending a secure (XFSEC) message to a task in another system.

If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port).

If not set, secure messages that cannot be delivered will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered, or if the receiving port is closed (e.g., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFFWD - Forwarding message. The sender information in the message will not be updated. To the receiver, it will appear that the message was sent directly from the previous sending port.

XFROU - Route message. Ignore the remoteMagicNum parameter and send the message to the local routing task (XROUT). The message contents should be parameters to XROUT. (See appendix B on XROUT services.)

XFRRO - Remote route message. If the XFROU flag is also set, then send the message to a remote routing task (XROUT). The 16 most significant bits of remoteMagicNum is assumed to contain the system number to which the message will be sent. The message contents should be parameters to XROUT.

Note that if the XFROU flag is not set and XFRRO is set, the message will be sent as if XFHIP had been set.

XFHIP - High priority message. If the XFROU flag is not set, the message will be chained to the head of the receiver's queue, instead of the tail, following any other high priority messages already queued.

Note that if both the XFROU flag and the XFHIP flag are set, the message will be sent as if XFROU and XFRRO had been set (i.e., when XFROU is set, setting the XFHIP flag will act as if the XFRRO flag had been set instead).

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., the routines XMPFRCV, XMPFRRH or XMPFRRE), which would have led to this message being received, it will instead be returned to the sender.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFSEC + 2**XFHIP =: flags
xmpfsnd(flags,localPort,remoteMagicNum) =: returnStatus
```

Routine name : XMPFSTD			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	XTblockAddress Integer	R	The address of the driver task block.

FUNCTION. . : Starts driver.

EXPLANATION : This call starts the execution of a driver which has already been defined by the XMPFCRD call.

XTblockAddress must contain the driver's task block address as returned from the XMPFCRD call. XMPFSTD overwrites the driver's L register with the XTblockAddress before starting the driver.

In this way a started driver will have the L register containing its XT-block address. The driver must make sure that the L register still contains the XT-block address before calling XMSG.

XMPFSTD does not set the appropriate bit in the PIE register. Nor does it load or fix any segments. This should be done using the FIXC and ENTSG monitor calls.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in ND-500.

EXAMPLE . . : `xmpstd(0,XTblockAddress) =: returnStatus`

Routine name : XMPFWDF			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	Bregister Integer	R	The B register of the driver on restart.
3	restartAddress Integer	R	Restart address for the driver.

FUNCTION. . : Defines wake up context.

EXPLANATION : If a driver uses the XFWAK (wake up) option, XMSG must be told where to restart the driver. This is done by using the XMPFWDF call. When the driver is restarted by XMSG, it will be restarted in the address specified by restartAddress with its B register set to the address specified by Bregister.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.
Not permitted for RT-programs.
Not available for tasks running in an ND-500.

EXAMPLE . . : xmpfwdf(0,Bregister,restartAddress) =: returnStatus

Type: XMSG Function

Routine name : XMPFWHD

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	bytes0To1 Integer2	R	Bytes 0 and 1 of the message header.
3	bytes2To3 Integer2	R	Bytes 2 and 3 of the message header.
4	bytes4To5 Integer2	R	Bytes 4 and 5 of the message header.

FUNCTION. . : Writes to the header of the 'task current' message buffer.

EXPLANATION : If the 'whole-message-read' flag has been set (see XMPFREA), it is cleared and the current message length (not the same as size) is set to 0. Then the routine inserts bytes0To1, bytes2To3 and bytes4To5 as the first six bytes of the message. If this results in the message being longer than before, the current message length is set to 6. It then sets the current message displacement to 6.

If the message lenght is less than 6 bytes, an error return occurs.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : xmpfwhd(0,bytes0To1,bytes2To3,bytes4To5) =: returnStatus

Routine name : XMPFWRI			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgDisp Integer	R	Displacement within message in bytes.
3	userAddress Xmuseraddress	R	Address of the user buffer.
4	userDisp Integer	R	Displacement within userAddress in bytes. See note below.
5	userLength Integer	R	Number of bytes you want to write.
6	writtenLength Integer	W	Number of bytes actually written.

FUNCTION. . : Writes user data into a message buffer.

EXPLANATION : After building up a data buffer in its own space, a task transfers the data buffer into the 'task current' message buffer using XMPFWRI. If the 'whole-message-read' flag has been set (see XMPFREA), it is cleared and the current message length (not the same as size) is set to 0. If msgDisp is -1, a value for msgDisp equal to the current message displacement is assumed instead, thus providing an appending function. If msgDisp is odd, 1 is added to it, and a zero bytes inserted in the message.

If msgDisp+userLength is greater than the message size, an error return occurs. Otherwise, userLength bytes are copied from the user buffer into the message buffer, and the current message displacement is set to msgDisp+writtenLength (where msgDisp has been rounded up, if odd). If this copying resulted in the message being longer than before, the current message length is also set to msgDisp+writtenLength. writtenLength is returned to indicate the actual number of bytes transferred.

Note that the displacement within the message is always rounded up to the next even byte and, on an ND-100, that userDisp is always rounded down to the previous even byte before the data is written.

Continued on next page.

OPTIONS . . : XFRES - Reset current message length. If set, it leads to the current message length being set to 0 before the user data is transferred into the message buffer. (In fact it acts as if the 'whole-message-read' flag had been set.)

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpfwri(flags,msgDisp,&
        addr(inbuffer(0)) force Xmuseraddress,&
        userDisp,userLength,writtenLength)&
        =: returnStatus
```

Routine name : XMPINFC	Type: XROUT Service

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portNumber Integer	R	Port number returned by XMPOPCN.
3	extraConn Integer	R	Number of extra connections.
4	serialNumber Integer	R	Reference number.

FUNCTION. . : Increments (or decrements) the free connection count.

EXPLANATION : After opening a connection port using XMPOPCN, a task can later increment (when the connections become available) or decrement (when number of connections need to be reduced) the free connection counter associated with that port.

If extraConn is positive, the maximum number of connections that portNumber can handle is increased. If extraConn is negative, the maximum number of connections that portNumber can handle will be decreased. If the resulting number of free connections becomes negative, an error status will be returned from XROUT.

Note that this routine will not wait for a reply from XROUT, and so the caller will later receive this reply from XROUT on the port specified by portNumber. serialNumber is put into byte 0 of the request sent to XROUT to allow the caller, who may have many requests outstanding at the same time, to recognize the reply.

Note that since this routine has to reserve and send a message to XROUT to increase (or decrease) the number of connections accepted, the routine will change the task's current definition of 'task current' message, as well as the current definition of 'port current' message on portNumber, if any.

Continued on next page.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : When a server port with port number 12, which has previously been created using XMPOPCN, is able to handle a new connection, we should inform XROUT.

```
% We are able to handle one more connection
  1 =: extraConn
% on the port number returned from xmpopcn
  12 =: portNumber
% To recognize the reply, we need a reference no.
  100 =: serialNumber
% Tell XROUT
  xmpinfc(0,portNumber,extraConn,serialNumber)&
         =: returnStatus
% Check returnStatus, and if Ok, the request has
% been sent to XROUT.
```

Routine name : XMPOPCN			Type: XROUT Service
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portName Bytes	R	Name of the port.
3	uniqueName Boolean	R	Uniqueness flag.
4	maxConnections Integer	R	Maximum number of connections accepted.
5	portNumber Integer	W	Port number.

FUNCTION. . : Creates a connection port.

EXPLANATION : This call is very similar to XMPOPNM, but allows XROUT to control the number of connections that a port can handle simultaneously, and even distribute connections among server (connection) ports.

As for XMPOPNM, a port is opened and its port number is returned in portNumber, and the port is given the name specified by portName. If uniqueName is specified as FALSE, different connection ports are allowed to have identical names. This means that a system can have several server tasks, all being accessible through the same server (connection) port name. Otherwise, if uniqueName is TRUE, only this port is allowed to have the name specified by portName. When the port has been named as portName, XROUT sets a counter (the free connection counter) associated with that port to the value specified in maxConnections. The number of connections that this port can handle, may later be increased or decreased using XMPINFC.

If another port opened, and named using XMPOPNM, already has the specified port name (portName), an error status is returned in returnStatus. The same error is returned if another port has been created as a connection port using XMPOPCN with uniqueName set to TRUE.

When somebody contacts portName by sending a letter via XROUT, XROUT looks at the free connection counter and if it is greater than zero, XROUT decrements it and forwards the letter. If there are no free connections, XROUT tries to find another port with the same name. See also the description under the routine XMPBLET.

The maximum port name length accepted by the routine is defined by the symbol XMMAXNameLength in the XMP:DEFS file. If portName is longer than XMMAXNameLength in bytes, -1 will be returned as error code in returnStatus. If the name length exceed another limit, which is set at XMSG generation time, the port name will be truncated by XMSG, i.e., excess characters are discarded.

Note that since this routine has to reserve and send a message to XROUT to name the port, the routine will change the task's current definition of 'task current' message. Note that the opened port becomes the task's default port. When this port is closed, the previously opened port, if any, becomes the task's default port.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : In this example we create a server port named 'xx-server', and allows another server port to have the same name.

```
% Specify the port name
  'xx-server' =: portName
% Other ports should also be able to use this name
  FALSE =: uniqueName
% Specify maximum no of connections
  3 =: maxConnections
% Create the port as a connection port
  xmpopcn(0,portName,uniqueName,maxConnections,&
          portNumber) =: returnStatus
% Check returnStatus, and if Ok, 'xx-server' has
% been created as a connection port with port
% number = portNumber.
```

Routine name : XMPOPNM			Type: XROUT Service
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	portName Bytes	R	Name of the port.
3	portNumber Integer	W	Port number.

FUNCTION. . : Opens and names a port.

EXPLANATION : A port is opened and given the name specified by portName. The port number is returned in portNumber.

If another open port already has the specified name (portName), an error status is returned in returnStatus.

The maximum port name length accepted by the routine is defined by the symbol XMMAXNameLength in the XMP:DEFS file. If portName is longer than XMMAXNameLength in bytes, -1 will be returned as error code in returnStatus. If the name length exceed another limit, which is set at XMSG generation time, the port name will be truncated by XMSG, i.e., the excess characters are discarded.

Note that since this routine has to reserve and send a message to XROUT to name the port, the routine will change the task's current definition of 'task current' message. Note that the opened port becomes the task's default port. When this port is closed, the previously opened port, if any, becomes the task's default port.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example opens a port with name 'torunn'.

```
% No options permitted, so
  0 =: flags
% Name the port
  'torunn' =: portName
% Let xmpopnm do the job
  xmpopnm(flags,portName,portNumber) =: returnStatus
% Check returnStatus, and if Ok, the port
% number is returned in portNumber.
```

Type: XMSG Function

Routine name : XMPREAD

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	msgDisp Integer	R	Displacement within message in bytes.
4	userAddress Xmuseraddress	R	Address of user buffer.
5	userDisp Integer	R	Displacement within userAddress in bytes.
6	userLength Integer	R	Number of bytes you want to read.
7	readLength Integer	W	Number of bytes actually read.

FUNCTION. . : Reads user data from a specified message buffer.

EXPLANATION : The data will be read from the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message, then the user data will be read as described under routine XMPFREA. If msgIdentifier is -1, the currently defined 'task current' message is assumed. readLength is returned to indicate the actual number of bytes transferred.

OPTIONS . . : As for XMPFREA.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpread(flags,msgIdentifier,msgDisp,&
        addr(inbuffer(0)) force Xmuseraddress,&
        userDisp,userLength,readLength)&
        = : returnStatus
```

Routine name : XMPROUT			Type: XROUT Service
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	localPort Integer	R	Number of the sending port.

FUNCTION. . : Sends a message to, or via, the local routing task (XROUT).

EXPLANATION : The message specified by msgIdentifier is set as 'task current' message and as 'port current' message for localPort. Then the message is sent to the local routing task (XROUT). If msgIdentifier is -1, the current (default) message is assumed instead, namely the 'port current' message for localPort if one exists, or, if none, the 'task current' message.

Note that the message contents should be parameters to XROUT. However, if the message contains a letter service request (see XMPBLET) which is sent via XROUT, the remainder of the message can contain data for the (remote) receiving (server) task. When a message is sent to another task via XROUT, it is forwarded to the (remote) receiving port as a secure message (i.e., the message is forwarded as if it had been sent with the XFSEC flag set, see the description of XFSEC under the routine XMPFSND).

Note that the routine returns to the caller as soon as the message has been sent to (or via) XROUT, which means that it does not wait for (or receives) any reply from XROUT. This must be done explicitly by the caller.

OPTIONS . . : Not available, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : This example creates a letter, writes the letter into an XMSG buffer, fills in data for the receiving task, and sends the message via XROUT using XMPROUT to a remote (server) port.

```
% The message will be sent to a (server) port
% named 'zz-port' in the system named 'gokk'.
  'zz-port' =: portName
  'gokk' =: systemName
% Let xmpblet create the letter in our local buffer.
  xmpblet(myBuffer,60,offSet,123,&
          systemName,portName) =: returnStatus
% Check returnStatus, and if Ok, let's
% reserve an XMSG buffer of 200 bytes.
  xmpfget(0,200,msgIdent) =: returnStatus
% Check returnStatus, and if Ok, copy the letter
% created by xmpblet into the XMSG buffer.
  offSet =: uLength
  xmpfwri(0,0,myBuffer,0,uLength,wLength)&
          =: returnStatus
% Check returnStatus, and if Ok, write data for
% the receiving server task into the XMSG buffer.
  xmpfwri(0,wLength,serverData,0,50,offSet)&
          =: returnStatus
% Check returnStatus, and if Ok, open a port
% so that we can send the message.
  xmpfopn(0,myPort) =: returnStatus
% Check returnStatus, and if Ok, send the message
% from myPort via XROUT to the remote port.
  xmprout(0,msgIdentifier,myPort) =: returnStatus
% Check returnStatus, and if Ok, the message
% has been sent.
```

Routine name : XMPSEND			Type: XMSG Function
No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	localPort Integer	R	Number of the sending port.
4	remoteMagicNum Integer4	R	Magic number of the receiving port.

FUNCTION. . : Sends specified message to another task.

EXPLANATION : The message specified by msgIdentifier is set as 'task current' message and as 'port current' message for localPort. Then the message will be sent as described under routine XMPFSND. If msgIdentifier is -1, the current (default) message is assumed instead (i.e., in this case the routine will act exactly as XMPFSND).

OPTIONS . . : As for XMPFSND.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
2**XFSEC + 2**XFHIP =: flags
xmpsend(flags,msgIdentifier,localPort,&
    remoteMagicNum) =: returnStatus
```

Type: XMSG Function

Routine name : XMPWRHD

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	bytes0To1 Integer2	R	Bytes 0 and 1 of the message header.
4	bytes2To3 Integer2	R	Bytes 2 and 3 of the message header.
5	bytes4To5 Integer2	R	Bytes 4 and 5 of the message header.

FUNCTION. . : Writes to the header of the specified message buffer.

EXPLANATION : The data will be written into the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message, then the user data will be written as described under routine XMPFWHD. If msgIdentifier is -1, the currently defined 'task current' message is assumed.

OPTIONS . . : As for XMPFWHD.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpwrhd(0,msgIdentifier,bytes0To1,bytes2To3,&
        bytes4To5) =: returnStatus
```

Type: XMSG Function

Routine name : XMPWRTE

No:	Parameter Name/ Type:	R/W	Explanation:
1	flags Integer	R	Options.
2	msgIdentifier Xmmsgidentifier	R	Message identifier.
3	msgDisp Integer	R	Displacement within message in bytes.
4	userAddress Xmuseraddress	R	Address of the user buffer.
5	userDisp Integer	R	Displacement within userAddress in bytes.
6	userLength Integer	R	Number of bytes you want to write.
7	writtenLength Integer	W	Number of bytes actually written.

FUNCTION. . : Writes user data into the specified message buffer.

EXPLANATION : The data will be written into the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message. Then the user data will be written as described under routine XMPFWRI. If msgIdentifier is -1, the currently defined 'task current' message is assumed. writtenLength is returned to indicate the actual number of bytes transferred.

OPTIONS . . : As for XMPFWRI.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
xmpwrte(flags,msgIdentifier,msgDisp,&
        addr(inbuffer(0)) force Xmuseraddress,&
        userDisp,userLength,writtenLength)&
        = : returnStatus
```

CHAPTER 3

XMSG/FORTRAN REFERENCE GUIDE

3 XMSG/FORTRAN REFERENCE GUIDE

Function name : XMFBADB			Type: Buffer Formatting
No:	Parameter Name/ Type:	I/O	Explanation:
1	outBuffer Integer*2	I	Local user buffer.
2	offset Integer	IO	Current number of bytes in outBuffer.
3	paramValue Integer*4	I	32 bit value to be coded.
4	paramNumber Integer	I	Parameter number.

FUNCTION. . : Appends a 32-bit value as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append paramValue as the next integer parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and update the offSet parameter accordingly.

paramValue will be put into the buffer as an integer*4 parameter, or, if (and only if) this is valid, as an integer*2 parameter.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

Continued on next page.

RULES . . . : The user buffer must have been initialized using XMFBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats a user buffer containing only one 32-bit value coded as an integer parameter.

```
C First, we initialize the user buffer
  returnStatus = xmfbini(myBuff,lengthBuffer,offSet)
C Check returnStatus, and if Ok,
C append a 32-bit value as parameter 1
  returnStatus = xmfbadb(myBuff,offSet,magicNumb,1)
C Check returnStatus, and if Ok, put in
C the serial number and the service number
  returnStatus = xmfbrdy(myBuff,XSGNM,serialnumber)
```

Function name : XMFBAIN			Type: Buffer Formatting
No:	Parameter Name/ Type:	I/O	Explanation:
1	outBuffer Integer*2	I	Local user buffer.
2	offset Integer	IO	Current number of bytes in outBuffer.
3	paramValue Integer*2	I	16 bit value to be coded.
4	paramNumber Integer	I	Parameter number.

FUNCTION. . : Appends a 16-bit value as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append paramValue as the next integer parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and it will update the offSet parameter accordingly.

paramValue will be put into the buffer as an integer*2 parameter.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMFBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats a user buffer containing only one 16-bit value coded as an integer parameter.

```
C First, we initialize the user buffer
  returnStatus = xmfbini(myBuff,lengthBuffer,offSet)
C Check returnStatus, and if Ok,
C append a 16-bit value as parameter 1
  returnStatus = xmfbain(myBuff,offSet,noOfServic,1)
C Check returnStatus, and if Ok, put in
C the serial number and the service number
  returnStatus = xmfbrdy(myBuff,XSNSP,serialnumber)
```

Type: Buffer Formatting

Function name : XMFBAST

No:	Parameter Name/ Type:	I/O	Explanation:
1	outBuffer Integer*2	I	Local user buffer which needs formatting.
2	offset Integer	IO	Current number of bytes in the buffer.
3	string Character	I	String to be appended.
4	paramNumber Integer	I	Parameter number.

FUNCTION. . : Appends a string as the next parameter in the buffer. The parameter is coded according to the XROUT message format described in appendix B.

EXPLANATION : This call will append the specified string as the next string parameter, with parameter number equal to paramNumber, in the user buffer specified by outBuffer and it will update the offSet parameter accordingly.

Note that outBuffer must start on an even byte boundary and that the call parameter offSet must be equal to the current number of bytes in outBuffer. If one of these checks fails or if outBuffer is too small to contain the parameter, -1 will be returned as error code in returnStatus.

RULES . . . : The user buffer must have been initialized using XMFBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats a user buffer containing only one string parameter.

```
C First, we initialize the user buffer
  returnStatus = xmfbini(myBuff,lengthBuffer,offSet)
C Check returnStatus, and if Ok,
C append the string as parameter 1
  returnStatus = xmfbast(myBuffer,offset,
                         portName(1:-1),1)
C Check returnStatus, and if Ok, put in
C the serial number and the service number
  returnStatus = xmfbrdy(myBuff,XSGIN,serialnumber)
```

Function name : XMFBINI	Type: Buffer Formatting

No:	Parameter Name/ Type:	I/O	Explanation:
1	outBuffer Integer*2	I	Local user buffer.
2	lengthBuffer Integer	I	Total length of the buffer in bytes.
3	offSet Integer	O	Number of bytes used in outBuffer after initializing.

FUNCTION. . : Initializes the user buffer. The buffer is initialized according to the XROUT message format described in appendix B.

EXPLANATION : When a task sends a service request to XROUT, the request (and the response from XROUT) must be coded according to the XROUT message format.

This function will build and initialize the XROUT header in the user buffer specified by outBuffer (and lengthBuffer) for repeated use of the other buffer formatting functions. On return from the function, the offSet parameter will contain the size in bytes used for the XROUT header descriptor, i.e., the space left in outBuffer for coding of parameters using XMFBADB, XMFBAIN and XMFBAST is equal to lengthBuffer-offSet. Thus make sure that the buffer length is big enough to contain the parameter(s).

Note that outBuffer must start on an even byte boundary and that lengthBuffer must be big enough to contain the XROUT header. If one of these checks fails, -1 will be returned as error code in returnStatus.

Continued on next page.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example below formats a buffer containing two parameters.

```
C Specify the total buffer length in bytes.
  lengthBuffer = 100
C Initialize the user buffer
  returnStatus = xmfbini(myBuff,lengthBuffer,offSet)
C offSet no of bytes are used for the XROUT header.
C Check returnStatus, and if Ok, append parameters.
  returnStatus = xmfbast(myBuff,offSet,
                         systName(1:-1),1)
C Check returnStatus, and if Ok, we have used offSet
C no of bytes for the XROUT header and parameter 1.
  returnStatus = xmfbain(myBuff,offSet,systNumb,2)
C Check returnStatus, and if Ok, we have used offSet
C no of bytes for the XROUT header and the two
C parameters (i.e., no of bytes not yet used in
C outBuffer equals 100-offSet).
```

Type: Buffer Formatting

Function name : XMFBLET

No:	Parameter Name/ Type:	I/O	Explanation:
1	headerBuffer Integer*2	I	Local user buffer.
2	lengthBuffer Integer	I	Total length of the buffer in bytes.
3	offSet Integer	O	Number of bytes used in headerBuffer after formatting.
4	serialNumber Integer	I	Reference number.
5	systemName Character	I	Name of destination system.
6	portName Character	I	Name of remote port.

FUNCTION. . : Formats and codes a header for the XROUT 'Send Letter' service (XSLET). The letter is created according to the XROUT message format described in appendix B.

EXPLANATION : The function will create the XROUT header that is required when a task wants to send a letter (service XSLET) to XROUT.

The header will be created and formatted in the user buffer specified by headerBuffer. serialNumber is put into byte 0 of the letter to allow the user task, which may have more than one request outstanding at the same time, to distinguish this letter from other messages. The port name specified by portName is appended as parameter 1, and the system name specified by systemName is appended as parameter 2 in the header.

systemName is the name of the system the letter will be sent to, whereas portName is the name of a (remote server) port in systemName that you want to contact. If the length of the name specified by systemName is 0, the local system is assumed.

Note that this function just prepares the letter in a local buffer (headerBuffer). It does not copy the header into an XMSG message buffer, nor does it send anything. The data copying and sending must be done using other functions, i.e., XMFFWRI and XMFROUT.

The XMSG system provides task to task communication within the same system and between tasks running in different systems. When a task wants to send a message to another task, the sending task must know the magic number of a port belonging to the receiving task. Since

the magic number of a port (and the port number) is allocated by XMSG when a task opens a port, the sending task must obtain the magic number of the remote (receiving) port via XROUT (routing task).

When a task (usually a server task) has opened and named a port (e.g., using XMFOPCN or XMFOPNM), another task can now send a message from one of its own ports, via XROUT, to the named (remote) port. The header of the message sent via XROUT must contain a 'Send Letter' (XSLET) service request to XROUT. The remainder of the message can contain user data for the receiving (server) task (e.g., protocol information, user name, password etc.). The remainder of the message will not be looked at by XROUT (i.e., XROUT will only look at the header of the message - the letter), thus the user data can be (coded) in any format legible to the receiving (server) task.

When XROUT receives the letter, XROUT will look at systemName, and if systemName has been defined as a (remote) system name, XROUT will forward the letter (message) to the XROUT in the specified system.

The destination XROUT will look up the specified portName in its name table. If portName is a normal named port (i.e., a port named using XMFOPNM), then XROUT will forward the whole message to portName. If portName is a connection port (i.e., a port named using XMFOPCN), XROUT will look at the free connection counter for portName and if this is greater than zero, XROUT will decrement the counter and forward the whole message to portName. If there are no free connections, XROUT tries to find another port with the same name (portName) that have a free connection, and if found, XROUT will decrement the free connection counter and forward the message to that port.

When the (server) task receives the message on portName, it can, and it normally will, check that the sending task is allowed to use the server before it sends a (positive) reply to the requester, and thereby gives away its own magic number. If the server task does not want to give away its own magic number, it can do so by sending a (negative) reply with the XFFWD (forward message) option.

When the requesting task receives the (positive) reply from the (server) task, it can then use the XMFFMST function to extract the magic number of the remote (server) port (portName), and direct communication with the remote (server) port can begin.

Note that if XMSG is unable to send the letter to XROUT in the specified system, or if the destination XROUT does not know the name of the destination port (i.e., if portName does not match any named port in systemName),

or if portName is a connection port with no free connections, then XROUT will return the letter (message) to the sending port with an error status. The requesting (sending) task can test whether the remote (server) task returned a reply or XROUT returned the message by checking the message type (which is returned as a result from a call to a 'receive message' or a call to a 'port status' function).

Note that headerBuffer must start on an even byte boundary and that lengthBuffer must be big enough to contain the formatted XROUT 'Send Letter' service header. If one of these checks fails, -1 will be returned as error code in returnStatus.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example formats the header for an XROUT letter in our local buffer.

```
C We are going to send to a port named 's-port' in the
C system named 'scholar'.
  portName = 's-port'
  systemName = 'scholar'
C To recognize the message, we put in a reference no.
  serialNumber = 111
C Give xmfblet more than enough space for formatting.
  lengthBuffer = 60
C Let xmfblet create the necessary header.
  returnStatus = xmfblet(myBuffer,lengthBuffer,offSet,
       serialNumber,systemName(1:-1),portName (1:-1))
C Check returnStatus, and if Ok, then offSet
C number of bytes have been used for the XROUT
C header. (In this example, offSet=22.)
C (If 'scholar' is the name of our own system, we
C can of course create the letter specifying a
C system name length of zero mstead)
C  xmfblet(headerBuffer,lengthBuffer,offSet,
C       serialNumber,systemName(0: -1),portNamel:-1))
C We can now copy the service header from our
C local buffer into the header of a message, fill
C the remainder of the message with data for the
C receiving task, and send the message via XROUT.
C (If we need to send data to the receiving task, we
C must make sure that these data are written into the
C message after the XROUT header.)
```

Function name : XMFBLOC			Type: Buffer Formatting
No:	Parameter Name/ Type:	I/O	Explanation:
1	lowBuffer Integer*2	I	Local user buffer.
2	paraNumber Integer	I	Parameter number of the parameter that is to be found.
3	startOfParam Integer	O	Displacement within localBuffer in bytes.
4	paramType Character	O	Indicates the parameter type of paramNumber. The parameter type is returned as INTEGER or STRING.
5	paramLength Integer	O	Length of parameter in bytes.

FUNCTION. . : Locates a parameter within a buffer coded according to the XROUT message format described in appendix B.

EXPLANATION : Since the parameters in localBuffer may have been put into the buffer in a random order, this function can be used to locate a specified parameter.

The function will search through localBuffer for the parameter number specified by paramnumber. If the parameter is found, paramType indicates the parameter type of the located parameter, startOfParam indicates the position of the first significant byte in the parameter, and paramLength gives number of significant bytes in the parameter. If no parameter with the requested parameter number is found, -1 is returned as error code in returnStatus.

Note that localBuffer must start on an even byte boundary. If not, -1 will be returned as error code in returnStatus.

Continued on next page.

RULES . . . : The user buffer must have been coded according to the XROUT message format.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example locates parameter number 1 (which has been coded as a string parameter containing only the two characters H and I) in a user buffer containing:

```
Byte 0   = 123  Serial number
     1   = 0    Status from XROUT (0=Ok)
     2-3 = 32   Length of remainder of mess in bytes
     4   = -1   Param. no. 1(negative means string)
     5   = 2    Length of parameter 1 in bytes
     6   = 72   First byte of parameter 1
     7   = 73   Second byte of parameter 1
     8-9 = 14   Length of parameter 2 in bytes
     etc.

C Specify and locate parameter number
  paramNumber = 1
  returnStatus = xmfbloc(localBuffer,paramNumber,
                         startOfParam,paramType,paramLength)
C Check returnStatus, and if Ok, startOfParam=6,
C paramType=STRING and paramlength=2.
```

Type: Buffer Formatting

Function name : XMFBRDY

No:	Parameter Name/ Type:	I/O	Explanation:
1	outBuffer Integer*2	R	Local user buffer.
2	serviceNumber Integer	R	XROUT service number.
3	serialNumber Integer	R	Reference number.

FUNCTION. . : Puts the serial number and the service number into a buffer which has been coded according to the XROUT message format described in appendix B.

EXPLANATION : When all the necessary parameters have been appended to outBuffer, this function can be used to insert the serial number and the service number in the buffer. serialNumber will be put into byte 0 and serviceNumber will be put into byte 1 of the buffer.

Note that since this function overwrites the first two bytes of the XROUT header descriptor with the serial number and the service number, this should be the last buffer preparation function called, before the user buffer is written into a message and sent to XROUT. Note that this function does not copy the local user buffer into any message; this must be done using a function such as XMFFWRI.

Note that outBuffer must start on an even byte boundary. If not, -1 will be returned as error code in returnStatus.

Continued on next page.

RULES . . . : The user buffer must have been initialized using XMFBINI.
Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example build a buffer for the 'Send Letter' (XSLET) service request. It is similar to the behaviour of function XMFBLET.

```
C Initialize the user buffer
  returnStatus = xmfbini(myBuffer,lengthBuffer,offSet)
C Check returnStatus, and if Ok, append
C the port name as parameter 1
  returnStatus = xmfbast(myBuffer,offSet,
                         portName(1:-1),1)
C Check returnStatus, and if Ok, append
C the system name as parameter 2
  returnStatus = xmfbast(myBuffer,offSet,
                         systemName(1:-1),2)
C Check returnStatus, and if Ok, we will insert
C the serial number and the service number XSLET
  returnStatus = xmfbrdy(myBuffer,XSLET,serialNumber)
C Check returnStatus, and if Ok, we can now write
C the buffer into a message and send the request
C to XROUT.
```

Function name : XMFCLNM			Type: XROUT Service
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portName Character	I	Name of the port.
3	portNumber Integer	I	Number of the port.

FUNCTION. . : Closes a port and clears the port name.

EXPLANATION : This function can be used to close a named port that has been opened and named using XMFOPNM or XMFOPCN. The port specified by portNumber will be closed and the name assigned to portNumber will be cleared (i.e., the name will be removed from XROUT's name table).

When portNumber is closed, all nonsecure messages currently queued for that port are released, while all secure messages (as well as the 'port current' message, if any) are set nonsecure and returned to the sender.

The specified portNumber should be a port number returned from a call to XMFOPNM or XMFOPCN, and the specified portName should be the port name declared when the port was opened and named.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : This simple example will close a port and clear the name of a previously opened port.

```
C Specify the port's name and number
  portName = 'xx-server'
  portNumber = 4
  returnStatus = xmfclnm(0,portName(1:-1),portNumber)
```

Type: XMSG Function

Function name : XMFCONF

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	XMSGpassword Integer	O	XMSG password (the same as XMSG version code).
3	configMask Integer	O	Configuration mask. See below.
4	XMSGrestartCnt Integer	O	XMSG restart count.

FUNCTION. . : Gets information about the running XMSG system.

EXPLANATION : On return, XMSGpassword contains the password which is needed in order to become a privileged XMSG task (see function XMFFPRV). XMSGrestartCnt returns the number of times XMSG has been (re)started since the last warmstart.

The bits currently defined in configMask are:

```
bit 0 : set if inter-system XMSG
    1 :  "  "  generated with tracing
    2 :  "  "  generated for ND-100
    3 :  "  "  file server for file transfer is incl.
    4 : is not used
    5 : set if running on page table 3
    6 :  "  "  generated for ND-100/CX instruction set
    7 :  "  "  generated with gateway software for
               network servers
```

Note that this bit mask, which is based on XMSG version J, will most certainly be extended in later XMSG versions.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmfconf(0,XMSGpassword,configMask,
                       XMSGrestartCnt)
```

Type: XMSG Function

Function name : XMFFABR

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	userBuffer Integer*2	I	User buffer.
3	userDisp Integer	I	Address of the user buffer in bytes. See note below.
4	readLength Integer	I	Number of bytes to read.
5	memoryDisp Integer	I	Address within bankNumber.
6	bankNumber Integer	I	The seven least significant bits (0-6) specifies bank number.

FUNCTION. . : Is a absolute reading from the part of the physical memory used by XMSG.

EXPLANATION : This call allows a task to read a block of data from the physical memory into its user area specified by userBuffer (and userDisp).

If bankNumber is zero, a value for bankNumber equal to the bank in which the XMSG kernel code has been fixed is assumed. The data is read from the specified bank, starting from the address specified by memoryDisp, into the user buffer.

Note that on an ND-100, the userDisp (displacement within the user buffer) is always rounded down to the previous even byte, if an odd displacement is specified.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged users.
Not permitted for drivers.
Not available for tasks running in an ND-500.

EXAMPLE . . :
```
returnStatus = xmffabr(0,userBuffer,userDisp,readLength,
                       memoryDisp,bankNumber)
```

Function name : XMFFALM			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	messageSize Integer	I	Message size in bytes.
3	numberOfMsgs Integer	I	Number of messages to allocate.

FUNCTION. . : Allocates message buffers to a task.

EXPLANATION : Normal message buffers that have been reserved using the XMFFGET call, lose their association with the task that got them when they are sent to another task. This implies that the task has no guarantee that it will be able to get space later.

By allocating message buffers, a task can indicate to XMSG its long-term buffer requirements. Allocated messages are removed from the free space pool, and marked as allocated to the original caller. They do change owners when sent within a system, but when released, or sent out of the local system, the message buffer is put back on the original allocator task's "Available Allocated Message List (AAML)".

All allocated messages for a given task must be of the same size. When an XMFFGET is executed by that task for a buffer of that size, XMSG will first look at the task's AAML and take a message buffer from it, if one is available. Similarly, when a message of that size comes into the system from another system, XMSG will first look at the AAML for the receiving task and take a message buffer from it, if one is available.

Note that if the function fails, due to lack of buffer space, no messages are allocated.

Continued on next page.

OPTIONS . . : XFEXC - Exclusive buffers. If set, the message buffers are allocated and set aside for exclusive use by the task, i.e., these message buffers will not be used by XMSG when a message of messageSize is received from another system. Buffers allocated with XFEXC do not have to be of the same size as buffers allocated without this option set. However, all exclusive buffers must be of the same size. To reserve one of these exclusive buffers, the task must call the XMFFGET function with the XFEXC flag set.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffalm(0,messageSize,numberOfMsgs)
```

Function name : XMFFCLS			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portNumber Integer	I	Number of port to be closed.

FUNCTION. . : Close(s) port(s).

EXPLANATION : Closes the specified local port. If portNumber is less than zero, all ports owned by the calling task will be closed. If portNumber is zero, the most recently opened port (i.e., the default port) will be closed.

When a port is closed, all nonsecure messages currently queued for that port are released, while all secure messages (as well as the 'port current' message, if any) are set nonsecure and returned to the sender. If the port had a name, the name is cleared (i.e., the name is removed from XROUT's name table).

See also the disconnect call, XMFFDCT.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : This will close all ports owned by this task:

```
portNumber = -1
returnStatus = xmffcls(0,portNumber)
```

Type: XMSG Function

Function name : XMFFCPV

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	The identifier of a received message.
3	accessInfo Integer	O	Access information. See below.
4	additionalInfo Integer	O	Additional information. See below.

FUNCTION. . : Checks system and user privileges.

EXPLANATION : This call allows a task, when a message has been received, to check the privileges of the sender.

If the sending task is allowed to update the routing tables (i.e., execute the privileged XROUT services XSDRN and XSDSY) on this system, then accessInfo = 1. If the message is sent from a task within the local system, then additionalInfo = 0. If the message is sent from a task in another system, then additionalInfo = 1.

If the sending task is not allowed to update the routing tables, then accessInfo = 0 and D contains the reason:

additionalInfo = 0 implies that the sending task, as well as the source system are nonprivileged.
additionalInfo = 1 implies that the source system is privileged, but the sending task is not.
additionalInfo = 2 implies that the sending task is privileged, but the source system is not.
additionalInfo = 3 if the specified message is returned (it could not be delivered).

An nonprivileged task is a task which has not (yet) successfully executed the XMFFPRV call. A nonprivileged system is a remote system which has not (yet) been defined as a friend to your system, see the XROUT service XSDAT.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffcpv(0,msgIdentifier,accessInfo,
               additionalInfo)
```

Type: XMSG Function

Function name : XMFFCRD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	interruptLevel Integer	I	The interrupt level that the driver should run on.
3	registerBlock Integer*2	I	Register block (an 8-word buffer).
4	XTblockAddress Integer	O	Address of the XT-block allocated to the driver.

FUNCTION. . : Defines a driver for XMSG.

EXPLANATION : This call is used to define an already existing driver, with a context as defined in the register block. The buffer must contain the register block that the driver will be started with, in the order required for the Load Register Block (LRB) hardware instruction. XMSG will allocate a task block (XT-block) to the driver and return its address in XTblockAddress.

OPTIONS . . : XFPON - Paging on. This must be set if the driver is running with paging on.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in an ND-500.

EXAMPLE . . :
```
flags = 2**XFPON
returnStatus = xmffcrd(flags,interruptLevel,
                       registerBlock,XTblockAddress)
```

Type: XMSG Function

Function name : XMFFDBK

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	bankNo Integer	I	Bank number.

FUNCTION. . : Defines a bank number for drivers.

EXPLANATION : When calling functions which transfer data between a user area and an XMSG buffer (e.g., XMFFREA, XMFFWRI or XMFFSMC), drivers specify a physical address of the user buffer (the inBuffer parameter). This is in bank 0, unless they have previously defined a bank number using the XMFFDBK call.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.
Not permitted for RT programs.
Not available for tasks running in ND-500.

EXAMPLE . . : returnStatus = xmffdbk(0,bankNo)

Function name : XMFFDCT			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.

FUNCTION. . : Disconnects from XMSG.

EXPLANATION : Releases all XMSG resources. All ports opened by the task are closed and all XMSG space belonging to the current caller is released. Special action is taken in the case of current messages, and messages waiting on the input queue (see XMFFSND, XMFFRCV and XMFFCLS).

Note that the following automatic disconnects are executed by SINTRAN:

User disconnect:
- On return to the background command processor
- On log out or RT program termination

System mode disconnect:
- On log out or RT program termination

There is no return from a driver call to XMFFDCT (as the driver context is released by the call).

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : returnStatus = xmffdct(0)

Function name : XMFFDMM			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	requestedTaskSp Integer	I	Requested task space in bytes.

FUNCTION. . : Defines the maximum limit of memory usage.

EXPLANATION : When a new task is defined in XMSG, its maximum buffer space is set to a predefined value (defined when the XMSG system is generated). This can be changed for privileged tasks by using this call. requestedTaskSp will be set equal to the maximum number of bytes of message space that can be owned by the task at one time.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.

EXAMPLE . . : returnStatus = xmffdmm(0,requestedTaskSp)

Function name : XMFFDUB			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	bufferAddress Integer*4	I	Address of physical memory buffer.
3	bufferLength Integer	I	Number of bytes in the buffer.

FUNCTION. . : Defines a user buffer.

EXPLANATION : This is a privileged call that allows a task to associate a physical memory buffer with a message descriptor previously obtained by XMFFGET with sizeBuffer = 0. All XMSG calls then operate on that message, as the buffer space was part of the general XMSG buffer pool, except that XMFFREL only releases the message descriptor and not the buffer area.

This allows special systems or drivers to fully control their memory allocation procedures.

This call acts on the 'task current' message.

Buffers that have been defined in this way cannot be sent to other systems.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not available for tasks running in an ND-500.

EXAMPLE . . : returnStatus = xmffdub(0,bufferAddress,bufferLength)

Function name : XMFFDUM			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.

FUNCTION. . : Dummy call.

EXPLANATION : This call may be useful if the programmer wants to check that XMSG is up and running. It is also useful for benchmarking.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for all users.

EXAMPLE . . : returnStatus = xmffdum(0)

Function name : XMFFFRM			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	noOfMsgToFree Integer	I	Number of allocated message buffers to free.
3	noOfMsgFreed Integer	O	Number of message buffers freed.

FUNCTION. . : Frees allocated message buffers.

EXPLANATION : This call frees message buffers which have been allocated by XMFFALM.

OPTIONS . . : XFEXC - Exclusive buffers. If set, only those message buffers which have been allocated with the XFEXC option set will be freed. If not set, only those message buffers which have been allocated without the XFEXC option will be freed.

RULES . . . : Permitted for all users.

EXAMPLE . . :
```
flags = 2**XFEXC
returnStatus = xmfffrm(flags,noOfMsgToFree,
        noOfMsgFreed)
```

Function name : XMFFGET			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	sizeBuffer Integer	I	Number of bytes requested.
3	msgIdent Integer	O	Message identifier.

FUNCTION. . : Reserves a message buffer from XMSG's buffer pool.

EXPLANATION : msgIdent is returned to the caller for possible use in subsequent functions. Each buffer size has a maximum, system dependent size defined when the XMSG system is generated. The total XMSG buffer space owned by a task cannot exceed another limit, which is initially set to a value defined at XMSG generation time. It may be changed, however, by privileged tasks using the Define Maximum Memory (XMFFDMM) function.

Only the current owner of a message is allowed to read or write in it, give it to someone else, or release it.

Specifying a buffer size of 0 bytes implies that only a message descriptor will be reserved. Privileged tasks can then associate a physical memory area with that message descriptor using the XMFFDUB function. It is not allowed to send a buffer of size 0 to another system.

OPTIONS . . : XFWTF - Wait flag. If no message buffer of the requested size is available, the task will be suspended. Execution resumes when a buffer becomes available.

XFEXC - Exclusive buffer. If set, it implies that the caller wants to reserve exclusively a message buffer allocated using the XMFFALM function. If no such allocated message buffer is available, an error status is returned.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
sizeBuffer = 20
returnStatus = xmffget(0,sizeBuffer,msgIdent)
```

Function name : XMFFGST			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	startScanPort Integer	I	Last port to be scanned.
3	portNumber Integer	O	Port number where the message is waiting.

FUNCTION. . : A task may have many open ports. It does not always know on which one the next message is arriving. XMFFGST allows the programmer to check all ports belonging to the task, i.e., it allows the task to find out whether any messages have been received on any port.

EXPLANATION : The call parameter startScanPort specifies the last port to be searched. If startScanPort is zero, this implies the most recently opened port (i.e., the default port). Note that the search will begin with the next port (if any) after that specified, and then follow the task's port list (see example below).

On return, the parameter portNumber contains the port number where the message, if any, is waiting. If no message is waiting and the XFWTF flag is not used, the base value of the error codes (XMXENTM) is returned as a status.

For example, if the task has opened four ports and have got the port numbers 15 (from the 1st XMFFOPN), 4 (from the 2nd XMFFOPN), 6 (from the 3rd XMFFOPN) and 19 (from the 4th XMFFOPN), then the port list comprises the ports 19-6-4-15 (in that order!). Port number 19 (the first port in the list) is the task's default port (i.e., if startScanPort is zero, this port is assumed). If the task has just handled a message received on port 6, it can, when it wants to have a 'round-robin' scheduling of requests, call XMFFGST with startScanPort=6. Port 6 will then be the last port to be looked at by XMSG. XMSG will start looking at port 4 to see if a message is waiting. If no message is waiting on port 4, XMSG will look at port 15. If no message is waiting on port 15, XMSG will look at port 19, and, if no message is waiting on port 19, XMSG will finally look at port 6.

Note that calling this function, when a message is waiting on one of the ports, will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on the returned portNumber. If no message

is waiting on any of the ports, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on all ports opened by the task before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMFFGST call, and the 'wake up' bit on the individual ports may have been set as a result from previously executed XMFFPST, XMFFRCV, XMFFRRH or XMFFRRE calls.)

OPTIONS . . : XFWTF - Wait flag. If set, the task is suspended if no messages are waiting. Execution resumes when a message arrives on one of the ports.

XFWAK - General wake up. Unless a message is already waiting on one of the ports, a 'general wake up' bit will be set for this task. When 'general wake up' is set, the next transmission to any of the ports opened by the task will lead to a wake up of the receiver task, and clearing of the 'general wake up' bit for that task.

However, be aware that if the task is in XMSG wait position (for example, if sending a secure message with wait), when the task should have been woken up as a result of a message being sent to one of its ports, the 'general wake up' bit will be cleared, but the task will not (and cannot) be woken up.

XFHIP - High priority message. Allows a task to check the arrival of high priority messages. If a high priority message is waiting on one of the ports and XFHIP is set, the port number where the high priority message is waiting is returned in portNumber. If no high priority message is waiting on any of the ports and XFHIP is set, and XFWTF is not set, the base value of the error codes (XMXENTM) is returned as returnStatus. When the next message of any type is sent to a port opened by the task, the task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFWTF
startScanPort = 4
returnStatus = xmffgst(flags,startScanPort,portNumber)
```

Type: XMSG Function

Function name : XMFFLMP

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier or 0.
3	portNo Integer	I	Port number or 0.
4	msgIdentFound Integer	O	Message ID for the first message found equal to or greater than requested.
5	msgSize Integer	O	Message size in bytes.
6	portNoFound Integer	O	Port no. of the first port equal to or greater than requested.

FUNCTION. . : Lists messages and ports.

EXPLANATION : This call allows a task to obtain information about its own open ports and its own messages.

msgSize contains the number of bytes obtained when the message was reserved using the Get Message Buffer (XMFFGET) call, or allocated using the Allocate Message Buffers (XMFFALM) call.

If there is no message found equal to or greater than that requested, then msgIdentFound is 0.

If there is no port found equal to or greater than that requested, then portNoFound is 0.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmfflmp(0,msgIdentifier,portNo,
                       msgIdentFound,msgSize,portNoFound)
```

Type: XMSG Function

Function name : XMFFM2P

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	magicNumber Integer*4	I	Magic number.
3	portNumber Integer	O	Port number.
4	systemNumber Integer	O	System number.
5	RTinOrOtherInfo Integer	O	RT index or other information. See below.
6	additionalInfo Integer	O	Additional information. See below.

FUNCTION. . : Converts magic number to a port and a system number.

EXPLANATION : This call allows you to convert the magic number to a port number and a system number.

RTinOrOtherInfo may contain additional information about the port owner task. If the magic number was that of a system or that of a remote port, then RTinOrOtherInfo = -1. If the magic number was that of a local port and the port owner task is a driver, then RTinOrOtherInfo = -2. If the magic number was that of a local port and the port owner task is an RT-program, then RTinOrOtherInfo = RT-index of the RT-program.

additionalInfo may contain additional information about the specified magic number. If the magic number was that of a system, then additionalInfo = 3. If the magic number was that of a local port and the port owner task is privileged, then additionalInfo = 2. If the magic number was that of a remote port, or that of a local port and the port owner task is nonprivileged, then additionalInfo = 1.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffm2p(0,magicNumber,portNumber,
               systemNumber,RTinOrOtherInfo,additionalInfo)
```

Type: XMSG Function

Function name : XMFFMST

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	msgType Integer	O	Type of message, see explanation below.
4	remoteMagicNum Integer*4	O	Magic number of port that sent the message.
5	msgLength Integer	O	Message length in bytes.

FUNCTION. . : Obtains message status.

EXPLANATION : This call allows a task to extract the sender's magic number, and get the length and type of a received message.

Message type:

XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent secure, but could not be delivered)

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffmst(0,msgIdentifier,msgType,
                       remoteMagicNum,msgLength)
```

Function name : XMFFOPN			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portnumber Integer	O	Port number.

FUNCTION. . : Opens a port.

EXPLANATION : A port is opened and the port number (i.e., the port identifier) is returned in portNumber.

The opened port becomes the task's default port. When this port is later closed, the previously opened port, if any, becomes the task's default port.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : returnStatus = xmffopn(0,portnumber)

Function name : XMFFP2M			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portNumber Integer	I	Port number.
3	magicNumber Integer*4	O	Magic number.

FUNCTION. . : Converts port number to magic number.

EXPLANATION : This function allows a task to convert a local port number to a magic number. Any task may obtain the magic number of its own ports. Privileged tasks can obtain the magic number of a port owned by another local task.

Note that this function will only return the magic number of ports owned by tasks in the local system.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : `returnStatus = xmffp2m(0,portNumber,magicNumber)`

Function name : XMFFPRV			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	xmsgPassword Integer	I	XMSG password.

FUNCTION. . : Makes the calling task privileged.

EXPLANATION : Some of the functions can only be executed by privileged XMSG tasks. In order to become privileged (for XMSG), a task must successfully execute the XMFFPRV call. When you want the task no longer to be privileged, the same call should be used, but with xmsgPassword equal to zero. The reason for specifying the XMSG password is to ensure that privileged programs, that base themselves on accessing XMSG table structures, have been updated to the current XMSG table definitions.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : The caller must be either a driver, a direct task, a foreground program, or a background program logged in as user system. Besides this, the program must also specify the current XMSG password, which can be obtained using the XMFFCONF function.

EXAMPLE . . : returnStatus = xmffprv(0,xmsgPassword)

Function name : XMFFPST			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	localPort Integer	I	Port number to be checked.
3	msgType Integer	O	Message type, see explanation below.
4	remotePort Integer	O	Hashed magic number of remote port.
5	msgIdentifier Integer	O	Message identifier.
6	queueLength Integer	O	Number of messages queued for localPort.

FUNCTION. . : Checks a port to see if any message is waiting.

EXPLANATION : If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

On return from the function, msgType indicates the message type of the first message queued to localPort. If no message is waiting, msgType is zero. If a message is waiting, the remotePort, msgIdentifier and queueLength parameters will contain the hashed magic number of the sending port, the message address and the number of messages chained to the localPort.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

Note that calling this function when a message is waiting on localPort will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on localPort. If no message is waiting on localPort, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on localPort before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMFFGST call, and the 'wake up' bit on localPort may have been set as a result from a previously executed XMFFPST, XMFFRCV, XMFFRRH or XMFFRRE call.)

Continued on next page.

Message type:
XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent secure but, could not be delivered)

OPTIONS . . : XFWTF - Wait flag. If no message is waiting on localPort, the task is suspended. Execution resumes when a message arrives on localPort.

XFWAK - Wake up. Unless a message is already waiting on localPort, a 'wake up' bit will be set on localPort. When 'wake up' is set on localPort, the next transmission to this port will lead to a wake up of the receiver task, and clearing of the 'wake up' bit on localPort. This option can be enabled on more than one port at a time.

However, be aware that if the task is in XMSG wait position (for example, sending a secure message with wait), when the task should have been woken up as a result of a message being sent to localPort, the 'wake up' bit will be cleared but the task will not (and cannot) be woken up.

XFHIP - High priority message. Allows a task to check the arrival of high priority messages. If a high priority message is waiting and XFHIP is set, the message type XMTHI is returned in msgType. If no high priority message is waiting, and XFHIP is set and XFWTF is not set, a zero (0) is returned in msgType. When the next message of any type is sent to localPort, the receiving task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffpst(flags,localPort,msgType,
                       remotePort,msgIdentifier,queueLength)
```

Type: XMSG Function

Function name : XMFFRCV

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	localPort Integer	I	Number of receiving port.
3	msgType Integer	O	Message type, see explanation below.
4	remotePort Integer	O	Hashed magic number of remote port.
5	msgIdent Integer	O	Message identifier.
6	msgLengthOrStat Integer	O	Message length in bytes. If msgType is XMTRE, msgLengthorStat contains the error status.

FUNCTION. . : Receives a message when it is queued for a port.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue) and its address returned in msgIdentifier. msgType indicates the message type of the received message, msgLengthorStat gives the message length and remotePort contains the hashed magic number of the sending port. If the message type is XMTRE (returned message), then msgLengthorStat contains the reason for return.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

A successful receiving causes the received message to become the 'task current' message. In addition, if it is a secure message (i.e., a message sent with option XFSEC set), it becomes the 'port current' message for localPort. If the task aborts or localPort is closed while the message is 'port current', the message will be returned to the sender with return status.

The current task message is cleared by releasing/sending it to someone else, or receiving another message. The current port message is cleared by releasing/sending it to someone else or receiving another secure message. A task may also change the value of the current message using the XMFFSCM function.

If no message is waiting on localPort and the XFWTF flag is not used, the base value of the error codes (XMXENTM) is returned as returnStatus.

Note that calling this function when a message is waiting on localPort will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on localPort. If no message is waiting on localPort, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on localPort before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XMFFGST function, and the 'wake up' bit on localPort may have been set as a result from a previously executed XMFFPST, XMFFRCV, XMFFRRH or XMFFRRE function.)

Message type:

XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority (sent with XFHIP option)

XMTRE - Returned message (sent secure but could not be delivered)

OPTIONS . . . : XFWTF - Wait flag. If no message is waiting on localPort, the task is suspended. Execution resumes when a message arrives on localPort.

XFWAK - Wake up. Unless a message is already waiting on localPort, a 'wake up' bit will be set on localPort. When 'wake up' is set on localPort, the next transmission to this port will lead to a wake up of the receiver task, and clearing of the 'wake up' bit on localPort.

When the wake up is done, the message is not received, and so the receiving must be repeated. This option can be enabled on more than one port at a time.

However, be aware that if the task is in XMSG wait position (for example, sending a secure message with wait), when the task should have been woken up as a result of a message being sent to localPort, the 'wake up' bit will be cleared, but the task will not (and cannot) be woken up.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFWTF
returnStatus = xmffrcv(flags,localPort,msgType,
                       remotePort,msgIdent,msgLengthOrStat)
```

Type: XMSG Function

Function name : XMFFREA

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgDisp Integer	I	Displacementwithin messagein bytes.
3	inbuffer Integer*2	I	User buffer.
4	userDisp Integer	I	Displacement within inBuffer in bytes. See note below.
5	userLength Integer	I	Number of bytes you want to read.
6	readLength Integer	O	Number of bytes actually read.

FUNCTION. . : Reads user data from a message buffer.

EXPLANATION : The data is read from the 'task current' message, starting with displacement msgDisp (rounded up to the next even byte), into the user buffer specified by inBuffer (and userDisp). readLength is returned to indicate the actual number of bytes read. If msgDisp is -1, the reading of the message is resumed from the current message displacement.

On return, if the last byte in the message is read, the current message displacement is set to 0, and the 'whole-message-read' flag is set, so that the next 'write message' function (e.g.,XMFFWRI or XMFFWHD) will reset the current message length to zero. Otherwise, except when readLength is zero, the current message displacement is set to msgDisp+readLength, where msgDisp is the specified displacement (rounded up if necessary) and readLength is the actual number of bytes transferred. If readLength is zero, the current message displacement is not updated.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffrea(0,msgDisp,inbuffer,userDisp,
                       userLength,readLength)
```

Function name : XMFFREL			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.

FUNCTION. . : Releases message buffer.

EXPLANATION : This call is used to release a message buffer reserved by the task. A message buffer is reserved by the task when the task issues the XMFFGET call, and when a message is sent to it from another task. In the latter case the message must be received before it can be released.

At any particular time, the total message buffer space owned by a task cannot exceed a limit defined when the XMSG system is generated. Therefore,as a general rule for a task, its message buffer should be released as soon as the task is through with it.

A msgIdentifier parameter of -1, will release the 'task current' message.

If the specified message is an allocated message (i.e., a message allocated using the XMFFALM call), the message will be put back on the original task's 'Available Allocated Message List' (AAML), see the function XMFFALM.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : returnStatus = xmffrel(0,msgIdentifier)

Type: XMSG Function

Function name : XMFFRHD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	bytes0To1 Integer*2	O	Bytes 0 and 1 of message header.
4	bytes2To3 Integer*2	O	Bytes 2 and 3 of message header.
5	bytes4To5 Integer*2	O	Bytes 4 and 5 of message header.

FUNCTION. . : Reads only the header of a message buffer.

EXPLANATION : The first 6 bytes of a message buffer are read and returned in bytes0To1, bytes2To3 and bytes4To5, and then the current message displacement is set to 6.

If msgIdentifier is not -1, the specified message becomes the 'task current' message.

If the message size is less than 6 bytes, an error return occurs.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffrhd(0,msgIdentifier,bytes0To1,
                       bytes2To3,bytes4To5)
```

Type: XMSG Function

Function name : XMFFRRE

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	localPort Integer	I	Number of the receiving port.
3	inBuffer Integer*2	I	User buffer.
4	userDisp Integer	I	Displacement within inBuffer in bytes. See note below.
5	userLength Integer	I	Number of bytes you want to read.
6	msgType Integer	O	Message type, see explanation below.
7	remotePort Integer	O	Hashed magic number of the remote port.
8	msgIdent Integer	O	Message identifier.
9	msgLengthOrStat Integer	O	Message length in bytes. If msgType is XMTRE, msgLengthOrStat contains the error status.

FUNCTION. . : Receives a message queued on/for a port and reads from the message buffer.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue) and then userLength number of bytes will be read from the first byte in the message buffer into the user buffer specified by inBuffer (and userDisp). If the last byte in the message is read, the current message displacement is set to 0, and the 'whole-message-read' flag is set. Thus the next 'write message' call will reset the current message length to zero. Otherwise, if the last byte is not read, the current message displacement is set to the actual number of bytes read.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed. If userLength is greater than the message length, only message length number of bytes will be read into the user buffer.

Note that on an ND-100, the userDisp (displacement within the user buffer) is always rounded down to the previous even byte.

Note that when the message is received, both the 'task current' message and the 'port current' message will be set as described under function XMFFRCV. Note also that the handling of flags (options) in this function is identical to the handling described under XMFFRCV. Also the return parameters from the function are identical to the return parameters from the XMFFRCV call.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

You should note that this function act as if both XMFFRCV and XMFTREA had been called. Calling this function, instead of the other two functions, eliminates the overhead associated with each function and XMSG call.

Message type: As for XMFFRCV.

OPTIONS . . : As for XMFFRCV.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFWTF
returnStatus = xmffrre(flags,localPort,userBuffer,
     userDisp,userLength,msgType,remotePort,msgIdent,
     msgLengthOrStatus) =: returnStatus
```

Type: XMSG Function

Function name : XMFFRRH

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	localPort Integer	I	Number of the receiving port.
3	msgType Integer	O	Message type, see explanation below.
4	remotePort Integer	O	Hashed magic number of the remote port.
5	msgIdentifier Integer	O	Message identifier.
6	bytesOTolorStat Integer*2	O	Normally first 2 bytes of message. If msgType is XMTRE, bytesOTolorStat contains the error status.

FUNCTION. . : Receives and reads the header.

EXPLANATION : If a message is waiting on localPort, it will be received (unchained from the message queue). Then the first two bytes of the message buffer are read and returned in the bytesOTolorStat parameter. If the length or size of the received message is less than two bytes, two random bytes will be returned in bytesOTolorStat.

If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

Note that when the message is received, both the 'task current' message and the 'port current' message will be set as described under function XMFFRCV. Note also that the handling of flags (options) in this function is identical to the handling described under XMFFRCV. The return parameters from the function are identical to the return parameters from the XMFFRCV call, except that the first two bytes of user data is returned instead of the message length.

If no message is waiting on localPort and the XFWTF flag is not set, the base value of the error codes (XMXENTM) is returned as returnStatus.

Message type: As for XMFFRCV.

Continued on next page.

OPTIONS . . : As for XMFFRCV.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFWTF
returnStatus = xmffrrh(flags,localPort,msgType,
                       remotePort,msgIdentifier,bytesOTolorStatus)
```

Function name : XMFFRTN			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	localPort Integer	I	Number of the sending port.
4	data0 Integer*2	I	First 2 bytes of the message header.

FUNCTION. . : Returnes a message to the port from which it came.

EXPLANATION : The user often needs to write a return status into a message and send it back to the port from which it came (e.g., replying to a transaction). This call leads to msgIdentifier being set as the 'task current' message and the 'port current' message for localPort, data0 being written into the first two bytes of the message buffer. Then the message is being returned to the port from which it was last sent.

The localPort parameter specifies the port from which the message will be sent. If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

OPTIONS . . : XFWTF - Wait flag. This is only significant when sending a secure (XFSEC) message to a task in another system.

If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port).

If not set, secure messages that cannot be delivered will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered, or if the receiving port is closed (e.g., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFHIP - High priority message. The message will be chained to the head of the receiver's queue instead of the tail, following any other high priority messages already queued.

XFFWD - Forwarding message. The sender information in the message will not be updated. To the receiver, it will appear that the message was sent directly from the previous sending port.

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., the functions XMFFRCV, XMFFRRH or XMFFRRE), which would have led to this message being received, it will instead be returned to the sender.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : `returnStatus = xmffrtn(0,msgIdentifier,localPort,data0)`

Function name : XMFFSCM			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portNo Integer	I	Port number.
3	msgIdentifier Integer	I	Message identifier.

FUNCTION. . : Sets the current message.

EXPLANATION : Since many functions implicitly operate on the current message, it is useful to be able to set the latter. This call sets the specified message as the 'task current' message. If portNo is >=0, the message is also set as 'port current' for the specified port. If portNo is zero, the most recently opened port (i.e., the default port) is assumed.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : returnStatus = xmffscm(0,portNo,msgIdentifier)

Function name : XMFFSIN			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	XMSGbase Integer	O	XMSG base field address.

FUNCTION. . : Gets XMSG's base field address.

EXPLANATION : This call returns the base field address of the message system in the memory bank, where the XMSG kernel code has been fixed. This address is needed in order to be able to access XMSG tables.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged users.
Not permitted for drivers.
Not available for tasks running in an ND-500.

EXAMPLE . . : `returnStatus = xmffsin(0,XMSGbase)`

Type: XMSG Function

Function name : XMFFSMC

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	noOfCalls Integer	I	Number of XMSG functions to be executed.
3	userBuffer Integer*2	I	Buffer containing the parameters.
4	userDisp Integer	I	Displacement within the userBuffer, in bytes, see note below.
5	Treg Integer	O	The content of the T-register.
6	Areg Integer	O	The content of the A-register.
7	Dreg Integer	O	The content of the D-register.
8	Xreg Integer	O	The content of the X-register.

FUNCTION. . : Starts multi call.

EXPLANATION : This call allows a task to execute a set of XMSG functions issuing only one function call. This eliminates the overhead associated with each function call (and XMSG monitor call).

noOfCalls is the number of XMSG functions to be executed and userBuffer is the buffer containing the parameters for the functions. Each set of parameters comprise 4 words (T, A, D and X registers), so the buffer length should be 8*noOfCalls bytes long. noOfCalls has a maximum, system dependent size defined when the XMSG system is generated. If noOfCalls is 0 (or -1), then the previously executed multi call request will be re-executed.

Note that on an ND-100, the userDisp (displacement within the user buffer) is always rounded down to the previous even byte.

The meaning of the T, A, D and X registers depend on the particular XMSG function. A documentation of the XMSG functions is provided in appendix A.

XMFFSMC returns as soon as an XMSG function terminates with status less than or equal to zero (or when all the functions have been executed). The return parameters (Treg, Areg, Dreg and Xreg) are set according to the return registers from the last XMSG function executed.

Completion status is also returned in the returnStatus, XMOK, if the multi call has been successfully executed,in XMXENTM, if one of the XMSG functions in the multi call was not terminated, otherwise returnStatus contains an error code.

You should be aware of the fact that if an XMSG disconnect function is specified (and executed) as one of the functions in the multi call, the succeeding functions in the multi call will not be executed, as the task context (XT-block) is released by the disconnect (XFDCT) function.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks. Not available for tasks running in an ND-500.

EXAMPLE . . :
```
returnStatus = xmffsmc(0,noOfcalls,userBuffer,
                       userDisp,Treg,Areg,Dreg,Xreg)
```

Type: XMSG Function

Function name : XMFFSND

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	localPort Integer	I	Number of the sending port.
3	remoteMagicNum Integer*4	I	Magic number of the receiving port.

FUNCTION. . : Sends the current message to another task.

EXPLANATION : When a task wants to send a message to another task, it must know the magic number of a port of the other task. A description of how to obtain the magic number is given in the sample programs and under the XMFBLET call. A remoteMagicNum parameter of -1 will direct the message back to the port from which it was last sent.

The localPort parameter specifies the port from which the message will be sent. If localPort is zero, the most recently opened port (i.e., the default port) is assumed.

Note that there is no parameter specifying the message that is to be sent, for the reason that the current (default) message buffer is assumed, namely the 'port current' message if one exists, or, if none, the 'task current' message.

OPTIONS . . : XFWTF - Wait flag. This is only significant when sending a secure (XFSEC) message to a task in another system.

If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port).

If not set, secure messages that cannot be delivered will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered, or if the receiving port is closed (e.g., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFFWD - Forwarding message. The sender information in the message will not be updated. To the receiver, it will appear that the message was sent directly from the previous sending port.

XFROU - Route message. Ignore the remoteMagicNum parameter and send the message to the local routing task (XROUT). The message contents should be parameters to XROUT. (See appendix B on XROUT services.)

XFRRO - Remote route message. If the XFROU flag is also set, then send the message to a remote routing task (XROUT). The 16 most significant bits of remoteMagicNum is assumed to contain the system number to which the message will be sent. The message contents should be parameters to XROUT.

Note that if the XFROU flag is not set and XFRRO is set, the message will be sent as if XFHIP had been set (i.e., when XFROU is not set, setting the XFRRO flag will act as if the XFHIP flag had been set instead).

XFHIP - High priority message. If the XFROU flag is not set, the message will be chained to the head of the receiver's queue, instead of the tail, following any other high priority messages already queued.

Note that if both the XFROU flag and XFHIP are set, the message will be sent as if XFROU and XFRRO had been set.

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., the functions XMFFRCV, XMFFRRH or XMFFRRE), which would have led to this message being received, it will instead be returned to the sender.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFSEC + 2**XFHIP
returnStatus = xmffsnd(flags,localPort,remoteMagicNum)
```

Type: XMSG Function

Function name : XMFFSTD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	XTblockAddress Integer	I	The address of the XT-block belonging to the driver.

FUNCTION. . : Starts driver.

EXPLANATION : This call starts the execution of a driver which has already been defined by the XMFFCRD call.

XTblockAddress must contain the driver's task block address as returned from the XMFFCRD call. XMFFSTD overwrites the driver's L register with the XTblocAddress before starting the driver.

In this way a started driver will have the L register containing its XT-block address. The driver must make sure that the L register still contains the XT-block address before calling XMSG.

XMFFSTD does not set the appropriate bit in the PIE register. Nor does it load or fix any segments. This should be done using the FIXC and ENTSG monitor calls.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Only permitted for privileged tasks.
Not permitted for drivers.
Not available for tasks running in ND-500.

EXAMPLE . . : returnStatus = xmfstd(0,XTblockAddress)

Function name : XMFFWDF			Type: XMSG Function
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	Bregister Integer	I	The B-register of the driver on restart.
3	restartAddress Integer	I	Restart address for the driver.

FUNCTION. . : Defines wake-up context.

EXPLANATION : If a driver uses the XFWAK (wake up) option, XMSG must be told where to restart the driver. This is done by using the XMFFWDF call. When the driver is restarted by XMSG, it will be restarted in the address specified by restartAddress with its B register set to the address specified by Bregister.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.
Not permitted for RT-programs.
Not available for tasks running in an ND-500.

EXAMPLE . . : returnStatus = xmffwdf(0,Bregister,restartAddress)

Type: XMSG Function

Function name : XMFFWHD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	bytes0To1 Integer*2	I	Bytes 0 and 1 of the message header.
3	bytes2To3 Integer*2	I	Bytes 2 and 3 of the message header.
4	bytes4To5 Integer*2	I	Bytes 4 and 5 of the message header.

FUNCTION. . : Writes to the header of the 'task current' message buffer.

EXPLANATION : If the 'whole-message-read' flag has been set (see XMFFREA), it is cleared and the current message length (not the same as size) is set to 0. Then the function inserts bytes0To1, bytes2To3 and bytes4To5 as the first six bytes of the message. If this results in the message being longer than before, the current message length is set to 6. It then sets the current message displacement to 6.

If the message size is less than 6 bytes, an error return occurs.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : returnStatus = xmffwhd(0,bytes0To1,bytes2To3,bytes4To5)

Type: XMSG Function

Function name : XMFFWRI

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgDisp Integer	I	Message displacement (within XMSG buffer), in bytes.
3	outbuffer Integer*2	I	User buffer.
4	userDisp Integer	I	Displacement in the user buffer, in bytes. This must be an even number.
5	userLength Integer	I	Number of bytes you want to write.
6	writtenLength Integer	O	Number of bytes actually written.

FUNCTION. . : Writes user data into a message buffer.

EXPLANATION : After building up a data buffer in its own space, a task transfers the data buffer into the 'task current' message buffer using XMFFWRI. If the 'whole-message-read' flag has been set (see XMFFREA), it is cleared and the current message length (not the same as size) is set to 0. If msgDisp is -1, a value for msgDisp equal to the current message displacement is assumed instead, thus providing an appending function. If msgDisp is odd, 1 is added to it, and a zero bytes inserted in the message.

If msgDisp+userLength is greater than the message size, an error return occurs. Otherwise, userLength bytes are copied from the user buffer into the message buffer, and the current message displacement is set to msgDisp+writtenLength (where msgDisp has been rounded up if odd). If this copying resulted in the message being longer than before, the current message length is also set to msgDisp+writtenLength. writtenLength is returned to indicate the actual number of bytes transferred.

Note that the displacement within the message is always rounded up to the next even byte and, on an ND-100, that userDisp (displacement within the user buffer) is always rounded down to the previous even byte before the data is written.

Continued on next page.

OPTIONS . . : XFRES - Reset current message length. If set, it leads to the current message length being set to 0 before the user data is transferred into the message buffer. (In fact it acts as if the 'whole-message-read' flag had been set.)

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmffwri(flags,msgDisp,outbuffer,userDisp,
                       userLength,writtenLength)
```

Type: XROUT Service

Function name : XMFINFC

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portNumber Integer	I	Port number returned by XMFOPCN.
3	extraConn Integer	I	Number of extra connections.
4	serialNumber Integer	I	Reference number.

FUNCTION. . : Increments (or decrements) the free connection count.

EXPLANATION : After opening a connection port using XMFOPCN, a task can later increment (when connections become available) or decrement (when number of connections need to be reduced) the free connection counter associated with that port.

If extraConn is positive, the maximum number of connections that portNumber can handle is increased. If extraConn is negative, the maximum number of connections that portNumber can handle will be decreased. If the resulting number of free connections becomes negative, an error status will be returned from XROUT.

Note that this function will not wait for a reply from XROUT, and so the caller will later receive this reply from XROUT on the port specified by portNumber. serialNumber is put into byte 0 of the request sent to XROUT to allow the caller, who may have many requests outstanding at the same time, to recognize the reply.

Note that since this function has to reserve and send a message to XROUT to increase (or decrease) the number of connections accepted, the function will change the task's current definition of 'task current' message, as well as the current definition of 'port current' message on portNumber, if any.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : When a server port with port number 12, which has previously been created using XMFOPCN, is able to handle a new connection, we should inform XROUT.

```
C We are able to handle one more connection
  extraConn = 1
C on the port number returned from xmfopcn
  portNumber = 12
C To recognize the reply, we need a reference no.
  serialNumber = 100
C Tell XROUT
  returnStatus = xmfinfc(0,portNumber,extraConn,
                 serialNumber)
```

Type: XROUT Service

Function name : XMFOPCN

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portName Character	I	Name of the port.
3	uniqueName Logical	I	Uniqueness flag.
4	maxConnections Integer	I	Maximum number of connections accepted.
5	portNumber Integer	O	Port number.

FUNCTION. . : Creates a connection port.

EXPLANATION : This call is very similar to XMFOPNM, but allows XROUT to control the number of connections that a port can handle simultaneously, and even distribute connections among server (connection) ports.

As for XMFOPNM, a port is opened and its port number is returned in portNumber, and the port is given the name specified by portName. If uniqueName is specified as FALSE, different connection ports are allowed to have identical names. This means that a system can have several server tasks, all being accessible through the same server (connection) port name. Otherwise, if uniqueName is TRUE, only this port is allowed to have the name specified by portName. When the port has been named as portName, XROUT sets a counter (the free connection counter) associated with that port to the value specified in maxConnections. The number of connections that this port can handle, may later be increased or decreased using XMFINFC.

If another port, opened and named using XMFOPNM, already has the specified port name (portName), an error status is returned in returnStatus. The same error is returned if another port is created as a connection port using XMFOPCN with uniqueName set to TRUE.

When somebody contacts portName by sending a letter via XROUT, XROUT looks at the free connection counter and if it is greater than zero, XROUT decrements it and forwards the letter. If there are no free connections, XROUT tries to find another port with the same name. See also the description under the function XMFBLET.

The maximum port name length accepted by the function is defined by the symbol XMMAXNameLength in the XMP:DEFS file. If portName is longer than XMMAXNameLength in bytes, -1 will be returned as error code in returnStatus. If the name length exceed another limit, which is set at XMSG generation time, the port name will be truncated by XMSG, i.e., excess characters are discarded.

Note that since this function has to reserve and send a message to XROUT to name the port, the function will change the task's current definition of 'task current' message. Note that the opened port becomes the task's default port. When this port is closed, the previously opened port, if any, becomes the task's default port.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : In this example we create a server port named 'xx-server', and allows another server port to have the same name.

```
C Specify the port name
  'xx-server' =: portName
C Other ports should also be able to use this name
  uniqueName = FALSE
C Specify maximum no of connections
  maxConnections = 3
C Create the port as a connection port
  returnStatus = xmfopen(0,portName(1:-1),uniqueName,
                 maxConnections,portNumber)
C Check returnStatus, and if Ok, 'xx-server' has
C been created as a connection port with port
```

Function name : XMFOPNM			Type: XROUT Service
No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	portName Character	I	Name of the port.
3	portNumber Integer	O	Port number.

FUNCTION. . : Opens and names a port.

EXPLANATION : A port is opened and given the name specified by portName. The port number is returned in portNumber.

If another open port already has the specified name (portName), an error status is returned in returnStatus.

The maximum port name length accepted by the function is defined by the symbol XMMAXNameLength in the XMP:DEFS file. If portName is longer than XMMAXNameLength in bytes, -1 will be returned as error code in returnStatus. If the name length exceed another limit, which is set at XMSG generation time, the port name will be truncated by XMSG, i.e., the excess characters are discarded.

Note that since this function has to reserve and send a message to XROUT to name the port, the function will change the task's current definition of 'task current' message. Note that the opened port becomes the task's default port. When this port is closed, the previously opened port, if any, becomes the task's default port.

Continued on next page.

OPTIONS . . : Not implemented, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . : The example opens a port with name 'torunn'.

```
C No options permitted, so
  flags =0
C Name the port
  portName = 'torunn'
C Let xmfopnm do the job
  returnStatus = xmfopnm(flags,
                         portName(1:-1),portNumber)
C Check returnStatus, and if Ok, the port
C number is returned in portNumber.
```

Type: XMSG Function

Function name : XMFREAD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	msgDisp Integer	I	Message displacement (within XMSG buffer), in bytes.
4	inbuffer Integer*2	I	User buffer.
5	userDisp Integer	I	Displacement in user buffer, in bytes.
6	userLength Integer	I	Number of bytes you want to read.
7	readLength Integer	O	Number of bytes actually read.

FUNCTION. . : Reads user data from a specified message buffer.

EXPLANATION : The data will be read from the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message, then the user data will be read as described under function XMFFREA. If msgIdentifier is -1, the currently defined 'task current' message is assumed. readLength is returned to indicate the actual number of bytes transferred.

OPTIONS . . : As for XMFFREA.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmfread(flags,msgIdentifier,msgDisp,
                       inbuffer,userDisp,userLength,readLength)
```

Type: XROUT Service

Function name : XMFROUT

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	localPort Integer	I	Number of the sending port.

FUNCTION. . : Sends a message to, or via, the local routing task (XROUT).

EXPLANATION : The message specified by msgIdentifier is set as 'task current' message and as 'port current' message for localPort. Then the message is sent to the local routing task (XROUT). If msgIdentifier is -1, the current (default) message is assumed instead, namely the 'port current' message for localPort if one exists, or, if none, the 'task current' message.

Note that the message contents should be parameters to XROUT. However, if the message contains a letter service request (see XMFBLET) which is sent via XROUT, the remainder of the message can contain data for the (remote) receiving (server) task. When a message is sent to another task via XROUT, it is forwarded to the (remote) receiving port as a secure message (i.e., the message is forwarded as if it had been sent with the XFSEC flag set, see the description of XFSEC under the function XMFFSND).

Note that the function returns to the caller as soon as the message has been sent to (or via) XROUT, which means that it does not wait for (or receives) any reply from XROUT. This must be done explicitly by the caller.

OPTIONS . . : Not available, flags should be zero.

RULES . . . : Permitted for both non-privileged and privileged tasks.

Continued on next page.

EXAMPLE . . : This example creates a letter, writes the letter into an XMSG buffer, fills in data for the receiving task, and sends the message via XROUT using XMFROUT to a remote (server) port.

```
 C The message will be sent to a (server) port
 C named 'zz-port' in the system named 'gokk'.
   portName = 'zz-port'
   systemName = 'gokk'
 C Let xmfblet create the letter in our local buffer.
   returnStatus = xmfblet(myBuffer,60,offSet,123,
                  systemName(1:-1),portName(1:-1))
 C Check returnStatus, and if Ok, let's
 C reserve an XMSG buffer of 200 bytes.
   returnStatus = xmffget(0,200,msgIdent)
 C Check returnStatus, and if Ok, copy the letter
 C created by xmfblet into the XMSG buffer.
   uLength = offSet(1:-1)
   returnStatus =
xmffwri(0,0,myBuffer,0,uLength,wLength)
 C Check returnStatus, and if Ok, write data for
 C the receiving server task into the XMSG buffer.
   returnStatus =
xmffwri(0,wLength,serverData,0,50,offSet)
 C Check returnStatus, and if Ok, open a port
 C so that we can send the message.
   returnStatus = xmffopn(0,myPort)
 C Check returnStatus, and if Ok, send the message
 C from myPort via XROUT to the remote port.
   returnStatus = xmfrout(0,msgIdentifier,myPort)
 C Check returnStatus, and if Ok, the message
 C has been sent.
```

Type: XMSG Function

Function name : XMFSEND

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	localPort Integer	I	Number of sending port.
4	remoteMagicNum Integer*4	I	Magic number of receiving port.

FUNCTION. . : Sends specified message to another task.

EXPLANATION : The message specified by msgIdentifier is set as 'task current' message and as 'port current' message for localPort, then the message will be sent as described under function XMFFSND. If msgIdentifier is -1, the current (default) message is assumed instead (i.e., in this case the function will act exactly as XMFFSND).

OPTIONS . . : As for XMFFSND.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
flags = 2**XFSEC + 2**XFHIP
returnStatus = xmfsend(flags,msgIdentifier,
       localPort,remoteMagicNum)
```

Type: XMSG Function

Function name : XMFWRHD

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	bytes0To1 Integer*2	I	Bytes 0 and 1 of the message header.
4	bytes2To3 Integer*2	I	Bytes 2 and 3 of the message header.
5	bytes4To5 Integer*2	I	Bytes 4 and 5 of the message header.

FUNCTION. . : Writes to the header of the specified message buffer.

EXPLANATION : The data will be written into the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message. Then the user data will be written as described under function XMFFWHD. If msgIdentifier is -1, the currently defined 'task current' message is assumed.

OPTIONS . . : As for XMFFWHD.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmfwrhd(0,msgIdentifier,bytes0To1,
                       bytes2To3,bytes4To5)
```

Type: XMSG Function

Function name : XMFWRTE

No:	Parameter Name/ Type:	I/O	Explanation:
1	flags Integer	I	Options.
2	msgIdentifier Integer	I	Message identifier.
3	msgDisp Integer	I	Message displacement (within XMSG buffer), in bytes.
4	inbuffer Integer*2	I	User buffer.
5	userDisp Integer	I	Displacement in the user buffer, in bytes.
6	userLength Integer	I	Number of bytes you want to write.
7	writtenLength Integer	O	Number of bytes actually written.

FUNCTION. . : Writes user data into the specified message buffer.

EXPLANATION : The data will be written into the message buffer specified by msgIdentifier. msgIdentifier will first be set as 'task current' message, then the user data will be written as described under function XMFFWRI. If msgIdentifier is -1, the currently defined 'task current' message is assumed. writtenLength is returned to indicate the actual number of bytes transferred.

OPTIONS . . : As for XMFFWRI.

RULES . . . : Permitted for both non-privileged and privileged tasks.

EXAMPLE . . :
```
returnStatus = xmfwrte(flags,msgIdentifier,msgDisp,
                       inbuffer,userDisp,userLength,writtenLength)
```

CHAPTER 4

INTRODUCTION TO RR-LIB

4 INTRODUCTION TO RR-LIB

4.1 Introduction

RR-LIB is a set of library routines, interfacing with XMSG. It is based on a request-response mode of interaction between a client program and a server program (or between multiple clients and servers). If your communication is structured as a series of request-response interactions, then RR-LIB is an efficient tool to use.

Since RR-LIB uses XMSG for communication, the communicating parties may run on a single system or on any two systems connected in a COSMOS network.

4.2 Concepts Related to Server/Client

A server is a program which gives service to one or more clients. In the data-transfer phase, a client using RR-LIB sends a request, which is a string of bytes, to a server. The client then has to wait for the response, which is another string of bytes from the server, before it may send a new request to that server.

RR-LIB has three types of calls:

- Calls performed by servers. In PLANC these are prefixed by the letters RRPS, where the S stands for Server.
- Calls performed by clients. In PLANC these are prefixed by the letters RRPC, where the C stands for Client.
- Calls performed by both servers and clients. In PLANC these are prefixed by RRPB, where B stands for Both.

The client calls are divided into low-level calls and high-level calls. The high-level calls provide a simpler interface, if the restrictions imposed by their use are acceptable.

With the high-level calls you may simply select a server, and then send a request to that server. You receive the response in the same call as you do the request. Each of these two calls is a combination of low-level client calls. For example, the select call is the combination of connection request, wait, and connection confirmation.

In general, we recommend that you use the high-level client calls if possible.

If you need to have several, simultaneously outstanding requests to different servers, then you need to use the low-level calls. If you, in addition to being a client also are a server, we recommend that you use the low-level client calls. The same program may indeed be both server and client.

Also, the low-level calls should be used, if you cannot afford to wait until the response returns.

4.3 How to Set up a Connection

A connection between a client and a server is created by the following steps:

1) The client sends a connection request. From PLANC the RRPCCNRQ call is used. Then the client goes into a waiting state.

2) The server, going into the waiting state (the RRPBWAIT call in PLANC), receives the connection indication event from RR-LIB.

3) The server processes the connection indication event by performing the connection indication call (RRPSCNIN).

4) The server sends a connection response to the client (RRPSCNRS).

5) RR-LIB signals the connection confirmation event to the client.

6) The client processes the connection confirmation event by performing the connection confirmation call (RRPCCNCF). The connection is now established, and the data transfer can start.

During the connection phase, it is possible to send user data. The server may, for example, require that the client sends an identification with his connection request.

4.4 The Data-Transfer Phase

During the data-transfer phase, the client sends a request (RRPCSNRQ), goes into a waiting state, and later receives the response from the server, signalled by an event. To obtain the response, the client has to perform the 'get response' call (RRPCGTRS).

If the client has set up connections with several servers, it can send several requests (maximum one per connection) before going into a waiting state. Note that each response indication event has to be waited for, because the arrival of responses is signalled one at a time.

4.5 Disconnecting

Either the server or the client may take the initiative of disconnecting. A disconnect request is sent (RRPBDCRQ), and a disconnect indication event is signalled by RR-LIB to the other party, which must then perform the disconnect indication call (RRPBDCIN) before the disconnect is complete.

4.6 Addressing

When the server starts executing, it has to identify itself with a server name (which is equivalent to an XMSG port name) before any clients can connect.

The client addresses the server by specifying the system name where the server resides, plus the server name.

4.7 Events

Certain changes in the communication system, resulting from actions by the remote user or RR-LIB, are signalled to you by the "event" mechanism.

Examples of events are: timeout, response has arrived, and disconnect from remote. Most of these events require processing by calls to RR-LIB.

The only way an event can be signalled to you is through the RRPBWAIT call.

Only one event can be signalled in one RRPBWAIT call.

4.8 General Information about the Routines

The routine implementation in PLANC gives the status as an out value:

```
ROUTINE VOID,INTEGER (parameters.....).
```

Example of a call:

```
RRPCcncf(static,remoteID,serverInfo) =: RRstatus
```

Normal return status is zero. If you include the appropriate :DEFS file in your source code, you may use the symbol OK for the zero status. The file is called RRP:DEFS for PLANC. Other return status are error codes. A list of the error codes plus their corresponding symbols is provided in appendix E.

In the description of the PLANC calls, an R is used to denote a read parameter and W stands for write.

4.9 Table of Events

The symbols for the event codes are defined in the RRP:DEFS file. You may specify a whole set of events by using logical OR to form a bit mask.

Event code	Value	Explanation
RREVtime	1	Timeout
RREVcnin	2	Connection request has arrived
RREVcncf	4	Connection is accepted
RREVrqin	8	Request arrived
RREVrsin	16	Response has arrived
RREVdcin	32	Disconnected from remote
RREVdccf	64	User initiated disconnect is complete
RREVunkn	128	Message arrived on port not known to RR-LIB
RREVothr	256	No RR-LIB event, timeout, or unknown port

The reason for RREVunkn is that you may combine the use of RR-LIB and XMSG in the same program. As you performed the RRPBWAIT call, to wait for an event, a message may arrive on a port opened by an XMSG call. The RREVunkn event tells you that this happened.

The RRPBWAIT routine is at some point performing the 'tmout' monitor call. If your RT-program becomes rescheduled for execution, for example by another program, the RREVothr will occur.

4.10 Table of Server Calls

The calls used by server programs are listed in alphabetical order in this table:

Routine	Purpose	Phase where used
RRPBABRT	Abort connection	Disconnect
RRPBDCIN	Disconnect indication	Disconnect
RRPBDCRQ	Disconnect request	Disconnect
RRPBINIT	Initialize RR-LIB	
RRPBWAIT	Wait for event	Used in all phases
RRPSCNIN	Connection indication	Connection establishment
RRPSCNRS	Connection response	Connection establishment
RRPSEND	Clear up untidy ends	When finished
RRPSGTRQ	Get request	Data transfer
RRPSSNRS	Send response	Data transfer

4.11 Table of Low-Level Client Calls

The calls used by low-level client programs are listed in alphabetical order in this table:

Routine	Purpose	Phase where used
RRPBABRT	Abort connection	Disconnect
RRPBDCIN	Disconnect indication	Disconnect
RRPBDCRQ	Disconnect request	Disconnect
RRPBINIT	Initialize RR-LIB	
RRPBWAIT	Wait for event	Used in all phases
RRPCCNCF	Connection confirmation	Connection establishment
RRPCCNRQ	Connection request	Connection establishment
RRPCEND	Clear up untidy ends	When finished
RRPCGTRS	Get response	Data transfer
RRPCSNRQ	Send request	Data transfer

4.12 Table of High-Level Client Calls

The calls used by high-level client programs are listed in alphabetical order in this table:

Routine	Purpose
RRPBINIT	Initialize RR-LIB
RRPCCALL	Send request and get response
RRPCSLCT	Select a server
RRPCDISC	Disconnect request with wait for completion

CHAPTER 5

RR-LIB/PLANC REFERENCE GUIDE

5 RR-LIB/PLANC REFERENCE GUIDE

5.1 RR-LIB server calls

Routine name : RRPBABRT			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned from RRPScnin.

FUNCTION. . : Aborts connection.

EXPLANATION : This end of a connection is aborted. You should use this call only if a disconnect attempt RRPBderq does not work, i.e., if you do not get a disconnect confirmation after a reasonable time.

USAGE . . . : RRPBabrt(static,remoteID) =: RRstatus

Routine name : RRPBDCIN

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when disconnect was signalled.
3	reason Integer	W	Reason for disconnect.
4	info Bytes pointer	W	Information from the client (if client-initiated disconnect).

FUNCTION. . : Disconnects indication.

EXPLANATION : A disconnect request from the client is perceived by you as a disconnect indication.

This routine should only be called after the disconnect indication event (RREVdcin) is received.

It should not be called immediately if a "disconnect pending" error return is obtained from another call. Otherwise not all of the associated data will have been received.

USAGE . . . : RRPBdcin(static,remoteID,reason,info) =: RRstatus

Routine name : RRPBDCRQ			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned from RRPScnin.
3	info Bytes	R	Information for remote end.

FUNCTION. . : Disconnects request.

EXPLANATION : This call disconnects you from a client. It is used to refuse a connection or break an established connection.

Information may only be sent by the server in response to a connection indication.

USAGE . . . : RRPBdcrq(static,remoteID,info) =: RRstatus

Routine name : RRPBINIT

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	dynamic Integer array	R	Work area for connection control blocks.
3	mode Rrmd	R	Should be specified as RRMDasServer or RRMDasBoth.
4	maxClientConn Integer	R	Dummy for servers.
5	serverName Bytes	R	Name by which you are known to clients.
6	bufferArea Bytes array	R	Buffers for user data. See below.
7	maxServerConn Integer	R	Maximum number of simultaneous connections to clients.
8	serverParam Rrsp	R	Other server parameter. See below.

FUNCTION. . : Initializes RR-LIB data structures.

EXPLANATION : This must be the first call to RR-LIB. You should use standard sizes for the RR-LIB work areas: static and dynamic. These required sizes are given in the RRP:DEFS file.

The number of buffers in bufferArea must be the same as maxServerConn. This means that you must use one buffer per connection, and this buffer is used both for the request and the response. All these buffers must be the same size and begin on a word boundary.

If you wish to increase the maximum XMSG task space for yourself, you must set RRSPisDefault = FALSE and set up the remaining parameters in RRSP.

Continued on next page.

This is the structure of the serverParameter record:

```
TYPE RRSP = RECORD
   BOOLEAN: RRSPisDefault    % TRUE => default
                             % parameters are used
                             % FALSE => must fill in
                             % remainder of record
   INTEGER: RRSPallocation   % requested value of XMSG
                             % task space
   INTEGER: RRSPpassword     % current XMSG password
ENDRECORD
```

RRSPisDefault = TRUE implies that default parameters are used. In that case you do not specify the other parameters in the RRSP record.

The server task space should be increased beyond the usual default value, if the server is likely to handle a significant number of active connections simultaneously.

RRSPisDefault = FALSE implies that you want to extend the XMSG task space for yourself. The remainder of the record has to be filled in.

RRSPallocation is the requested value of the XMSG task space.

RRSPpassword is the current XMSG password.

Changing the allocation requires the current XMSG password (XPASW) and that the server is a foreground program, or a background program logged in as user system. To obtain the XMSG password, you may use the XM-LIB routine XMPCONF.

USAGE . . . : RRPBinit(static,dynamic,mode,maxClientConn,
serverName,maxServerConn,serverParam) =: RRstatus

Routine name : RRPBWAIT			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	timeout Rrtm	R	Maximum waiting time.
3	requestedEvent Rrev	R	Requested event. This includes the connection identifier.
4	actualEvent Rrev	W	The event that actually occurred.

FUNCTION. . : Waits for event to occur.

EXPLANATION : This is the only call in which arriving XMSG messages (responses or acknowledgements for parts of requests) are processed.

The Rrtm type in timeout has the following definition:

```
TYPE RRTM = RECORD
      INTEGER:  RRTMlength
      INTEGER:  RRTMunits
ENDRECORD
```

RRTMlength is the length of the waiting time. RRTMunits is defined as for the SINTRAN HOLD command:

1 = Basic time units
2 = Seconds
3 = Minutes
4 = Hours

A value of 0 for RRTMlength is equivalent to a poll for outstanding RR-LIB events of the requested type, on the requested connection.

The set of desired events is formed from the logical OR of the corresponding event codes. With the exception of RREVunkn and RREVothr, you will only receive events permitted by requestedEvent. Your set of desired events should always include, RREVdcin because a disconnect may occur at any time.

Continued on next page.

A value of RRanyRemote may be used to accept any connection. Likewise, RRanyEvent may be used to accept any event.

Event codes used to form a bit mask:

```
CONSTANT RREVtime =    1    % timeout
CONSTANT RREVcnin =    2    % connection indication
CONSTANT RREVrqin =    8    % request arrived
CONSTANT RREVdcin =   32    % disconnected from remote
CONSTANT RREVdccf =   64    % user disconnect complete
CONSTANT RREVunkn =  128    % XMSG arrived on port
                            % not known to RR-LIB
CONSTANT RREVothr =  256    % exit caused by other than
                            % timeout, an RR-LIB event,
                            % or unknown port
                            % eg., by terminal input
```

Structure of an event:

```
CONSTANT RRMXevData = 2     % Length of the associated
                            % event data.

TYPE RREV = RECORD
      RRID:          RREVremote % ID for remote task
      INTEGER:       RREVevent  % BIT MASK of events
      INTEGER ARRAY: RREVdata(0:RRMXevData-1)
ENDRECORD
```

RREVdata contains data associated with the event. The meaning of these depends on the RREVevent.

If RREVevent = RREVunkn then RREVdata (0) gives the XMSG portnumber. In this case the return from RRPBwait is equivalent to a return from the XMSG general status function XFGST. Note that the arrived message must be received (XFRCV) before RRABwait is called again.

USAGE . . . :
```
RRPBwait(static,timeout,requestedEvent,
         actualEvent) =: RRstatus
```

Routine name : RRPSCNIN

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when the RREVcnin event occurred.
3	clientInfo Bytes pointer	W	Information sent by the client.
4	serverInfoBuff Bytes pointer	W	Buffer where you should place return information to the client.

FUNCTION. . : Connection indication.

EXPLANATION : This call gives you information from the client after an RREVcnin event.

The SIZE of the serverInfoBuff gives you the maximum amount of data that may be returned to the client.

USAGE . . . :
```
RRPScnin(static,remoteID,clientInfo,
         serverInfoBuff) =: RRstatus
```

Routine name : RRPSCNRS			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when the RREVcnin event occurred.
3	serverInfo Bytes	R	Information sent back to the client, placed in the buffer given by RRPScnin.

FUNCTION. . : Connection response.

EXPLANATION : Following the RRPScnin call, RRPScnrs accepts the connection from the client.

USAGE . . . : RRPScnrs(static,remoteID,serverInfo) =: RRstatus

Routine name : RRPSEND			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.

FUNCTION. . : Ends server functions.

EXPLANATION : Used when no more services is to be provided. This call closes XMSG ports, releases XMSG buffers, etc. You should disconnect all active connections first, so the clients are aware that you are gone.

USAGE . . . : RRPSend(static) =: RRstatus

Routine name : RRPSGTRQ

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when the RREVrqin event occurred.
3	request Bytes pointer	W	The request from the client.
4	responseBuffer Bytes pointer	W	Buffer where you should place the response.

FUNCTION. . : Gets request.

EXPLANATION : With this call you obtain a client request after its arrival has been signalled by the RREVrqin event.

The size of the responseBuffer gives you the maximum amount of bytes that may be returned to the client.

USAGE . . . :
```
RRPSgtrq(static,remoteID,request,responseBuffer)&
        =: RRstat
```

Routine name : RRPSSNRS

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when the RREVrqin event occurred.
3	response Bytes	R	Response to the client, placed in the buffer given by RRPSgtrq.

FUNCTION. . : Sends response.

EXPLANATION : With this call you send a response to the client's request.

Return occurs only after the first part of a long response is sent. Transmission of the remainder will occur during one or more succeeding RRPBwait(s).

USAGE . . . : RRPSsnrs(static,remoteID,response) =: RRstat

5.2 RR-LIB High Level Client - Calls

Routine name : RRPBINIT

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	dynamic Integer array	R	Work area for connection control blocks.
3	mode Rrmd	R	Should be specified as RRMDasClient or RRMDasBoth.
4	maxClientConn Integer	R	Maximum number of simultaneous connections to servers.
5	serverName Bytes	R	Dummy for clients.
6	bufferArea Bytes array	R	Dummy for clients.
7	maxServerConn Integer	R	Dummy for clients.
8	serverParam Rrsp	R	Dummy for clients.

FUNCTION. . : Initializes RR-LIB data structures.

EXPLANATION : This must be the first call to RR-LIB. You should use standard sizes for the RR-LIB work areas: static and dynamic. These required sizes are given in the RRP:DEFS file.

USAGE . . . :
```
RRPBinit(static,dynamic,mode,maxClientConn,
         serverName,bufferArea,maxServerConn,
         serverParam) =: RRstatus
```

Routine name : RRPCCALL

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	The identifier returned by RRPCSLCT.
3	request bytes	R	The client data to be transmitted to the server.
4	responseBuffer Bytes pointer	R	A buffer in which the server can place the response.
5	timeout Rrtm	R	Maximum time to wait for a response.
6	response Bytes pointer	W	Server response.
7	reason Integer	W	Reason for the disconnection or unexpected event value.

FUNCTION. . : Sends request and waits for response.

EXPLANATION : This call is equivalent to RRPCsnrq+RRPBwait+RRPCgtrs.

With this call you may wait for only one connection at a time. For handling several connections, low-level-client calls can be more efficient.

request and responseBuffer must start on a word boundary. The buffer used for the request and the response may be the same.

responseBuffer must be large enough to contain the entire response from the server.

The Rrtm type in timeout has the following definition:

```
TYPE RRTM = RECORD
       INTEGER:  RRTMlength
       INTEGER:  RRTMunits
ENDRECORD
```

Continued on next page.

RRTMlength is the length of the waiting time. RRTMunits is defined as for the SINTRAN HOLD command:

1 = Basic time units
2 = Seconds
3 = Minutes
4 = Hours

If the reason for the return is timeout and the client does not wish to continue to wait, a disconnect request should be issued.

reason gives the eventual reason if you were disconnected. If the return was caused by an unexpected event, then reason specifies it.

response contains the response from the server if, and only if, status was OK.

USAGE . . . : RRPCcall(static,remoteID,request,responsebuffer,
timeout,response,reason) =: RRstatus

Routine name : RRPCSLCT			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	destSys Bytes	R	String identifying the system where the server resides.
3	destServer Bytes	R	Server name.
4	clientInfo Bytes	R	Information from client to server (for example an identification).
5	serverInfoBuff Bytes pointer	R	Buffer in which to place information from the server back to you.
6	timeout Rrtm	R	Maximum waiting time for response from the server.
7	remoteID Rrid	W	Server reference number to be used in subsequent calls.
8	serverInfo Bytes pointer	W	Information returned by the server, or disconnect information.
9	reason Integer	W	Reason for the disconnection or unexpected event value.

FUNCTION. . : Selects a server.

EXPLANATION : A server is selected for future request/response interactions.

This call is equivalent to RRPCcnrq+RRPBwait+RRPCcncf/RRPBdcin. This is done synchronously, i.e., the return does not occur until timeout or until a response is received from the server.

If the reason is timeout and the client does not wish to continue to wait, a request to disconnect should be made.

clientInfo and serverInfoBuff must start on a word boundary.

Connection setup overhead is reduced if clientInfo is also used to include the first request to server and RRPScnrs is used to return the response.

serverInfoBuff must be large enough to contain the entire response from the server.

The Rrtm type in timeout has the following definition:

```
TYPE RRTM = RECORD
      INTEGER:  RRTMlength
      INTEGER:  RRTMunits
ENDRECORD
```

RRTMlength is the length of the waiting time. RRTMunits is defined as for the SINTRAN HOLD command:

1 = Basic time units
2 = Seconds
3 = Minutes
4 = Hours

reason gives the reason if you were disconnected. If the return was caused by an unexpected event, then reason specifies it.

serverInfo contains the information returned by the server if, and only if, status was OK (i.e., connected) or disconnected and if the reason parameter indicates that the disconnect was executed by the server.

reason gives the reason if you were disconnected. If the return was caused by an unexpected event, then reason specifies it.

USAGE . . . :
```
RRPCslct(static,destSys,destServer,clientInfo,
         serverInfoBuff,timeout,remoteID,serverInfo,
         reason) =: RRstatus
```

Routine name : RRPCDISC

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size for work space.
2	remoteID Rrid	R	ID returned from RRPCslct.
3	info Bytes	R	Information for remote end.
4	timeout Rrtm	R	Maximum time to wait for completion.
5	cause Integer	W	If unexpected event caused return, this specifies the event.

FUNCTION. . : Disconnect request with wait for completion.

EXPLANATION : This call is equivalent to RRPBdcrq+RRPBwait.

info must start on a word boundary.

USAGE . . . : RRPCdisc(static,remoteID,info,timeout,
cause) =: RRstatus

5.3 RR-LIB Low-Level Client Calls

Routine name : RRPBABRT			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size for work space.
2	remoteID Rrid	R	ID returned from RRPCcnrq.

FUNCTION. . : Abort connection.

EXPLANATION : This end of a connection is aborted. You should use this call only if a attempt to disconnect: RRPBdcrq does not work, i.e., if you do not get a confirmation after a reasonable time.

USAGE . . . : RRPBabrt(static,remoteID) =: RRstatus

Routine name : RRPBDCIN			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned in actualEvent when disconnect was signalled.
3	reason Integer	W	Reason for the disconnection.
4	info Bytes pointer	W	Information from the server (if server-initiated disconnect).

FUNCTION. . : Disconnect indication.

EXPLANATION : A disconnect request from the server is perceived by you as a disconnect indication.

This routine should only be called after the disconnect indication event (RREVdcin) is received.

It should not be called immediately if a "disconnect pending" error return is obtained from another call. Otherwise not all of the associated data will have been received.

USAGE . . . : RRPBdcin(static,remoteID,reason,info) =: RRstatus

Routine name : RRPBDCRQ			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	ID returned from RRPCcnrq.
3	info Bytes	R	Information for remote end.

FUNCTION. . : Disconnect request.

EXPLANATION : This call disconnects you from a server. It is used to break a connection already established or one you are in the process of establishing.

USAGE . . . : RRPBdcrq(static,remoteID,info) =: RRstatus

Routine name : RRPBINIT

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	dynamic Integer array	R	Work area for connection control blocks.
3	mode Rrmd	R	Should be specified as RRMDasClient or RRMDasBoth.
4	maxClientConn Integer	R	Maximum number of simultaneous connections to servers.
5	serverName Bytes	R	Dummy for clients.
6	bufferArea Bytes array	R	Dummy for clients.
7	maxServerConn Integer	R	Dummy for clients.
8	serverParam Rrsp	R	Dummy for clients.

FUNCTION. . : Initializes RR-LIB data structures.

EXPLANATION : This must be the first call to RR-LIB. You should use standard sizes for the RR-LIB work: areas static and dynamic. These required sizes are given in the RRP:DEFS file.

USAGE . . . :
```
RRPBinit(static,dynamic,mode,maxClientConn,
         serverName,bufferArea,maxServerConn,
         serverParam) =: RRstatus
```

Routine name : RRPBWAIT

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	timeout Rrtm	R	Maximum waiting time.
3	requestedEvent Rrev	R	Requested event. This includes the connection identifier.
4	actualEvent Rrev	W	The event that actually occurred.

FUNCTION. . : Waits for event to occur.

EXPLANATION : This is the only call in which arriving XMSG messages (responses or acknowledgements for parts of requests) are processed.

The Rrtm type in timeout has the following definition:

```
TYPE RRTM = RECORD
      INTEGER:  RRTMlength
      INTEGER:  RRTMunits
ENDRECORD
```

RRTMlength is the length of the waiting time. RRTMunits is defined as for the SINTRAN HOLD command:

1 = Basic time units
2 = Seconds
3 = Minutes
4 = Hours

A value of 0 for RRTMlength is equivalent to a poll for outstanding RR-LIB events of the requested type, on the requested connection.

The set of desired events is formed from the logical OR of the corresponding event codes. With the exception of RREVunkn and RREVothr, you will only receive events permitted by requestedEvent. Your set of desired events should always include RREVdcin, because a disconnect may occur at any time.

A value of RRanyRemote may be used to accept any connection. Likewise, RRanyEvent may be used to accept any event.

Event codes used to form a bit mask:

```
CONSTANT RREVtime =    1    % timeout
CONSTANT RREVcncf =    4    % connection accepted
CONSTANT RREVrsin =   16    % response has arrived
CONSTANT RREVdcin =   32    % disconnected from remote
CONSTANT RREVdccf =   64    % user disconnect complete
CONSTANT RREVunkn =  128    % XMSG arrived on port
                            % not known to RR-LIB
CONSTANT RREVothr =  256    % exit caused by other than
                            % timeout, an RR-LIB event,
                            % or unknown port
                            % e.g., by terminal input
```

Structure of an event:

```
CONSTANT RRMXevData = 2     % Length of the associated
                            % event data.

TYPE RREV = RECORD
       RRID:           RREVremote % ID for remote task
       INTEGER:        RREVevent  % BIT MASK of events
       INTEGER ARRAY: RREVdata(0:RRMXevData-1)
ENDRECORD
```

RREVdata contains data associated with the event. The meaning of these depends on the RREVevent.

If RREVevent = RREVunkn, then RREVdata (0) gives the XMSG portNumber. In this case the return from RRPBwait is equivalent to a return from the XMSG general status function XFGST. Note that the arrived message must be received (XFRCV) before RRABwait is called again.

USAGE . . . :
```
RRPBwait(static,timeout,requestedEvent,
         actualEvent) =: RRstatus
```

Routine name : RRPCCNCF			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	Identifier returned in actualEvent.
3	serverInfo Bytes pointer	W	The information returned by the server.

FUNCTION. . : Connection confirmation.

EXPLANATION : This call is used to process the connection confirmation event (RREVcncf). You obtain the data associated with the server's acceptance of a connect request.

USAGE . . . : RRPCcncf(static,remoteID,serverInfo) =: RRstatus

Routine name : **RRPCCNRQ**

No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	destSys Bytes	R	String identifying the system where the server resides.
3	destServer Bytes	R	Server name.
4	clientInfo Bytes	R	Information from client to server (for example an identification).
5	serverInfoBuff Bytes pointer	R	Buffer in which to place information from server to client.
6	remoteID Rrid	W	Server reference number to be used in subsequent calls.

FUNCTION. . : Connection request.

EXPLANATION : A connection between the client and a server is initiated.

clientInfo and serverInfoBuffer must start on a word boundary.

The connection setup overhead is reduced if clientInfo is also used to include the first request to the server. The server could include the response in serverInfo of RRPScnrs.

USAGE . . . :
```
RRPCcnrq(static,destSys,destServer,clientInfo,
         serverInfoBuff,remoteID) =: RRstatus
```

Routine name : RRPCEND			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.

FUNCTION. . : Ends the client functions.

EXPLANATION : Used when no more calls to servers are to be made. This call closes XMSG ports, releases XMSG buffers, etc. You should disconnect all active connections first, so the servers are aware that you are gone.

USAGE . . . : RRPCend(static) =: RRstatus

Routine name : RRPCGTRS			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	Identifier returned in actualEvent as an RREVrsin occurred.
3	response Bytes pointer	W	The response from the server.

FUNCTION. . : Gets response.

EXPLANATION : With this call you get the response to a previous request after a response event (RREVrsin) has been signalled. Your response is stored in the response buffer, provided by RRPCsnrq.

USAGE . . . : RRPCgtrs(static,remoteID,response) =: RRstatus

Routine name : RRPCSNRQ			
No:	Parameter Name/ Type:	R/W	Explanation:
1	static Integer array	R	Fixed size work space.
2	remoteID Rrid	R	The identifier returned by RRPCcnrq.
3	request Bytes	R	The client data to be transmitted to the server.
4	responseBuffer Bytes pointer	R	A buffer in which the server can place the response.

FUNCTION. . : Sends request.

EXPLANATION : This call initiates the sending of a request of arbitrary length. Return only occurs after the first part of a long request is sent. Transmission of the remainder will occur during one or more succeeding RRPBwait(s).

request and responseBuffer must start on a word boundary. The buffer used for the request and the response may be the same.

USAGE . . . :
```
RRPCsnrq(static,remoteID,request,responseBuffer)
         =: RRstatus
```

CHAPTER 6

INTRODUCTION TO TLIB

6 INTRODUCTION TO TLIB

6.1 Introduction

TLIB stands for Transport LIBrary. It is a tool for data transfer between RT programs (including background programs). These programs may or may not reside in different systems.

The services offered by TLIB are similar to those offered by the OSI Transport Service Specification. Many of the TLIB concepts are taken from the OSI reference model. The current implementation of TLIB consists of a set of routines interfacing with XMSG.

The advantages gained by the use of TLIB, rather than by the direct use of XMSG, are:

- an XMSG-independent interface, similar to the OSI Transport Service
- the provision of a flow-control mechanism, which means that the sending rate is adapted to the receiving rate

The disadvantages are:

- increased program size (see section 6.8.1)
- slightly decreased performance (see section 6.8.2)

6.2 General Concepts in TLIB

6.2.1 Connection

Before data can be exchanged between two communicating parties, a connection between them has to be established. When the data exchange is finished, the connection has to be terminated. We will refer to the connection establishment as the connect phase, the exchange of data as the data transfer phase, and the termination of a connection as the disconnect phase.

One RT program may operate several connections simultaneously. This means that you may communicate with several remote users at the same time. It also means that you may have several dialogues at the same time, with the same remote user, on different connections.

Each end point of a connection is identified by two connection identifiers:

1) The connection identifier selected by you as a user, is referred to by TLIB, when TLIB contacts you. For each new connection, you are free to choose a number for your identifier.

2) The connection identifier selected by TLIB is referred to by you, when you contact TLIB.

The reason for using two identifiers, rather than one, is that it provides a convenient way of referencing data belonging to a specific connection when several connections are open. You may build a table, in your program, with connection dependent data. Each table entry will correspond to a particular connection. When TLIB contacts you, the connection identifier that you get will give you the right table entry. When you contact TLIB, this is the technique that TLIB uses, and that is why TLIB wants its own connection identifier.

6.2.2 Ordinary Data and User Data

Data transmitted during the data transfer phase is referred to as ordinary data.

It is possible to send a smaller amount of data during the connect and disconnect phases. When a user requests a connection with another user, the former may want to give the latter some extra information besides the pure request for contact. The user receiving the connection request always has a choice of whether or not to accept the request. Thus it can make its choice dependent upon information coming from the sender, for example a password. When one user wants to disconnect the communication, it may want to provide a reason. Such extra information during connect or disconnect is referred to as user data.

6.2.3 Expedited Data

This is a limited amount of data which can be sent to the remote user, during the data transfer phase, outside the ordinary data stream. This data is not bound by the same flow control as ordinary data. The arrival of the expedited data will be signalled to the remote user before any ordinary data subsequently is sent. It may or may not be signalled to the remote user ahead of ordinary data sent previously.

Expedited data is intended only for the exchange of urgent user-level control information, which must not be blocked by the normal data flow.

6.2.4 Limitations on User Data and Expedited Data

User data is limited to 32 bytes (TLMXuserdata = 32 is defined in the TLP:DEFS and the TLF:DEFS files).

The amount of expedited data is limited to 8 bytes (TLMXexpeditedData = 8 is defined in the TLP:DEFS and TLF:DEFS files).

You, as a user, may not redefine these limits.

6.2.5 TLIB Service Data Unit - TSDU

During the data transfer phase, before you send data to a remote user, you divide the data into one or more logical units. Each unit is called a TSDU and is delimited by the end-of-TSDU flag. This flag is set by you when you send the data and it is passed on by TLIB to the receiving user. Each logical unit of data you receive from a remote user, during the data transfer phase, is also a TSDU. The TSDU can be of any length, and may be split arbitrarily between several user buffers.

Note that TLIB can buffer data internally, in the transport system, if you have not set the end-of-TSDU flag. To ensure that the last data given to TLIB has been transmitted to the remote user you must set this flag.

6.2.6 TLIB Protocol Data Unit - TPDU

These are the units of communication that TLIB uses, i.e., they are internal to the transport system. When you send a TSDU, TLIB will add a protocol header before it transmits the data. TLIB may split a TSDU into several TPDUs, depending on the size of the TSDU. However, such a splitting is not perceived by you as a user.

The data that TLIB receives from the remote end is one or more TPDUs. TLIB transfers the TSDU to you, i.e., the headers are stripped from the TPDUs.

Some TPDUs do not contain any data coming from a user. These are TPDUs used by TLIB for communication control purposes.

6.2.7 Credit

TLIB's flow control mechanism is based on the use of "credit". The purpose of the flow control is to avoid congestion in the transport system.

At any given point during the data transfer phase, there is a limit on the amount of data that you are allowed to send. This amount of data is termed your "credit". Your credit decreases as you send data, because you are using buffers in the transport system. Your credit increases as the remote user removes the incoming data from the transport system. The buffering strategy of the remote user, therefore, controls the rate at which you are permitted to send to him.

During the connect phase, you receive an initial value for your credit. New credit values are returned each time you send data. Credit changes, resulting from actions by the remote user or TLIB, are signalled to you by the "event" mechanism (see section 6.2.9). Before you send ordinary data, you should always check that you have enough credit.

6.2.8 TLIB Access Point - TLAP

This is an address, in a form accepted by TLIB, at which a user is accessible. This address identifies the user regardless of physical location.

There are two ways to describe a TLAP:

1) You may use the name of the system where the TLAP resides, and a "TLAP suffix". The suffix is a name, of your choice, which identifies the TLAP within the system.

2) You may use the "magic number", which uniquely defines the TLAP within the network.

In the connect phase, a connection is established between two TLAPs. You may establish several connections between the same TLAPs. Once a connection is established, TSDUs are transferred between the TLAPs.

6.2.9 Events

Certain changes in the communication system, resulting from actions by the remote user or TLIB, are signalled to you by the "event" mechanism.

Examples of events are: timeout, change in credit, and TLIB has received data. Some events require processing by calls to TLIB.

The only way an event can be signalled to you is through the TLPWAIT call.

6.3 General Information about the Routines

The routine implementation in PLANC is:

```
ROUTINE VOID,VOID (parameters.....)
Example of call: TLPTMLS(tlib_refno,retstat)
```

In FORTRAN a subroutine implementation is used.

```
Example of call: CALL TLFTMLS(ITLREF,ISTAT)
```

Normal return status is zero. If you include the appropriate definition file in your source code, you may use the symbol OK for the zero status. The file is called TLP:DEFS for PLANC and TLF:DEFS for FORTRAN. Other return status are error codes. A list of the error codes plus their corresponding symbols is provided in appendix F.

Notice that when we refer to a routine in this chapter, it starts with the letters TLP. This is how the PLANC routines are named. The FORTRAN routines start with TLF instead of TLP

6.4 Table of Events

The symbols for the event codes are defined in the TLP:DEFS and the TLF:DEFS files, for PLANC and FORTRAN respectively. You may specify a whole set of events by using logical OR to form a bit mask.

Event code	Value	Explanation
TLEVtime	1	Timeout
TLEVcnin	2	Connection request has arrived
TLEVcncf	4	Connection is accepted
TLEVdain	8	Data has arrived
TLEVcrdt	16	Credit change
TLEVxdin	32	Expedited data has arrived
TLEVxcdt	64	You may send more expedited data
TLEVdcin	128	Disconnect due to remote user or TLIB
TLEVdccf	256	Requested disconnection is completed
TLEVunkn	1024	Message arrived on port not known to TLIB
TLEVothr	2048	Not a TLIB event, timeout, or unknown port

The reason for TLEVunkn is that you may combine the use of TLIB and XMSG in the same RT program. As you performed the TLPWAIT call, to wait for an event, a message may arrive on a port opened by an XMSG call. The TLEVunkn tells you that this has happened.

The TLPWAIT routine is at some point performing the 'tmout' monitor call. If your RT-program becomes rescheduled for execution, for example by another program, the TLEVothr event will occur.

6.5 Handling of User Buffers

TLIB has no buffers of its own. This means that you have to provide all the input buffers, by using the TLPPRBF call. Incoming data is transferred directly from XMSG space into these buffers. The contents of your output buffers are transferred by TLIB directly into XMSG space.

When you have provided TLIB with a buffer, TLIB "owns" this buffer, although it still physically resides in your program area. Such a buffer always becomes allocated to a specific connection. To get the buffer back from TLIB with valid data, you have to do one of the following calls:

TLPCNIN, TLPCNCF, TLPDAIN, TLPEDIN, or TLPDCIN

You may have several buffers outstanding (i.e., owned by TLIB) to receive ordinary data. TLIB automatically transfers the data into the buffers when the data arrives. Having receive buffers available on a connection will increase throughput on that connection. If TLIB does not have buffers available, the received data has to wait in the transport system. Thus the remote user may be prevented from sending further data until you provide TLIB with the necessary buffer(s).

If a connection indication, a connection confirmation, or a disconnect indication contains user data, then the first, sufficiently large buffer provided, following the event, is used for the user data. In the presence of user data, such a buffer must be provided between the event and the call TLPCNIN, TLPCNCF, or TLPDCIN.

The first, sufficiently large, buffer provided on a connection after you are notified of the arrival of expedited data (TLEVxdin event), is used for the expedited data. Such a buffer must, therefore, be provided between the TLEVxdin event and the call TLPEDIN.

All buffers provided for TLIB are queued for ordinary data, if no user data or expedited data is waiting. All outstanding buffers on a connection are automatically returned to you at the end of the connection, i.e., following TLPDCRQ or TLPDCIN. Once a buffer has been returned to you either with data, empty, or with an error return, TLIB will not access the buffer again until you do another TLPPRBF on it.

6.6 Summary of the Different Routines

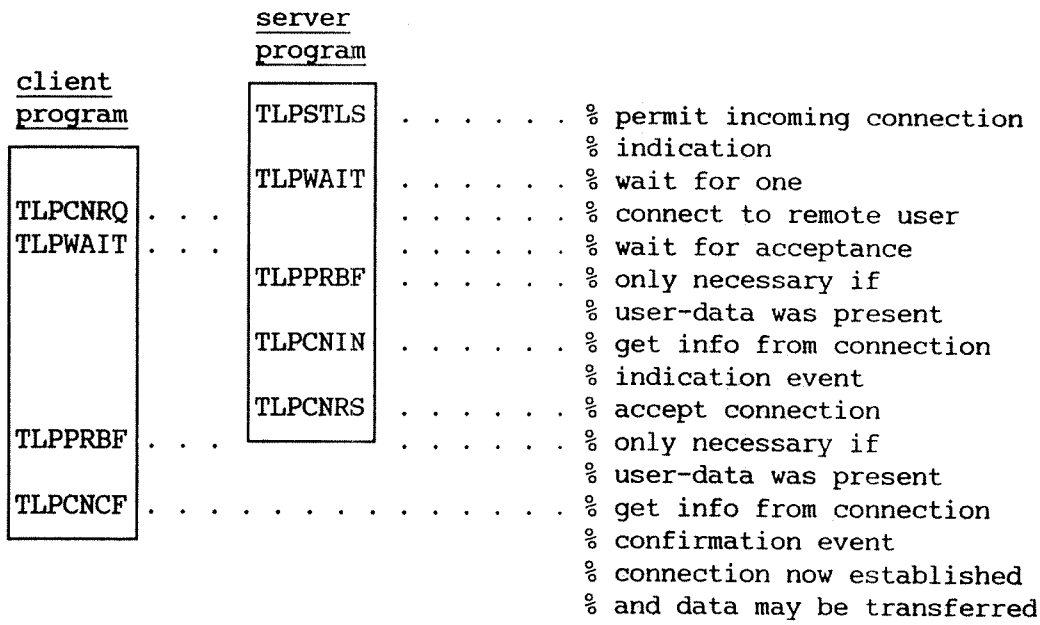
The different routines are listed in the following table. Notice that they start with the symbol TLP. This is how the PLANC routines are named. The corresponding FORTRAN routines start with TLF instead of TLP. The calls are listed alphabetically.

Routine	Purpose	Phase where used
TLPCNCF	Connection confirmation	Connection establishment
TLPCNIN	Connection indication	Connection establishment
TLPCNRQ	Connection request	Connection establishment
TLPCNRS	Connection response	Connection establishment
TLPDAIN	Data indication	Data transfer
TLPDARQ	Data request	Data transfer
TLPDCIN	Disconnect indication	Disconnect
TLPDCRQ	Disconnect request	Disconnect
TLPEDIN	Expedited data indication	Data transfer
TLPEDRQ	Expedited data request	Data transfer
TLPINIT	Initialize TLIB	
TLPPRBF	Provide buffer	Used in all phases
TLPSTLS	Start listening	Connection establishment
TLPTMLS	Terminate listening	Connection establishment
TLPWAIT	Wait for event to occur	Used in all phases

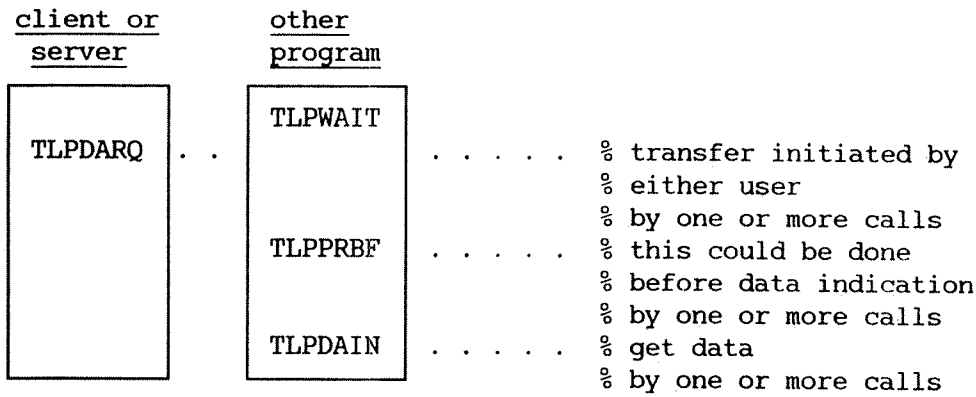
6.7 Example of Use

In this example, the calling user is termed "client", and the called user is termed "server".

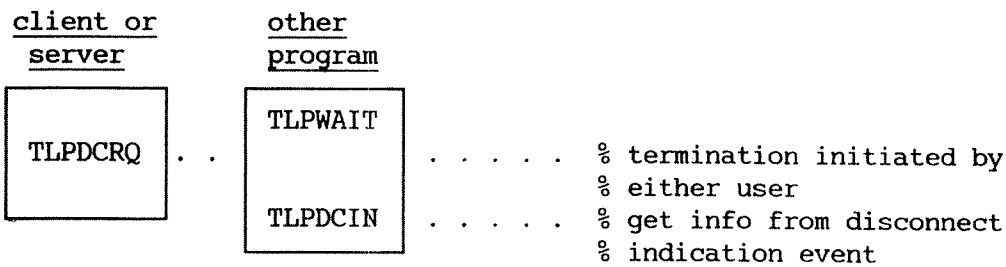
Successful Connection Establishment



Data Transfer - One or More Instances of the Following Sequence



User-Initiated Termination of Established Connection



6.8 XMSG-based TLIB

This section discusses various aspects of the version of TLIB based on XMSG.

6.8.1 TLIB Size

Code size is approximately 8.5K words.

The stack plus a data area, which is independent of the maximum number of connections etc., require approximately 800 words.

In addition, the space required for data which depends on library parameters is currently, approximately:

30 words per connection
25 words per XMSG port
10 words per queued user buffer
10 words per TPDU queued in XMSG space

6.8.2 Speed

The speed is discussed in terms of additional XMSG monitor calls during the data transfer phase of a connection compared to direct XMSG use.

The extra overheads are as follows:

i) Each data TPDU requires an extra read/write monitor call for the TLIB protocol header.

ii) Depending on the relative lengths of the user's logical data unit (TSDU) and the TLIB transfer unit (data TPDU), the TSDU may require splitting into a number of data TPDUs.

iii) The implementation of flow control requires the use of an ACK TPDU, carried by an XMSG message. This requires processing at both ends. One or more ACKs are generated when data from a TSDU is transferred into user space.

iv) Because TLIB is a general-purpose transport interface, it can make no assumptions about traffic flow. This can lead to inefficient use (unnecessary get/release) of XMSG buffers in the context of particular traffic patterns, e.g. transactions.

CHAPTER 7

TLIB/PLANC REFERENCE GUIDE

7 TLIB/PLANC REFERENCE GUIDE

7.1 The TLIB/PLANC Record Types

The following sections briefly explains the record types used in the TLIB PLANC interface.

7.2 Record Types

All the records are defined in the TLP:DEFS file. You should include it in your program.

7.2.1 Identification of a TLAP

The following record is used to define the format of a name. Names currently defined this way are TLAPnetAddress and TLAPsuffix.

```
TYPE TLNM = RECORD
        INTEGER4: TLNMlength
        BYTES: TLNMstring(0:TLMXnameLength - 1)
ENDRECORD
```

Currently there are two ways of defining a TLAP: Using the TLAP2String format, or the TLAPXMSG format. The type TLAP defines a base record for the two variants.

```
TYPE TLAP = RECORD
        INTEGER4: TLAPformat
ENDRECORD
```

When using the TLAP2String format, the TLAP is defined by a "network address" and a "TLAP suffix". A "network address" is the same as a system name. A "TLAP suffix" is a name used to identify the TLAP within the system. Both the network address and the TLAP suffix have a limit of 32 characters (TLMXnamelength = 32). TLAPformat must be set to TLtlap2String.

```
TYPE TLAP2String  = TLAP RECORD
        TLNM: TLAPnetAddress
        TLNM: TLAPsuffix
ENDRECORD
```

When using the TLAPXMSG format, the TLAP is defined by a magic number. TLAPformat must be set to TLtlapMagic.

```
TYPE TLAPXMSG = TLAP RECORD
        INTEGER4: TLAPmagic
ENDRECORD
```

7.2.2 Quality of Service

Currently, only default quality of service is provided. The value of TLQSisDefault has to be TRUE.

```
TYPE TLQS = RECORD
        BOOLEAN: TLQSisDefault
ENDRECORD
```

7.2.3 User Buffer Specification

This is a base record for variants. TLBFid is the buffer identifier. TLBFformat is the code for the address format.

```
TYPE TLBF = RECORD
        INTEGER: TLBFid
        INTEGER: TLBFformat
ENDRECORD
```

So far there is only one buffer address format: TLlogicalAddress. This is the corresponding buffer address variant record:

```
TYPE TLBFLogical = TLBF RECORD
        INTEGER: TLBFaddress
ENDRECORD
```

7.2.4 Structure of an Event

An event has the following structure:

```
TYPE TLEV = RECORD
        INTEGER: TLEVrefno
        INTEGER: TLEVcode
        INTEGER ARRAY: TLEVdata(0:TLMXevData-1)
ENDRECORD
```

TLMXevdata is currently set to 2.

TLEVrefno is a connection identifier belonging to either the user or TLIB.

TLEVcode is a bit mask of event codes. If this is a received event, TLEVcode is just one single event. See table of events on page 207.

TLEVdata has meaning only if this is a received event. The meaning is dependent upon the type of event:

TLEVcode = TLEVdain	TLEVdata(0) gives the total amount of data received since the last signalled event of this type. TLEVdata(1) gives the number of complete TSDUs in the total data.
TLEVcode = TLEVxdin	TLEVdata(0) gives the length of the expedited data.
TLEVcode = TLEVcnin or = TLEVcncf or = TLEVdcin	TLEVdata(0) gives the length of the user data.
TLEVcode = TLEVcrdt	TLEVdata(0) gives the new value of the user's credit.
TLEVcode = TLEVunkn	TLEVdata(0) gives the XMSG port number. In this case, TLEVrefno has no significance. If the TLEVunkn event occurs, the return from TLPWAIT is equivalent to a return from the XMSG general status call (XMPFGST). Note that the message must be received before TLPWAIT is called again.

7.2.5 TLIB/PLANC Reference Section

Routine name : TLPCNCF

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.
3	quality Tlqs	W	Quality of service.
4	initial_credit Integer	W	Number of bytes you are allowed to send initially. This may be zero.
5	userdata Tlbf	W	User data.
6	length_userdata Integer	W	Length of user data in bytes.

FUNCTION. . : Connection confirmation.

EXPLANATION : This call is used to receive the information associated with connection establishment, after TLEVcncf is signalled.

A connection response coming from the other end will be perceived by TLIB as a connection confirmation. This is being flagged as a TLEVcncf event. The W parameters in TLPCNCF are the data associated with this event.

If user data was sent by the remote end, it is placed in the first, sufficiently large buffer provided on the connection, after TLEVcncf is signalled. An error return results if user data is present and no such buffer was provided.

USAGE . . . :
```
tlpcncf(tlib_refno,retstat,quality,initial_credit,&
        userdata,length_userdata)
```

Routine name : TLPCNIN

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.
3	remote_address Tlap	W	Remote TLAP, given as magic number.
4	quality Tlqs	W	Quality of service.
5	initial_credit Integer	W	Number of bytes you are permitted to send initially (may be zero).
6	userdata Tlbf	W	User data.
7	length_userdata Integer	W	Length of user data in bytes.

FUNCTION. . : Connection indication.

EXPLANATION : This call is used to receive the information associated with connection establishment, after TLEVcnin is signalled.

When you receive a connection request from a remote TLAP, it will be perceived by your task as a connection indication. TLIB will signal the connection indication as an event, TLEVcnin. The W parameters in TLPCNIN are information associated with this event.

If user data was sent by the remote end, it is placed in the first, sufficiently large buffer provided on the connection, after TLEVcnin is signalled. An error return will result if user data is present and no such buffer was provided.

USAGE . . . :
```
tlpcnin(tlib_refno,retstat,remote_address,quality&
        initial_credit,userdata,length_userdata)
```

Routine name : TLPCNRQ

No:	Parameter Name/ Type:	R/W	Explanation:
1	user_address Tlap	R	Local TLAP.
2	remote_address Tlap	R	Remote TLAP.
3	user_refno Integer	R	Connection identifier belonging to user.
4	quality Tlqs	R	Quality of service.
5	userdata Tlbf	R	User data. Must start on a word boundary.
6	length_userdata Integer	R	Length of user data. Must be an even number.
7	retstat Integer	W	Return status.
8	tlib_refno Integer	W	Connection identifier belonging to TLIB.

FUNCTION. . : Connection request.

EXPLANATION : A connection between the local and the remote TLAPs is initiated. Before the exchange of TSDUs can take place, the connection must be completely established.

USAGE . . . :
```
tlpcnrq(user_address,remote_address,user_refno,quality,&
        userdata,length_userdata,retstat,tlib_refno)
```

Routine name : TLPCNRS

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	quality Tlqs	R	Quality of service.
3	userdata Tlbf	R	User data. Must start on a word boundary.
4	length_userdata Integer	R	Length of user data. Must be an even number.
5	retstat Integer	W	Return status.

FUNCTION. . : Connection response.

EXPLANATION : This call will tell the remote user that the connection request is accepted. It should be invoked when the connection indication data has been retrieved by TLPCNIN.

USAGE . . . :
```
tlpcnrs(tlib_refno,quality,userdata,length_userdata,&
        retstat)
```

Routine name : TLPDAIN			
No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.
3	user_buffer Tlbf	W	Buffer previously given to TLIB for use on this connection.
4	length_data Integer	W	Length of received data.
5	flags Integer	W	Indicates whether this data represents the end of a TSDU.

FUNCTION. . : Data indication.

EXPLANATION : When data is received, it is signalled via the event TLEVdain. If TLIB is not yet provided with a data buffer, you must perform a TLPPRBF call. Then you may, at any time, obtain the data buffer from TLIB.

When it is full, TLIB puts the data buffer in a queue, waiting to be transferred to you later. This transfer takes place when you perform the TLPDAIN call. A user buffer is considered to be full when it is actually filled with data, or when it contains data representing the end of a TSDU. The user will not receive a buffer containing data from more than one TSDU.

A buffer partally full is returned by TLPDAIN when there is outstanding data on the connection and no full buffers.

If there is no outstanding data, and you have previously given a buffer to TLIB, TLPDAIN will return the buffer to you.

When you receive data representing the end of a TSDU, flags will be set to TLeotsdumark = 1. This symbol is defined in the TLP:DEFS file.

USAGE . . . :
```
tlpdain(tlib_refno,retstat,user_buffer,&
        length_data,flags)
```

Routine name : TLPDARQ			
No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	user_buffer Tlbf	R	TSDU, or part of a TSDU. Must start on a word boundary.
3	length_data Integer	R	Number of bytes in user_buffer. Must be even.
4	flags Integer	W	Indicates whether this data represents the end of a TSDU.
5	retstat Integer	W	Return status.
6	new_credit Integer	W	The credit you have after a successful TLPDARQ.

FUNCTION. . : Data request.

EXPLANATION : To transfer ordinary data to the remote user, you have to use this call.

If this data represents the end of a TSDU, flags must be set to tleotsdumark = 1. This symbol is defined in the TLP:DEFS file.

Make sure that flags are set to tleotsdumark when you invoke TLPDARQ for the last time. This will ensure that nothing will be left in the internal buffers of the transport system.

USAGE . . . :
```
tlpdarq(tlib_refno,user_buffer,length_data,&
        flags,retstat,new_credit)
```

Routine name : TLPDCIN

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.
3	disconn_reason Integer	W	Reason for disconnection coming from the remote end or from TLIB. See appendix F.
4	userdata Tlbf	W	User data.
5	length_userdata Integer	W	Length of user data in bytes.

FUNCTION. . : Disconnect indication.

EXPLANATION : An incoming disconnect request is perceived by TLIB as a disconnect indication. TLIB signals this to you via a TLEVdcin event. TLPDCIN gives you the information associated with this event.

USAGE . . . :
```
tlpdcin(tlib_refno,retstat,disconn_reason,&
        userdata,length_userdata)
```

Routine name : TLPDCRQ

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	userdata Tlbf	R	User data. Must start on a word boundary. Delivery is not guaranteed.
3	length_userdata Integer	R	Length of user data. Must be an even number.
4	retstat Integer	W	Return status.

FUNCTION. . : Disconnect request.

EXPLANATION : The purpose of this call is one of the following:

1) To refuse a connection request from a remote user, signalled by an TLEVcnin event.

2) To terminate an established connection.

Any outstanding buffered data not yet sent to the remote end is flushed. The same is true for received data not yet given to you.

When the disconnect is complete, TLIB will signal this to you via the TLEVdccf event.

No other calls may be invoked on the connection after you have done a TLPDCRQ.

USAGE . . . : tlpdcrq(tlib_refno,userdata,length_userdata,retstat)

Routine name : TLPEDIN

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.
3	user_buffer Tlbf	W	Buffer provided for expedited data.
4	length_expdata Integer	W	Number of bytes in buffer.

FUNCTION. . : Expedited data indication.

EXPLANATION : By using this call, you will receive a unit of expedited data whose arrival has been signalled by a TLEVxdin event.

The first, sufficiently large, user buffer provided to TLIB for the connection following the TLEVxdin event will be used for the expedited data. If no such buffer was provided, an error return will be given by TLPEDIN.

USAGE . . . :
```
tlpedin(tlib_refno,retstat,user_buffer,&
        length_expdata)
```

Routine name : TLPEDRQ

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	user_buffer Tlbf	R	User buffer. Must start on a word boundary.
3	length_expdata Integer	R	Number of bytes in user buffer. Must be even.
4	retstat Integer	W	Return status.

FUNCTION. . : Expedited data request.

EXPLANATION : To transfer expedited data to the remote user, this call is provided.

TLIB guarantees that the arrival of the expedited data will be signalled to the remote user, before any ordinary data subsequently will be sent on the connection. The arrival of the expedited data may or may not be signalled to the remote user ahead of ordinary data sent previously.

Only one unit of expedited data may be outstanding at a time. You may send no more until you receive a TLEVxcdt event.

USAGE . . . :
```
tlpedrq(tlib_refno,user_buffer,length_expdata,&
        retstat)
```

Routine name : TLPINIT

No:	Parameter Name/ Type:	R/W	Explanation:
1	desired_mode Tlmd	R	Specification of TLIB operating mode.
2	tldynamic Integer array	R	Working storage for TLIB.
3	retstat Integer	W	Return status.

FUNCTION. . : Initializes TLIB. This must be the first call on TLIB.

EXPLANATION : TLIB operating mode is specified by the TLMD record which is defined in the TLP:DEFS file:

```
TYPE TLMD = RECORD
      TLXMuse: TLMDxmsgMode
      BOOLEAN: TLMDuniqueSuffix
      INTEGER: TLMDmxConnections
      INTEGER: TLMDmxAccessPoints
      INTEGER: TLMDmxBuffers
      INTEGER: TLMDmxQueuedTpdus
ENDRECORD
```

```
TLXMuse is defined by:
TYPE TLXMuse = ENUMERATION (TLXMinUserMode,
                            TLXMinSystemMode)
```

Note that TLXMinSystemMode is very rarely used. If you really need to use it, we refer you to the XMSG description.

TLMDuniqueSuffix is set to TRUE if suffix names should be unique for this task.

TLMDmxConnections is the maximum number of connections you want to use in this task. Standard value TLMXconnections = 20 may be used.

TLMDmxAccessPoints is the maximum number of TLAPs you want to use in this task. Standard value TLMXaccesspoints =3 may be used.

TLMDmxbuffers is the maximum number of user buffers in your task that TLIB can own at one time. Standard value TLMXbuffers = 20 may be used.

TLMDmxQueuedTpdus is the maximum number of TLIB TPDUs permitted to be queued in XMSG space at one time. Standard value TLMXqueuedTpdus = 50 may be used.

The size of tldynamic is dependent upon the values described above. You dimension it as follows:

INTEGER ARRAY: TLDYNAMIC(0:TLSZdynamic-1).

TLSZdynamic will automatically be calculated for you.

USAGE . . . : tlpinit(desired_mode,tldynamic,retstat)

Routine name : TLPPRBF

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	user_buffer Tlbf	R	User buffer. Must start on a word boundary.
3	length_buffer Integer	R	Length of user buffer. Must be an even number of bytes.
4	retstat Integer	W	Return status.

FUNCTION. . : Provides buffer.

EXPLANATION : A buffer is provided for use by TLIB. Whenever ordinary data arrives, TLIB stores the data in this buffer.

For user data or expedited data, a buffer has to be provided where the event, that signals the incoming data, has occurred.

To get the buffer back from TLIB, you have to perform a TLIB call. For example, you use the TLPDAIN routine to obtain ordinary data. To obtain user data from a connection request, you use the TLPCNIN routine.

USAGE . . . :
```
tlpprbf(tlib_refno, user_buffer,&
        length_buffer,retstat)
```

Routine name : TLPSTLS

No:	Parameter Name/ Type:	R/W	Explanation:
1	user_address Tlap	R	Local TLAP.
2	user_refno Integer	R	Connection identifier belonging to user.
3	retstat Integer	W	Return status.
4	tlib_refno Integer	W	Connection identifier assigned by TLIB.

FUNCTION. . : Starts listening.

EXPLANATION : This call tells TLIB that you are willing to receive a connection indication on the specified TLAP. To receive multiple simultaneous connections on a TLAP, TLPSTLS must be invoked the corresponding number of times.

TLIB automatically re-initiates a listening for a new connection indication after the termination of a previous connection. The same tlib_refno and user__refno apply to successive connections.

If you refuse to accept further connection indications, TLPTMLS should be invoked.

USAGE . . . :
```
tlpstls(user_address,user_refno,retstat,&
        tlib_refno)
```

Routine name : TLPTMLS

No:	Parameter Name/ Type:	R/W	Explanation:
1	tlib_refno Integer	R	Connection identifier belonging to TLIB.
2	retstat Integer	W	Return status.

FUNCTION. . : Terminates listening.

EXPLANATION : If you want to tell TLIB that you refuse to accept further connection indications on this connection, you should use this call. In other words, TLPTMLS cancels the effect of the prior TLPSTLS.

USAGE . . . : tlptmls(tlib_refno,retstat)

Routine name : TLPWAIT			
No:	Parameter Name/ Type:	R/W	Explanation:
1	timeout Tltm	R	Maximum waiting time for event.
2	requested_event Tlev	R	Requested event.
3	retstat Integer	W	Return status.
4	actual_event Tlev	W	Actual event.

FUNCTION. . : Waits for event to occur.

EXPLANATION : See also important note on next page.

The Tltm type in timeout has the following definition:

```
TYPE TLTM = RECORD
      TLTMcode: TLTMunits
      INTEGER:  TLTMlength
ENDRECORD
```

TLTMcode is defined as follows:

```
TYPE TLTMcode = ENUMERATION (TLTMbasic,TLTMsecs,
                             TLTMmins,TLTMhrs)
```

These TLTMunits denote: basic time units, seconds, minutes, and hours respectively.

TLTMlength describes the number of time units.

A value of TLinfiniteTime for <timeout.TLTMlength> specifies an infinite timeout period.

A value of 0 for <timeout.TLTMlength> is equivalent to a poll for outstanding TLIB events of the requested type, on the requested connection.

You may specify that TLIB should wait on a particular connection for a particular set of events.This is specified by appropriately setting the desired values in <requested_____event.TLEVrefno> and <requested event.TLEVcode>. <requested_event.TLEVrefno> requires a connection identifier belonging to TLIB.

Continued on next page.

The set of desired events is formed from the logical OR of the corresponding event codes.

A value of TLanyRefno may be used to accept any connection. Likewise, TLanyEvent may be used to accept any event.

The actual event that has occurred, is returned in actual_event, with <actual_event.TLEVrefno> set to a user_refno.

USAGE . . . :
```
tlpwait(timeout,requested_event,retstat,&
        actual_event)
```

IMPORTANT NOTE for those who use the TLEVothr event:

A routine TLOEV can be called as a part of TLPWAIT. If you use the TLEVothr event, you should provide a routine in your program called TLOEV:

```
ROUTINE STANDARD VOID,BOOLEAN: TLOEV
```

This routine will check all the possible "other" events you are using, e.g. for the presence of terminal input, and return TRUE if any are pending (and FALSE otherwise). TLIB provides a default version of TLOEV which always returns FALSE. If you want to make sure that no events of type TLEVothr are missed, you have to provide TLEOV in your program.

The reason that an "other" event may be missed, is that all wakeups from the wait state use one single bit in the program's RT-description (the 5REP bit). If two wakeups, one leading to an "other" event followed by one done by XMSG, occur close together just before TLIB executes the TMOUT monitor call, the information about the "other" event is lost.

CHAPTER 8

TLIB/FORTRAN REFERENCE GUIDE

8 TLIB/FORTRAN REFERENCE GUIDE

8.1 The TLIB/FORTRAN Data Types

The following sections briefly explains the data types used in the TLIB FORTRAN interface.

8.2 Data Specifications

All the symbols referred to, except array names like IRMADD and IEVENT, are defined in the TLF:DEFS file. You should include it in our program.

8.2.1 Specification of a TLAP

A TLAP is specified by an INTEGER*4 array. It may be dimensioned as follows :

```
    DIMENSION IRMADD(0:TLALTLAP-1)
```

where TLALTLAP = 3 + (TLMXNAMELENGTH/2).

The different positions in the array may be defined by the following symbols:

```
TLAPFORMAT = 0
TLAPLNNET = 1
TLAPNETADDRESS = 2
TLAPLNSUFFIX = 2+(TLMXNAMELENGTH/4)
TLAPSUFFIX = 3+(TLMXNAMELENGTH/4)
TLAPMAGIC = 1
```

IRMADD(TLAPFORMAT) specifies the format in which the address is given. There are two possible formats: TLTLAP2STRING or TLTLAPMAGIC.

If IRMADD(TLAPFORMAT) = TLTLAP2STRING, then the address is contained within the array as two character strings:

```
IRMADD(TLAPLNNET) gives the number of
characters in the "network address". The "network
address" is the same as the system name.
IRMADD(TLAPNETADDRESS), IRMADD(TLAPNETADDRESS+1)....
contain the packed character string for the network
address.
```

IRMADD(TLAPNSUFF) gives the number of characters in the "TLAP suffix". This suffix is a name used to identify the TLAP within the system.
IRMADD(TLAPSUFFIX), IRMADD(TLAPSUFFIX+1).....
contain the packed character string for the suffix.

If IRMADD(TLAPFORMAT) = TLTLAPMAGIC, then the address is given as a magic number. IRMADD(TLAPMAGIC) then contains the magic number.

8.2.2 Quality of Service

The quality of service is specified by an integer array. It may be dimensioned as follows: DIMENSION IQOS(0:TLALQOS-1) where TLALQOS = 1. Only the default quality of service is currently provided. One way of setting the default value is: IQOS(TLQSISDEFAULT) = TLISDEFAULT.

8.2.3 User Buffer Specification

Buffers are provided to TLIB in the form of INTEGER*2 ARRAYS. They are provided with a user-supplied identifier, which is used by TLIB to identify the particular buffer to the user when it is returned (since the user may have several buffers outstanding on the connection).

8.2.4 Structure of an Event

An event is specified by an integer array. It may be dimensioned as follows: DIMENSION IEVENT(0:TLALEVENT-1) where TLALEVENT = 4. The different positions in the array may be defined by the following symbols:

```
TLEVREFNO = 0
TLEVCODE  = 1
TLEV0DATA = 2
TLEV1DATA = 3
```

IEVENT(TLEVREFNO) is a connection identifier belonging to either the user or TLIB.

IEVENT(TLEVCODE) is a bit mask of event codes. If this is a received event, IEVENT(TLEVCODE) is just one single event. See table of events on page 207.

The last two elements of IEVENT have meaning only if this is a received event. The meaning is dependent upon the type of event:

(TLEVCODE) = TLEVDAIN	(TLEV0DATA) gives the total amount of data received since the last signalled event of this type. (TLEV1DATA) gives the number of complete TSDUs in the total data.
(TLEVCODE) = TLEVXDIN	(TLEV0DATA) gives the length of the expedited data.
(TLEVCODE) = TLEVCNIN or = TLEVCNCF or = TLEVDCIN	(TLEV0DATA) gives the length of the user data.
(TLEVCODE) = TLEVCRDT	(TLEV0DATA) gives the new value of the user's credit.
(TLEVCODE) = TLEVUNKN	(TLEV0DATA) gives the XMSG port number. In this case TLEVREFNO has no significance. If the TLEVUNKN event occurs, the return from TLFWAIT is equivalent to a return from the XMSG general status call (XMFFGST). Note that the message must be received before TLFWAIT is called again.

8.2.5 TLIB/FORTRAN Reference Section

Routine name : TLFCNCF

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.
3	IQOS Integer array	O	Quality of service.
4	ICREDT Integer	O	Number of bytes you are allowed to send initially. This may be zero.
5	IUBFID Integer	O	User data.
6	LENDAT Integer	O	Length of user data in bytes.

FUNCTION. . : Connection confirmation.

EXPLANATION : This call is used to receive the information associated with connection establishment, after TLEVCNCF is signalled.

A connection response coming from the other end will be perceived by TLIB as a connection confirmation. This is being flagged as a TLEVCNCF event. The output parameters in TLFCNCF are the data associated with this event.

If user data was sent by the remote end, it is placed in the first, sufficiently large buffer provided on the connection after TLEVCNCF is signalled. An error return results if no such buffer was provided.

USAGE . . . : CALL TLFCNCF(ITLREF,ISTAT,IQOS,ICREDT,IUBFID,LENDAT)

Routine name : TLFCNIN

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.
3	IRMADD Integer*4 array	O	Remote TLAP, given as magic number.
4	IQOS Integer array	O	Quality of service.
5	ICREDT Integer	O	Number of bytes you are permitted to send initially (may be zero).
6	IUBFID Integer	O	Identifier for user buffer.
7	LENDAT Integer	O	Length of user data in bytes.

FUNCTION. . : Connection indication.

EXPLANATION : This call is used to receive the information associated with the establishment of the connection after TLEVCNIN is signalled.

When you receive a connection request from a remote TLAP, it will be perceived by your task as a connection indication. TLIB will signal the connection indication as an event, TLEVCNIN. The W parameters in TLFCNIN are information associated with this event.

If user data was sent by the remote end, it is placed in the first, sufficiently large buffer provided on the connection after TLEVCNIN is signalled. An error return will result if no such buffer was provided.

USAGE . . . :
```
CALL TLFCNIN(ITLREF,ISTAT,IRMADD,IQOS,
             ICREDT,IUBFID,LENDAT)
```

Routine name : TLFCNRQ

No:	Parameter Name/ Type:	I/O	Explanation:
1	IUSADD Integer*4 array	I	Local TLAP.
2	IRMADD Integer*4 array	I	Remote TLAP.
3	IUSREF Integer	I	Connection identifier belonging to user.
4	IQOS Integer array	I	Quality of service.
5	IUBUF Integer*2 array	I	User data. Must start on a word boundary.
6	LENDAT Integer	I	Length of user data. Must be an even number.
7	ISTAT Integer	O	Return status.
8	ITLREF Integer	O	Connection identifier belonging to TLIB.

FUNCTION. . : Connection request.

EXPLANATION : A connection between the local and remote TLAPs is initiated. Before the exchange of TSDUs can take place, the connection must be completely established.

USAGE . . . :
```
CALL TLFCNRQ(IUSADD,IRMADD,IUSREF,IQOS,IUBUF,
             LENDAT,ISTAT,ITLREF)
```

Routine name : TLFCNRS

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	IQOS Integer array	I	Quality of service.
3	IUBUF Integer*2 array	I	User data. Must start on a word boundary.
4	LENDAT Integer	I	Length of user data. Must be an even number.
5	ISTAT Integer	O	Return status.

FUNCTION. . : Connection response.

EXPLANATION : This call is used to tell the remote user that the connection request is accepted. It must be invoked after the connection indication data has been retrieved by TLFCNIN.

USAGE . . . : CALL TLFCNRS(ITLREF,IQOS,IUBUF,LENDAT,ISTAT)

Routine name : TLFDAIN			
No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.
3	IUBFID Integer	O	Identifier for buffer previously given to TLIB for use on this connection.
4	LENDAT Integer	O	Length of received data.
5	IFLAGS Integer	O	Indicates whether this data represents the end of a TSDU.

FUNCTION. . : Data indication.

EXPLANATION : When data is received, it is signalled via the event TLEVDAIN. If TLIB is not yet provided with a data buffer, you must perform a TLFPRBF call. Then you may, at any time, obtain the data buffer from TLIB.

When it is full, TLIB puts the data buffer in a queue for an eventual transfer to you. This transfer from TLIB is initiated when you perform the TLFDAIN call. A user buffer is considered to be full when it is actually filled with data, or when it contains data representing the end of a TSDU. The user will not receive a buffer containing data from more than one TSDU.

A partially full buffer is returned by TLFDAIN when there is outstanding data on the connection, but no buffers are full.

If there is no outstanding data, and you have previously given a buffer to TLIB, TLFDAIN will return the buffer to you.

When you receive data representing the end of a TSDU, flags will be set to TLEOTSDUMARK = 1. This symbol is defined in the TLF:DEFS file.

USAGE . . . : CALL TLFDAIN(ITLREF,ISTAT,IUBFID,LENDAT,IFLAGS)

Routine name : TLFDARQ

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	IUBUF Integer*2 array	I	TSDU, or part of a TSDU. Must start on a word boundary.
3	LENDAT Integer	I	Number of bytes in IUBUF. Must be even.
4	IFLAGS Integer	O	Indicates whether this data represents the end of a TSDU.
5	ISTAT Integer	O	Return status.
6	ICREDT Integer	O	The credit you have after a successful TLFDARQ.

FUNCTION. . : Data request.

EXPLANATION : To transfer ordinary data to the remote user, you have to use this call.

If this data represents the end of a TSDU, flags must be set to TLEOTSDUMARK = 1. This symbol is defined in the TLF:DEFS file.

Make sure that IFLAGS is set to TLEOTSDUMARK when you invoke TLFDARQ for the last time, to ensure that nothing will be left in the internal buffers of the transport system.

USAGE . . . : CALL TLFDARQ(ITLREF,IUBUF,LENDAT,IFLAGS,ISTAT,ICREDT)

Routine name : TLFDCIN

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.
3	IDCRSN Integer	O	Reason for disconnection, coming from the remote end or from TLIB. See appendix F.
4	IUBFID Integer	O	Identifier for buffer with user data.
5	LENDAT Integer	O	Length of user data in bytes.

FUNCTION. . : Disconnect indication.

EXPLANATION : An incoming disconnect request is perceived by TLIB as a disconnect indication. TLIB signals this to you via a TLEVDCIN event. TLFDCIN also specifies the information associated with this event.

USAGE . . . : CALL TLFDCIN(ITLREF,ISTAT,IDCRSN,IUBFID,LENDAT)

Routine name : TLFDCRQ			
No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	IUBUF Integer*2 array	I	User data. Must start on a word boundary. Delivery is not guaranteed.
3	LENDAT Integer	I	Length of user data. Must be an even number.
4	ISTAT Integer	O	Return status.

FUNCTION. . : Disconnect request.

EXPLANATION : The purpose of this call is one of the following:

1) To refuse a connection request from a remote user, signalled by an TLEVCNIN event.

2) To terminate an established connection.

Any outstanding buffered data not yet sent to the remote end is flushed. The same is true for received data not yet given to you.

When the disconnection is complete, TLIB will signal this to you via the TLEVDCCF event.

No other calls may be invoked on the connection after you have done a TLFDCRQ.

USAGE . . . : CALL TLFDCRQ(ITLREF,IUBUF,LENDAT,ISTAT)

Routine name : TLFEDIN

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.
3	IUBFID Integer	O	Identifier for buffer provided for expedited data.
4	LENDAT Integer	O	Number of bytes in buffer.

FUNCTION. . : Expedited data indication.

EXPLANATION : By using this call, you will receive a unit of expedited data whose arrival has been signalled by a TLEVXDIN event.

The first, sufficiently large user buffer provided to TLIB for the connection, following the TLEVXDIN event, will be used for the expedited data. If no such buffer was provided, an error return will be given by TLFEDIN.

USAGE . . . : CALL TLFEDIN(ITLREF,ISTAT,IUBFID,LENDAT)

Routine name : TLFEDRQ

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	IUBUF Integer*2 array	I	User buffer. Must start on a word boundary.
3	LENDAT Integer	I	Number of bytes in user buffer. Must be even.
4	ISTAT Integer	O	Return status.

FUNCTION. . : Expedited data request.

EXPLANATION : To transfer expedited data to the remote user, you have to use this call.

TLIB guarantees that the arrival of the expedited data will be signalled to the remote user, before any ordinary data subsequently will be sent on the connection. The arrival of the expedited data may or may not be signalled to the remote user, before previousely sent ordinary data.

Only one unit of expedited data may be outstanding at a time. You may send no more until you receive a TLEVXDCT event.

USAGE . . . : CALL TLFEDRQ(ITLREF,IUBUF,LENDAT,ISTAT)

Routine name : TLFINIT			
No:	Parameter Name/ Type:	I/O	Explanation:
1	MODE Integer array	I	Specification of TLIB operating mode.
2	IDYNAM Integer array	I	Working storage for TLIB.
3	ISTAT Integer	O	Return status.

FUNCTION. . : Initializes TLIB. This must be the first call on TLIB.

EXPLANATION : The TLIB operating mode is specified by the MODE array (symbols are defined in the TLF:DEFS file):

```
DIMENSION MODE(0:TLALMODE-1)  where TLALMODE = 6.
```

The different positions in the array may be defined by the following symbols:

```
TLMDXMSGMODE = 0
TLMDUNIQUESUFFIX = 1
TLMDMXCONNECTIONS = 3
TLMDMXACCESSPOINTS = 4
TLMDMXBUFFERS = 5
TLMDMXQUEUEDTPDUS = 6
```

The different values that MODE(TLMDXMSGMODE) can have are: TLXMINUSERMODE and TLXMINSYSTEMMODE. Note that TLXMINSYSTEMMODE is very rarely used.

Continued on next page.

MODE(TLMDUNIQUESUFFIX) is set to TLISUNIQUE, if suffix names should be unique to this task. Otherwise it is set to TLNONUNIQUE.

MODE(TLMDMXCONNECTIONS) is the maximum number of connections you want to use in this task. Standard value TLMXCONNECTIONS = 20 may be used.

MODE(TLMDMXACCESSPOINTS) is the maximum number of TLAPs you want to use in this task. Standard value TLMXACCESSPOINTS =3 may be used.

MODE(TLMDMXBUFFERS) is the maximum number of user buffers in your task that TLIB can own at one time. Standard value TLMXBUFFERS = 20 may be used.

MODE(TLMDMXQUEUEDTPDUS) is the maximum number of TLIB TPDUs possible to be queued in XMSG space at one time. Standard value TLMXQUEUEDTPDUS = 50 may be used.

The size of IDYNAM is dependent upon the values described above. You dimension it as follows: DIMENSION IDYNAM(0:TLSZDYNAMIC-1). TLSZDYNAMIC will automatically be calculated for you.

USAGE . . . : CALL TLFINIT(MODE,IDYNAM,ISTAT)

Routine name : TLFPRBF

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	IUBUF Integer*2 array	I	User buffer. Must start on a word boundary.
3	LENUBF Integer	I	Length of user buffer. Must be an even number of bytes.
4	IUBFID Integer	I	Identifier you want to attach to the user buffer.
5	ISTAT Integer	O	Return status.

FUNCTION. . : Provides buffer.

EXPLANATION : A buffer is provided for use by TLIB. Whenever ordinary data arrives, TLIB stores the data in this buffer.

For user data or expedited data, a buffer has to be provided after the event has occured, that signals the incoming data.

To get the buffer back from TLIB, you have to perform a TLIB call. For example, you use the TLFDAIN routine to obtain ordinary data. To obtain user data from a connection request, you use the TLFCNIN routine.

USAGE . . . : CALL TLFPRBF(ITLREF,IUBUF,LENUBF,IUBFID,ISTAT)

Routine name : TLFSTLS

No:	Parameter Name/ Type:	I/O	Explanation:
1	IUSADD Integer*4 array	I	Local TLAP.
2	IUSREF Integer	I	Connection identifier belonging to user.
3	ISTAT Integer	O	Return status.
4	ITLREF Integer	O	Connection identifier assigned by TLIB.

FUNCTION. . : Starts listening.

EXPLANATION : This call tells TLIB that you are willing to receive a connection indication on the specified TLAP. To receive multiple simultaneous connections on a TLAP, TLFSTLS must be invoked the corresponding number of times.

TLIB automatically re-initiates a listening for a new connection indication after the termination of a previous connection. The same ITLREF and IUSREF apply to successive connections.

If you refuse to accept further connection indications, TLFTMLS has to be invoked.

USAGE . . . : CALL TLFSTLS(IUSADD,IUSREF,ISTAT,ITLREF)

Routine name : TLFTMLS

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITLREF Integer	I	Connection identifier belonging to TLIB.
2	ISTAT Integer	O	Return status.

FUNCTION. . : Terminates listening.

EXPLANATION : If you want to tell TLIB that you refuse to accept further connection indications on this connection, you must use this call. In other words, TLFTMLS cancels the effect of the prior TLFSTLS.

USAGE . . . : CALL TLFTMLS(ITLREF,ISTAT)

Routine name : TLFWAIT

No:	Parameter Name/ Type:	I/O	Explanation:
1	ITMOUT Integer array	I	Maximum waiting time for event.
2	IREQEV Integer array	I	Requested event.
3	ISTAT Integer	O	Return status.
4	IACTEV Integer array	O	Actual event.

FUNCTION. . : Waits for event to occur.

EXPLANATION : See also important note on next page.

ITMOUT has the following definition:

```
DIMENSION ITMOUT(0:TLALTIME-1) where TLALTIME = 2.
```

The different positions in the array may be defined by the following symbols:

```
TLTMUNITS = 0,      which gives the time unit.
TLTMLENGTH = 1,     which gives the time length.
```

The different values that ITMOUT(TLTMUNITS) can have are:

```
TLTMBASIC,      which means basic time units.
TLTMSECS,       which means seconds.
TLTMMINS,       which means minutes.
TLTMHRS,        which means hours.
```

ITMOUT(TLTMLENGTH) describes the number of time units.

A value of TLINFINITETIME for ITMOUT(TLTMLENGTH) specifies an infinite timeout period.

A value of 0 for ITMOUT(TLTMLENGTH) is equivalent to a poll for outstanding TLIB events of the requested type, on the requested connection.

Continued on next page.

You may specify that TLIB should wait on a particular connection for a particular set of events. This is done by appropriately setting the desired values in IREQEV(TLEVREFNO) and IREQEV(TLEVCODE). IREQEV(TLEVREFNO) requires a connection identifier belonging to TLIB.

The set of desired events is formed from the logical OR of the corresponding event codes.

The value TLANYREFNO may be used to accept any connection. Likewise, TLANYEVENT may be used to accept any event.

The actual event that has occurred is returned in IACTEV, with IACTEV(TLEVREFNO) set to a user's connection identifier.

USAGE . . . : CALL TLFWAIT(ITMOUT,IREQEV,ISTAT,IACTEV)

IMPORTANT NOTE for those who use the TLEVOTHR event:

A routine TLOEV can be called as a part of TLFWAIT. If you use the TLEVOTHR event, you should provide a routine in your program called TLOEV:

FUNCTION LOGICAL TLOEV

This routine will check all the possible "other" events you are using, e.g. for the presence of terminal input, and return TRUE if any are pending (and FALSE otherwise). TLIB provides a default version of TLOEV which always returns FALSE. If you want to make sure that no events of type TLEVOTHR are missed, you have to provide TLEOV in your program.

The reason that an "other" event may be missed, is that all wakeups from the wait state use one single bit in the program's RT-description (the 5REP bit). If two wakeups, one leading to an "other" event followed by one done by XMSG, occur close together just before TLIB executes the TMOUT monitor call, the information about the "other" event is lost.

A P P E N D I X A

XMSG FUNCTIONS

1 Introduction

This appendix gives a description of how to execute XMSG functions using the SINTRAN monitor call MON 200 (MON XMSG). It should also be of interest to the user that intend to use the XMPFSMC routine (PLANC) or the XMFFSMC routine (FORTRAN).

2 General

XMSG functions are normally executed using the monitor call MON 200 with parameters being passed in the registers. The T register contains the particular function required, with option bits set in its high order byte when required. Note that as a general rule, options, not described under a specific function, should not be set when the function is requested. However, an RT-program that wants to call XMSG as a system task can (and must) specify the XFSYS (system mode) option (see description of 'task' in Chapter 1).

Completion status is returned in the T register as a positive number (its precise meaning depends on the function) if successful, as zero if the operation was not terminated and as a negative number when indicating an error.

We will use the word task to mean a driver, a direct task, or an RT (foreground or background) program. The XMSG system allows tasks to send messages to each other, including handling of memory allocation, queueing, and task synchronization.

A task can open ports through which it can send and receive information about messages. Data is normally transferred between tasks via message buffers within XMSG. The sending task first opens a port, reserves an XMSG message buffer, transfers its data into that buffer and finally informs the receiving task's port that data is awaiting to be collected. Reservation and releasing of messages are done explicitly by the task.

A system is a Processing Unit that runs an independent XMSG kernel. An ND-100 CPU is a system, but a PIOC CPU or an ND-500 CPU is not. These are respectively seen as part of an ND-100, since every PIOC or ND-500 task which uses XMSG has a 'shadow' task in the ND-100.

The functions are divided into two groups: user functions (of general interest) and system functions (used mainly by XROUT and the XMSG-COMMAND program). The functions in each group are described in the corresponding sections below: 'User Function Specifications' and 'System Function Specifications'.

XROUT services are invoked by sending messages (using functions) to the task called XROUT. The services, and the method for accessing them, are described in appendix B.

Note that all functions, services, error codes and message types are referred to symbolically. Their values are defined in the file XMP:DEFS (or XMF:DEFS).

Note that in many of the functions which follow, there is currently no parameter returned in the A, D or X registers. However, you must be aware that this may change in the future. Thus you cannot assume that the A, D and X registers are preserved from a call of any of these functions.

In the following descriptions these symbols will be used in the parameter lists (integer unless otherwise specified):

ISTAT - result status.

XFxxx - function code or option (options are given in parentheses).

NBYTES - number of bytes.

METYP - message type.

MESAD - message identifier.

UADD - user buffer address.

ULEN - length of user data in bytes.

DISP - displacement within message in bytes.

NMESS - Number of messages.

PORTNO - local port identifier. If zero, the most recently opened port (i.e., the default port) is assumed.

RPORT - A value that is almost unique for each remote port.

MAGNO - double word (32 bits) containing remote port identifier.

QLEN - number of messages currently queued for a port.

DATA0 - first two bytes of user data.

B0to3 - Bytes 0 to 3.

B4to5 - Bytes 4 to 5.

BANKNO - Bank number.

PDISP - Address within a bank.

PHYSAD - Physical address (24 bits).

XPASW - Password (version code).

CONFI - Configuration mask.

RCOUNT - Restart count since last 'warm start'.

NCALLS - Number of functions requested.

SYSNO - System number.

INDEX - RT-index.

SPRIV - Task information.

AINFO - Additional information.

BASEAD - XMSG base field address (B register on level 5).

ILEV - Interrupt level.

XTADDR - Task block (XT-block) address.

DBREG - Driver's B register.

DREST - Driver's restart address

The calls will be described by showing the NPL code required to use them. Please remember that the T register always contains the status on return and should be checked.

3 User Function Specifications

Some functions and services are privileged. Before calling these, a task must be defined as privileged by invoking the XFPRV function described later.

3.1 Manipulating Ports

When a task opens ports they are identified locally with a port number (like a file number). A task identifies other tasks' ports using a 32 bit magic number (MAGNO) which consists of the port number, the system number and a random part. The latter guarantees that a port which has been closed and then reopened does not have the same identifier.

XMSG allocates a port list to your task. This is done so that XMSG can administrate all your opened ports in an efficient manner. When you open a port, XMSG inserts that port on top of the port list. When you close a port, XMSG takes that port out from the list. If you specify a port number parameter equal to zero in most of the XMSG functions, XMSG will use the first port in the port list. This port is referred to as the most recently opened port (or the default port).

3.1.1 Open a Port (XFOPN)

```
T:=XFOPN                    % T=function
*MON XMSG
T=:ISTAT                    % T=result status
A=:PORTNO                   % A=port number assigned by XMSG
```

A port is opened and its number (i.e., the port identifier) is returned to the calling task.

The opened port becomes the task's default port. When this port is closed, the previously opened port, if any, becomes the task's default port.

3.1.2 Close Ports (XFCLS)

```
T:=XFCLS                    % T=function
A:=PORTNO                   % A=number of port to be closed
*MON XMSG
T=:ISTAT                    % T=result status
```

The specified local port is closed. If A<0, all ports owned by the calling task will be closed. If A=0, the most port recently opened (i.e., the default port) will be closed.

When a port is closed, all 'non-secure' messages currently queued for that port are released, while all 'secure' messages (as well as the 'port current' message, if any) are set 'non-secure' and returned to the sender. If the port had a name, the name is cleared (i.e., the name is removed from XROUT's name table).

See also disconnect function (XFDCT) for closing ports.

3.1.3 Port Status (XFPST)

```
T:=XFPST (BONE XFWTF/XFWAK/XFHIP) % T=function BONE options
A:=PORTNO                   % A=port number to be checked
*MON XMSG
T=:METYP                    % T=message type or result status <=0
A=:RPORT                    % A=hashed magic number of remote port
A:=D=:MESAD                 % D=message identifier
X=:QLEN                     % X=queue length
```

If the specified port number in A is zero, the most recently opened port (i.e., the default port) is assumed.

On return, the T register indicates the message type (see below) of the first message in the queue (or 0, if there are none). If a message is waiting, D contains its address and A a value that is usually unique for the remote port, so it can be used for a quick check that

the message has come from a known partner. The X register always contains the queue length, i.e., number of messages queued for the port.

Note that calling this function, when a message is waiting on portNo, will lead to both the 'general wake up' bit for the task and the 'wake up' bit on portNo. If no message is waiting XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on portNo, before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XFGST function. The 'wake up' bit on portNo may have been set as a result from a previously executed XFPST, XFRCV, XFRRH or XFRRE function.)

Message types:

XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent as secure, but could not be delivered)

Options:

If no message is waiting on portNo and the XFWTF (wait flag) is set, the task is suspended until the next message arrives on portNo. If the XFWTF is not set, a zero status is returned. Then a 'wake up' bit will be set on portNo, if XFWAK (wake up) is set.

When 'wake up' has been set on portNo, the next transmission to that port will lead to a wake up of the receiver task and the 'wake up' bit on portNo is cleared. This allows timed-out waits to be executed.

The XFWAK option can be enabled on more than one port at a time. Be aware, however, that if the task is in XMSG wait (for example, by sending a secure message with wait) when the task should have been woken up as a result of a message being sent to portNo. The 'wake up' bit will be cleared, but the task will not (and cannot) be woken up.

The XFHIP (high priority message) option allows a task to check the arrival of high priority messages. If the XFHIP option is set and a high priority message is waiting, the message type XMTHI is returned in the T register. If no high priority message is waiting on portNo and if XFHIP is set, and XFWTF is not, a zero status is returned in the T register. When the next message of any type arrives, the task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

3.1.4 General Status (XFGST)

A task may have many open ports, and it may be unpredictable to which one the next message will arrive. This function allows the task to check all ports, i.e., it allows the task to find out whether any messages have been received on any port.

```
T:=XFGST (BONE XFWTF/XFWAK/XFHIP) % T=function BONE options
A:=PORTNO                         % A=last port to be scanned
*MON XMSG
T=:ISTAT                          % T=result status
A=:PORTNO                         % A=port no. where message is
                                    waiting
```

The call parameter portNo specifies the last port to be searched. If portNo is zero, this implies the most recently opened port (i.e., the default port). Note that the search will begin with the next port (if any) after the one specified, and then follow the task's port list (see example below).

On return, the A register contains the port number where the message, if any, is waiting.

For example, if the task has opened four ports and have got the port numbers 15 (from the 1st XFOPN), 4 (from the 2nd XFOPN), 6 (from the 3rd XFOPN) and 19 (from the 4th XFOPN), then the port list comprises the ports 19-6-4-15 (in that order!). Port number 19 (the first port in the list) is the task's default port (i.e., if portNo is zero, this port is assumed). If the task has just handled a message received on port 6, it can, when it wants to have a 'round-robin' scheduling of requests, call XFGST with portNo=6. Port 6 will then be the last port to be looked at by XMSG. XMSG will start looking at port 4 to see if a message is waiting. If no message is waiting on port 4, XMSG will look at port 15. Further if no message is waiting on port 15, XMSG will look at port 19. If no message is waiting on port 19, XMSG will finally look at port 6.

Note that calling this function, when a message is waiting on one of the ports, will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on the returned portNo. If no message is waiting on any of the ports, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on all ports opened by the task, before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XFGST function. The 'wake up' bit on the individual ports may have been set as a result from previously executed XFPST, XFRCV, XFRRH or XFRRE functions.)

Options:

If no messages are waiting on any of the ports and the XFWTF (general wait flag) is set, the task is suspended until the next message arrives on one of the ports. If, instead, the XFWTF is not set, a zero status is returned. Then a general wake up' bit will be set for this task, if XFWAK (general wake up) is set.

When 'general wake up' is set, the next transmission to one of the ports will lead to a wake up of the receiver task, and clearing of the 'general wake up' bit for that task.

However, be aware that if the task is in XMSG wait (for example, sending a secure message with wait) when the task should have been woken up as a result of a message being sent to one of its ports, the 'general wake up' bit will be cleared, but the task will not (and cannot) be woken up.

The XFHIP (high priority message) option allows a task to check the arrival of high priority messages. If a high priority message is waiting on one of the ports and XFHIP is set, the port number where the high priority message is waiting, is returned in the A register. If no high priority message is waiting on any of the ports and XFHIP is set. While XFWTF is not set, a zero status is returned in the T register. When the next message of any type is sent to a port opened by this task, the task will be woken up (i.e., if no high priority message is waiting, XFHIP has the same effect as XFWAK).

3.1.5 Disconnect (XFDCT)

```
T:=XFDCT                    % T=function
*MON XMSG
T=:ISTAT                    % T=result status
```

This function will release all XMSG resources reserved by the task. All the ports opened by the task are closed. All XMSG space belonging to the current caller is released. Special action is taken in the case of current messages, and messages waiting on the input queue (see XFSND, XFRCV and XFCLS).

There is no return from driver calls, to XFDCT (as the driver context is released by the call).

Note that the following automatic disconnection are executed by SINTRAN:

User task disconnection:
- On returning to the SINTRAN command processor (@)
- On logout/RT program termination

System task disconnection:
- On logout/RT program termination

3.2 Manipulating Message Buffers

Message buffers are simply variable length areas which can be reserved. When assigned to a task, they remain reserved until that task decides to release them or 'send' them to another task. By this transfer, point ownership is transferred to the receiving task thus being enable to read the data. Having read the data, the receiving task may either release the buffers back to the pool or use them

itself for storing a message to be sent back to the first or any other task.

Note that in many of the functions which follow, there is no parameter required to specify the message identifier (MESAD). The reason is, that a current (default) message buffer is assumed, namely the last message received on the appropriate port, or, if none, the last port operated on by the task. Sending or releasing a message leads to its currency being lost. The task may also change the value of the Current Message with the XFSCM function. A MESAD value of -1 implies the current message.

Messages cannot be released, read from or written to by tasks other than the current owner or while queued to a port. In the latter case the message must be received first.

3.2.1 Reserving Message Buffer (XFGET)

```
T:=XFGET (BONE XFWTF/XFEXC) % T=function BONE options
A:=NBYTES                   % A=number of bytes requested
*MON XMSG
T=:ISTAT                    % T=result status
A=:MESAD                    % A=message identifier
```

MESAD is returned to the caller for possible use in subsequent functions. The message buffer consists of a descriptor of the current owner, sender, size, length etc., and a buffer for user data. The buffer size has a maximum size, which is system dependent and defined when the XMSG system is generated.

At any particular time, the total space owned by a task cannot exceed another limit, which is initially set to a value defined at XMSG generation time. It can, however, be changed by privileged tasks using the Define Maximum Memory (XFDMM) function, see page 280.

Only the current owner of a message is allowed to read or write in it, give it to someone else or release it.

Specifying a buffer size of 0 bytes implies, that only a message descriptor will be reserved. Privileged tasks can then associate a physical memory area with that message descriptor by using the Define User Buffer (XFDUB) function described below. It is not allowed to send a buffer of size 0 out of the system.

If no message buffer of the requested size is available and XFWTF (wait flag) is set, the task will be suspended until a message of that size is available.

If the XFEXC (exclusive buffer) flag is set, it implies, that the caller wants to reserve an exclusive message buffer, which has been allocated using the Allocate Message Buffers (XFALM) function with option XFEXC. If no such allocated message buffer is available, an error code is returned in the T register.

3.2.2 Defining a User Buffer (XFDUB)

This is a privileged function (see XFPRV) that allows a user to associate a physical memory buffer with a message descriptor previously obtained by XFGET with NBYTES=0. All XMSG functions whill then operate on that message as though the buffer space was part of the general XMSG buffer. An exception is that XFREL (described later) only releases the message descriptor and not the buffer area.

This allows special systems or drivers to fully control their memory allocation procedures.

```
T:=XFDUB                    % T=function
AD:=PHYSAD                  % AD=address of physical memory buffer
X:=NBYTES                   % X=buffer size in bytes
*MON XMSG
T=:ISTAT                    % T=result status
```

The function acts on the 'task current' message. PHYSAD is the physical (24 bit) address of the start of the buffer. NBYTES is its size in bytes.

Buffers been defined in this way cannot be sent to other systems.

3.2.3 Releasing Message Buffer (XFREL)

This function is used to release a message buffer reserved by the task. A message buffer is reserved by the task when the task issues the 'Reserve Message Buffer' (XFGET) function when a message is sent to it from another task. Recension is also made,but then the message must be received before it can be released.

At any particular time, the total message buffer space owned by a task cannot exceed a limit defined when the XMSG system is generated. Thus it is a general rule for a task to release its message buffer as soon as it is through with it.

```
T:=XFREL                    % T=function
A:=MESAD                    % A=message identifier
*MON XMSG
T=:ISTAT                    % T=result status
```

A message identifier of -1 in the A register, will release the 'task current' message.

If the specified message is an allocated message (i.e., a message allocated using the 'Allocating Message Buffers (XFALM) function), the message will be put back on the original task's 'Available Allocated Message List' (AAML), see function XFALM.

3.2.4 Allocating Message Buffers (XFALM)

Normal messages buffers that have been reserved using the XFGET function lose their association with the task that initially reserved them, when they are sent to another task. This implies that the sending task has no guarantee that it will be able to reserve space later.

By allocating message buffers, a task can indicate to XMSG its long-time buffer requirements. Allocated messages are removed from the free space pool and marked as allocated to the original caller. They do change owners when sent within a system (as do normal message buffers). When they are released, however, or sent out of the local system, the message buffer is put back on the originating task's 'Available Allocated Message List (AAML)'.

All allocated messages for a given task must be of the same size. When an XFGET is executed by that task for a buffer of that size, XMSG will first look at the task's AAML and take a message buffer from it, if one is available. Similarly, when a message of that size comes into the system from another system, XMSG will first look at the AAML of the receiving task and take a message buffer from it, if one is available.

Messages can be allocated to the calling task by:

```
T:=XFALM  (BONE XFEXC)      % T=function BONE option
A:=NBYTES                   % A=message size in bytes
X:=NMESS                    % X=no. of messages to allocate
*MON XMSG
T=:ISTAT                    % T=result status
```

Note that if the function fails, due to lack of buffer space, no messages are allocated.

If XFEXC (exclusive buffers) is set, the messages are allocated and set aside for exclusive use by the task, i.e., these message buffers will not be used by XMSG when a message of NBYTES bytes is received from another system. Buffers allocated with XFEXC do not have to be of the same size as buffers allocated without this option set. However, all exclusive buffers must be of the same size. To reserve one of these exclusive buffers, the task must execute the XFGET function with the option XFEXC set, see page 266.

3.2.5 Freeing Allocated Message Buffers (XFFRM)

Allocated messages can be freed by:

```
T:=XFFRM (BONE XFEXC)        % T=function
X:=NMESS                     % X=no. of allocated messages to free
*MON XMSG
T=:ISTAT                     % T=result status
A=:NMESS                     % A=no. of allocated messages freed
```

If XFEXC (exclusive buffers) is set, only those message buffers which are previousely allocated with the XFEXC option set, will be freed. If XFEXC is not set, only those message buffers which are previousely allocated without the XFEXC option will be freed. On return, the A register contains the number of allocated messages actually freed.

3.2.6 Writing into Message Buffers (XFWRI)

After building up a data buffer in its own space, a task transfers the data buffer into the 'task current' message buffer as follows:

```
T:=XFWRI (BONE XFRES)        % T=function BONE option
NBYTES=:D                    % D=number of bytes to write
A:=UADD                      % A=address of user buffer
X:=DISP                      % X=displacement within message in bytes
*MON XMSG
T=:ISTAT                     % T=result status
A:=D=:NBYTES                 % D=number of bytes actually written
```

If the 'whole-message-read' flag has been set (see XFREA), it is cleared, and the current message length (not the same as size) is set to 0. If DISP is -1, a value for DISP, equal to the current message displacement, is assumed instead, thus providing an appending function. If the displacement (DISP) is odd, 1 is added to it, and a zero byte inserted in the message. If DISP+NBYTES is greater than the message size, an error return occurs. Otherwise NBYTES bytes are copied from UADD into the message buffer, and the current message displacement is set to DISP+NBYTES (where DISP has been rounded up, if odd). If this copying resulted in the message being longer than before, the current message length is also set to DISP+NBYTES. NBYTES is returned to indicate the actual number of bytes transferred.

Note that the displacement (DISP) is always rounded up to the next even byte before the data is written.

If you have access to the buffer area directly (either because it was defined using the XFDUB function or because you have access to the physical memory), you can, of course, read and write yourself. You must then however, be aware that the 'current displacement' and 'current length' information in the message descriptor will not be updated.

If the option XFRES (reset message length) is set, it causes the current message length being set to 0, before the user data is transferred into the current message (in fact, it acts as if the 'whole-message-read' flag had been set).

3.2.7 Writing only the Header of a Message Buffer (XFWHD)

This function will write six bytes into the 'task current' message.

```
T:=XFWHD                    % T=function
AD:=B0to3                   % AD=bytes 0 to 3 of message header
X:=B4to5                    % X=bytes 4 to 5 of message header
*MON XMSG
T=:ISTAT                    % T=result status
```

If the 'whole-message-read' flag has been set (see XFREA), it is cleared and the current message length (not the same as size) is set to 0. Then the function inserts the A, D and X registers as the first six bytes of the message. If this results in the message being longer than before, the current message length is set to 6. It then sets the current message displacement to 6.

If the message size is less than 6 bytes, an error return occurs.

3.2.8 Reading from a Message Buffer (XFREA)

```
T:=XFREA                    % T=function
NBYTES=:D                   % D=number of bytes to read
A:=UADD                     % A=address of user buffer
X:=DISP                     % X=displacement within message in bytes
*MON XMSG
T=:ISTAT                    % T=result status
A:=D=:NBYTES                % D=number of bytes actually read
```

The data is read from the 'task current' message, starting with displacement DISP (rounded up to the next even byte) into the user buffer specified by UADD (length NBYTES). On return, NBYTES is set to the actual number of bytes read. If DISP is -1, the reading of the message is resumed from the current message displacement. Note that the displacement (DISP) is always rounded up to the next even byte before the data is read.

On return, if the last byte in the message is read, the current message displacement is set to 0. Also the 'whole-message-read' flag is set, so that the next XFWRI or XFWHD will reset the current message length to zero. Otherwise, except when NBYTES is zero, the current message displacement is set to DISP+NBYTES. If NBYTES is zero, the current message displacement is not updated.

3.2.9 Reading only the Header of a Message Buffer (XFRHD)

The first six user bytes of a message can be read using:

```
T:=XFRHD                    % T=function
A:=MESAD                    % A=message identifier
*MON XMSG
T=:ISTAT                    % T=result status
AD=:B0to3                   % AD=bytes 0 to 3 of message header
X=:B4to5                    % X=bytes 4 to 5 of message header
```

The first six bytes of the message are returned in the A, D and X registers (in that order!), and the current message displacement is set to 6. If MESAD is not -1, the specified message becomes the 'task current' message.

If the message size is less than 6 bytes, an error return occurs.

3.2.10 Define Bank Number for Drivers (XFDBK)

When calling functions which transfer data between a user area and an XMSG buffer (e.g., XFREA, XFWRI or XFSMC), drivers specify a physical address as user address (UADD). This is in bank 0, unless they have previously defined a bank number using the XFDBK function:

```
T:=XFDBK                    % T=function
A:=BANKNO                   % A=bank number
*MON XMSG
T=:ISTAT                    % T=result status
```

This function is only allowed for drivers (i.e., not allowed from RT-programs).

3.2.11 Sending Message (XFSND)

When a task wants to 'send' a message to another task, it must know the 'address' or MAGNO of a port of the task. Since port numbers (and hence MAGNOs) are allocated by XMSG when the port is opened, the destination MAGNO must be obtained by a task via XROUT.

The task establishes initial contact by sending a message to a dedicated task named XROUT (see appendix B), to name its port(s). Subsequently a second task may send a 'letter' via one of its ports also to XROUT, specifying a destination port by name (see XROUT Letter Service XSLET). If this name has been previously declared, XROUT will forward the message to the named port.

The first task can then use the XFMST function (described later) to extract the MAGNO of the second task. Hence a direct dialogue can begin. (Note that only ports expecting letters need to have names. These will usually be ports providing services - 'server ports'.)

By convention, all names of ND standard products and of ND standard systems will start by *xx, where xx are specific to the product.

In this section it is assumed, that the sender now knows the destination magic number (MAGNO).

A Message Buffer is 'transferred' from one task to another in this way:

```
T:=XFSND (BONE options)      % T=function BONE options (see below)
AD:=MAGNO                    % AD=magic number of receiving port
X:=PORTNO                    % X=number of sending port
*MON XMSG
T=:ISTAT                     % T=result status
```

A magic number parameter of -1 (in both A and D) will direct the message back to the port from which it was last sent.

The call parameter portNo specifies the port from which the message will be sent. If portNo is zero, the most recently opened port (i.e. the default port) is assumed.

Note that there is no parameter specifying the message that is to be sent, for the reason that the current (default) message buffer is assumed, namely the 'port current' message if one exists, or, if none, the 'task current' message.

The options are:

XFWTF - Wait flag. This is only significant when a secure message (XFSEC) is sent to another system (see below).

If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port).

If not set, secure messages that cannot be delivered, will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered, or if the receiving port is closed (eg., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFFWD - Forwarding message. The sender information in the message will not be updated. Thus, it will appear to the receiver that the message was sent directly from the previous sending port.

XFROU - Route message. Ignore the MAGNO parameter and send the message to the local routing task (XROUT). The message contents should be parameters to XROUT. (See appendix B on XROUT services.)

XFRRO - Remote route message. If the XFROU bit is also set, then the message is sent to a remote routing task (XROUT). The A register is assumed to contain the system number to which the message will be sent. The message contents should be parameters to XROUT.

Note that if the XFROU bit is not set and XFRRO is set, the message will be sent as if XFHIP had been set (i.e., when XFROU is not set, setting the XFRRO bit will act as if the XFHIP bit had been set instead).

XFHIP - High priority message. If the XFROU bit is not set, the message will be chained to the head of the receiver's queue, following any other high priority messages already queued.

Note that if both the XFROU bit and the XFHIP are set, the message will be sent as if XFROU and XFRRO had been set (i.e., when XFROU is set, setting the XFHIP bit will act as if the XFRRO bit had been set instead).

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., XFRCV, XFRRH or XFRRE), which would have led to this message being received, it will instead be returned to the sender.

3.2.12 Returning a Message (XFRTN)

You will often need to write a return status into a message and send it back to the port from which it came (e.g., replying to a transaction):

```
T:=XFRTN (BONE options)      % T=function BONE options (see below)
DATA0=:D                     % D=first two bytes of message header
A:=MESAD                     % A=message identifier
X:=PORTNO                    % X=number of sending port
*MON XMSG
T=:ISTAT                     % T=result status
```

This leads to MESAD being set as the 'task current' message and the 'port current' message for portNo, DATA0 being written into the first two bytes of the message buffer. Then the message buffer (MESAD) will be returned to the port from which it was last sent.

If MESAD is -1, the current message is assumed, namely the 'port current' message if one exists, or, if none, the 'task current, message.

The call parameter portNo specifies the port from which the message will be sent. If portNo is zero, the most recently opened port (i.e. the default port) is assumed.

The options are:

XFWTF - Wait flag. This is only significant when a secure message (XFSEC) is sent to another system (see below). If set, it implies that the caller will only be restarted (with proper status) when the message has been put into the receiver's input queue (i.e., the sending task is suspended until the message has been sent to the remote port). If not set, secure messages that cannot be delivered, will be returned to the sending port.

XFSEC - Secure message. The message will be returned to the sending port if it cannot be delivered, or if the receiving port is closed (eg., if the receiving task terminates) while the message is 'port current'. Non-secure messages are discarded and released by XMSG if they cannot be delivered.

XFHIP - High priority message. It will be chained to the head of the receiver's queue, following any other high priority messages already queued.

XFFWD - Forwarding message. The sender information in the message will not be updated. Thus, it will appear to the receiver that the message was sent directly from the previous sending port.

XFBNC - Bounce message. When the receiver issues 'Receive Message' (i.e., XFRCV, XFRRH or XFRRE), which would have led to this message being received, it will instead be returned to the sender.

(In fact the function is equal to XFSND, except that the D register contains two bytes of data and the A register the message address.)

3.2.13 Receiving Next Message (XFRCV)

When a task is ready to handle the next request, it calls XFRCV:

```
T:=XFRCV (BONE XFWTF/XFWAK) % T=function BONE options
A:=PORTNO                   % A=number of receiving port
*MON XMSG
T=:METYP                    % T=message type or result status <=0
A=:RPORT                    % A=hashed magic no. of remote port
A:=D=:MESAD                 % D=message identifier
X=:NBYTES                   % X=message length in bytes
```

If portNo is zero, the most recently opened port (i.e., the default port) is assumed.

If a message is waiting on the specified port, it is received (unchained from the message queue) and its address returned in the D register. The A register contains a value that is usually unique for the remote port. Thus, it enables a quick check that the message has come from a known partner. X contains the message length in bytes, and T the message type (see below).

Note that calling this function when a message is waiting on portNo, will lead to the clearing of both the 'general wake up' bit for the task and the 'wake up' bit on portNo. If no message is waiting on portNo, XMSG will clear both the 'general wake up' bit for the task and the 'wake up' bit on portNo, before checking the requested option(s). (The 'general wake up' bit may have been set as a result from a previously executed XFGST function, and the 'wake up' bit on portNo may have been set as a result from a previously executed XFPST, XFRCV, XFRRH or XFRRE function.)

Message types:

XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent as secure, but could not be delivered)

If the message type (returned in T register) is 'returned' message (XMTRE), the X register contains the reason for the return (error code <0).

A successful XFRCV will qualify the received message as the 'task current' message. In addition, if it is a secure message, it becomes the 'port current' message for the receiving port. If the message is 'secure', the task aborts or the port is closed while the message is the 'port current' message, the message will be returned to the sender with 'return' status.

The 'task current' message is cleared by releasing/sending it to someone else, or receiving another message. The 'port current' message is cleared by releasing/sending it to someone else, or receiving another secure message. A task may also change the value of the current message by the XFSCM function.

Options:

If no message is waiting on portNo and XFWTF (wait flag) is set, the task is suspended until the next message arrives on portNo. If, instead, the XFWTF is not set, a zero status is returned. If XFWAK (wake up) is set, a 'wake up' bit will be set on portNo. The XFWAK option can be enabled on more than one port at a time.

When 'wake up' is set on portNo, the next transmission to that port will lead to a wake up of the receiver task, and clearing of the 'wake up' bit on portNo. This allows timed-out waits to be executed.

When the wake up is done, the message is not received. Hence the receive must be repeated.

However, be aware that if the task is in XMSG wait (for example, sending a secure message with wait) when the task should have been woken up as a result of a message being sent to portNo, the 'wake up' bit will be cleared, but the task will not (and cannot) be woken up.

3.2.14 Receive and Read Header (XFRRH)

As an alternative to receive, a task may call the XFRRH function, which receives the next message in the queue (as XFRCV), and also (additional to XFRCV) reads the first two bytes of the message.

```
T:=XFRRH (BONE XFWTF/XFWAK) % T=function BONE options
A:=PORTNO                   % A=number of receiving port
*MON XMSG
T=:METYP                    % T=message type or result status <=0
A=:RPORT                    % A=hashed magic number of remote port
A:=D=:MESAD                 % D=message identifier
X=:DATA0                    % X=first two bytes in message
```

If portNo is zero, the most recently opened port (i.e., the default port) is assumed. The options are the same as for the receive function (XFRCV). When the message is received, both the 'task current' message and the 'port current' message will be set as described under XFRCV.

As mentioned this function returns the first two bytes of user data instead of the message length. If the length or size of the received message is less than two bytes, two random bytes will be returned in the X register.

3.2.15 Receive and Read Message (XFRRE)

As an alternative to receive, a task may also call the XFRRE function, which receives the next message in the queue (as XFRCV), and also (additional to XFRCV) reads data from the current (received) message. When the message is received, data will be read from the first byte in the message buffer into the user buffer. If the last byte in the message is read, the current message displacement is set to 0, and the 'whole-message-read' flag is set. Thus the next 'write message' function will reset the current message length to zero. Otherwise, if the last byte is not read, the current message displacement is set to the actual number of bytes read.

```
T:=XFRRE (BONE XFWTF/XFWAK)  % T=function BONE options
NBYTES=:D                    % D=number of bytes to read
A:=PORTNO                    % A=number of receiving port
X:=UADD                      % X=address of user buffer
*MON XMSG
T=:METYP                     % T=message type or result status <=0
A=:RPORT                     % A=hashed magic no. of remote port
A:=D=:MESAD                  % D=message identifier
X=:NBYTES                    % X=message length in bytes
```

If portNo is zero, the most recently opened port (i.e., the default port) is assumed. The options are the same as in the receive function (XFRCV). If the number of bytes to read (NBYTES) exeeds the message length, only the number of bytes specified by the message length will be read into the user buffer.

Note that when the message is received, both the 'task current' and the 'port current' message will be set as described under function XFRCV. Note also that the output parameters are identical to the output parameters of the receive function (XFRCV).

3.2.16 List Messages and Ports (XFLMP)

This function allows a task to get information about its own open ports and its own messages.

```
T:=XFLMP                     % T=function
A:=MESAD                     % A=message identifier or 0
X:=PORTNO                    % X=port number or 0
*MON XMSG
T=:ISTAT                     % T=result status
A=:MESAD                     % A=message identifier
A:=D=:NBYTES                 % D=message size in bytes
X=:PORTNO                    % X=port number
```

On return, the A register contains the message identifier for the first message found equal to or greater than that requested, or 0 if not higher. D contains the message size in bytes, i.e., number of bytes obtained when the message was reserved, either by the Get Message Buffer (XFGET) function, or received from another task, allocated by the Allocate Message Buffers (XFALM) function. X contains

the port number of the first port found, with a value equal to or greater than that requested or 0 if not higher.

3.2.17 Message Status (XFMST)

XFMST allows a task to extract the sender's magic number, and get the length and type of a received message.

```
T:=XFMST                    % T=function
A:=MESAD                    % A=message identifier
*MON XMSG
T=:METYP                    % T=message type or result status <=0
AD=:MAGNO                   % AD=magic number of sending port
X=:NBYTES                   % X=message length in bytes
```

If MESAD is not -1, the specified message becomes the 'task current' message.

The message type is returned in the T register.

Message types:

XMTNO - Normal message

XMROU - Message last sent by XROUT (routing program)

XMTHI - High priority message (sent with XFHIP option)

XMTRE - Returned message (sent as secure, but could not be delivered)

It might be expected that this requires an extra call, but:

a) - you often just send a message back to its sender (XFSND with MAGNO=-1 or XFRTN), and

b) - you can read the magic number once, hence that use the system and port information (RPORT), returned by XFRCV, to identify the sender, whose MAGNO you now have.

3.2.18 Set Current Message (XFSCM)

Since many functions implicitly operate on the current message, it is useful to be able to set the latter:

```
T:=XFSCM                    % T=function
PORTNO=:D                   % D=port number
A:=MESAD                    % A=message identifier
*MON XMSG
T=:ISTAT                    % T=result status
```

The specified message is set as the 'task current' message. If the port number is >=0, the message is also set as the 'port current' message for the specified port. If the port number is zero, the most recently opened port (i.e., the default port) is assumed.

4 Miscellaneous Functions

4.1 Dummy function (XFDUM)

The effect of this function is merely to return the current configuration. It is also useful for benchmarking.

```
T:=XFDUM                    % T=function
*MON XMSG
T=:ISTAT                    % T=result status
A=:XPASW                    % A=XMSG password (version code)
A:=D=:CONFI                 % D=configuration mask
X=:RCOUNT                   % X=XMSG restart count
```

On return, the A register contains the password which is needed in order to become a privileged XMSG task (see function XFPRV). The X register returns the number of times that XMSG has been (re)started since the last warm start.

The bits currently defined in the configuration mask (D register) are:

```
bit 0 : set if inter-system XMSG
    1 :  "  "  generated with tracing
    2 :  "  "  generated for ND-100
    3 :  "  "  file server for file transfer is included
    4 : is not used
    5 : set if running on page table 3
    6 :  "  "  generated for ND-100/CX instruction set
    7 :  "  "  generated with gateway software for network servers
```

Note that this bit mask, which is based on XMSG version J, will most certainly be extended in later XMSG versions.

4.2 Start Multi-Call (XFSMC)

This function allows a task to execute a set of XMSG functions, issuing only one XMSG monitor call. This eliminates the overhead associated with each XMSG monitor call.

```
T:=XFSMC          % T=function
X:=NCALLS         % X=number of functions requested
A:=UADD           % A=address of buffer containing the parameters
*MON XMSG
                  % TADX are parameters from last executed function
```

NCALLS is the number of functions to be executed. UADD is the address of a buffer containing the parameters for these functions. Each set of parameters comprise 4 words (T, A, D and X registers), so the buffer length should be 8*NCALLS bytes long. NCALLS has a maximum size system dependent, and defined when the XMSG system is generated. If NCALLS is 0 (or -1), then the previously request for multi-calls will be re-executed.

XFSMC returns as soon as a function, with status less than or equal to zero, terminates (or when all the functions have been executed), and the return registers (T, A, D and X) are set according to the return parameters from the last function executed.

You should be aware of the fact that if the XFDCT function is specified (and executed) as one of the functions in the multi-call, the succeeding functions in the multi-call will not be executed, as the task context (XT-block) is released by the disconnect (XFDCT) function.

4.3 Define Maximum Memory (XFDMM)

When a new task is defined in XMSG, its maximum memory usage is set to a predefined value (a size system dependent and defined when the XMSG system is generated). This can be changed for privileged tasks by using XFDMM.

```
T:=XFDMM          % T=function
A:=NBYTES         % A=requested task space in bytes
*MON XMSG
T=:ISTAT          % T=result status
```

NBYTES will be set as the maximum number of bytes that can be owned by the task as message space at one time.

4.4 Convert Magic Number to Port and System Number(XFM2P)

This function allows a task to convert the magic number to a port number and a system number.

```
T:=XFM2P                    % T=function
AD:=MAGNO                   % AD=magic number
*MON XMSG
T=:ISTAT                    % T=result status
A=:PORTNO                   % A=port number
A:=D=:SYSNO                 % D=system number
X=:INDEX                    % X=RT-index or -1/-2
```

Note that the returned T register may contain additional information about the specified magic number. If the magic number was that of a system, then T=3. If the magic number was that of a local port, and the task of the port owner is a privileged task, then T=2. If the magic number was that of a remote port, or that of a local port, and the task of the port owner is unprivileged, then T=1.

The returned X register may contain additional information about the port owner task. If the magic number was that of a system, or that of a remote port, then X=-1. If the magic number was that of a local port and the task of the port owner is a driver, then X=-2. If the magic number was that of a local port and the task of the port owner is an RT- program, then X equals SINTRAN's RT-index of the RT-program.

4.5 Convert Port Number to Magic Number (XFP2M)

This function allows a task to convert a local port number to a magic number. Any task may obtain the magic number of its own ports. Privileged tasks can obtain the magic number of a port owned by another local task.

```
T:=XFP2M                    % T=function
A:=PORTNO                   % A=port number
*MON XMSG
T=:ISTAT                    % T=result status
AD=:MAGNO                   % AD=magic number for port
```

Note that this function will only return the magic number of ports opened by tasks in the local system.

4.6 Define Wake Up Context (XFWDF)

If a driver uses the XFWAK option, XMSG must be informed about where to start the driver. This is done by using this function.

```
T:=XFWDF                    % T=function
DBREG=:B                    % B=driver's B register on restart
A:=DREST                    % A=driver's restart address (P reg)
*MON XMSG
T=:ISTAT                    % T=result status
```

This function is only allowed for drivers (i.e., not allowed from RT-programs).

4.7 Check System and User Privileges (XFCPV)

This function allows a task, when a message has been received, to check the privileges of the sender.

```
T:=XFCPV                    % T=function
A:=MESAD                    % A=message identifier
*MON XMSG
T=:ISTAT                    % T=result status
A=:SPRIV                    % A=access information
A:=D=:AINFO                 % D=additional information
```

If MESAD is not -1, the specified message becomes the 'task current' message.

If the sending task is allowed to update the routing tables on this system, (i.e., execute the privileged XROUT services XSDRN and XSDSY) then A=1. If the message is sent from a task within the local system, then D=0. If the message is sent from a task in another system, then D=1.

If the sending task is not allowed to update the routing tables, then A=0 and D contains the reason:

- D=0 implies that the sending task, as well as the source system are unprivileged.
- D=1 implies that the source system is privileged, but the sending task is not.
- D=2 implies that the sending task is privileged, but the source system is not.
- D=3 if the specified message is a returned (non-delivery) message.

(An unprivileged task is a task which has not (yet) successfully executed the XFPRV function. An unprivileged system is a remote system which has not (yet) been defined as a friend to your system, see XROUT service XSDAT.)

4.8 Make Calling Task Privileged (XFPRV)

Most system functions, as well as some user functions (e.g., Define Maximum Memory - XFDMM) can only be executed by privileged tasks. In order to become privileged (for XMSG), a task must successfully execute the XFPRV function.

In order to do this the caller must be either a driver, direct task, foreground program or background program, logged in as user SYSTEM. Besides this, the program must also specify the current XMSG password in the A register. The password (XPASW) is returned by the dummy function (XFDUM).

```
T:=XFPRV                    % T=function
A:=XPASW                    % A=XMSG password (version code)
*MON XMSG
T=:ISTAT                    % T=result status
```

When the task should no longer be privileged, the same call should be used, but with the A register equal to zero.

The reason for specifying the XMSG password, is to ensure that privileged programs, that base themselves on accessing XMSG table structures, have been updated to the current XMSG table definitions.

5 System Function Specifications

Note that all system functions are privileged.

These system functions are mainly used by the XMSG-COMMAND program to enable you to find out what the message system is doing. They should not normally be called by users, but are included here for reasons of completeness.

5.1 Initialize for System Functions (XFSIN)

This returns the base field address (B register) of the XMSG system in the memory bank where the XMSG kernel code has been fixed. This address is needed in order to be able to access XMSG tables by the use of the system functions.

```
T:=XFSIN                    % T=function
*MON XMSG
T=:ISTAT                    % T=result status
A=:BASEAD                   % A=XMSG base field address
                              (B register)
```

This privileged function is only allowed for RT-programs (i.e., not allowed from drivers).

5.2 Absolute Read from Physical Memory (XFABR)

This function allows a program to read a block of data from the part of the physical memory which is used by XMSG, into its own user area.

```
T:=XFABR                    % T=function
ULEN=:D                     % D=no. of bytes to read
A:=UADD                     % A=address of user buffer
X:=PDISP                    % X=address within the bank
*MON XMSG
T=:ISTAT                    % T=result status
```

Note that when calling this function, then bits 8 - 14 of the T register specify system (source) bank number. If zero (default), the data will be copied from the bank in which the XMSG kernel has been loaded, starting from the address specified by PDISP, to the user's logical space. If not zero, the data will be copied from the specified bank, starting from the address specified by PDISP.

This privileged function is only allowed for RT-programs (i.e., not allowed from drivers).

5.3 Create Driver (XFCRD)

This function is used to create a driver with a given context, as defined by the register block. XMSG then allocates an XT-block (task block) for the driver.

```
T:=XFCRD (BONE XFPON)       % T=function BONE option
UADD=:D                     % D=address of register block
A:=ILEV                     % A=interrupt level
*MON XMSG
T=:ISTAT                    % T=result status
A=:XTADDR                   % A=XT-block address
```

The ILEV parameter contains the interrupt level on which the driver should run on. XFPON (paging on) should be set if paging should be on when the driver is started. UADD points to an 8 word buffer which contains the register block that the driver will be started with, in the order required for the Load Register Block (LRB) hardware instruction (cf. NORD-100 Reference Manual - ND-06.014).

XFCRD allocates an XT-block to the driver and returns its address in the A register.

This privileged function is only allowed for RT-programs (i.e., not allowed from drivers).

5.4 Start Driver (XFSTD)

Starts an already created driver:

```
T:=XFSTD                    % T=function
A:=XTADDR                   % A=XT-block address
*MON XMSG
T=:ISTAT                    % T=result status
```

XFSTD overwrites the driver's L register with the driver's XT-block address before starting the driver.

In this way a started driver will store its XT-block address in the L register. The driver must make sure that the L register still contains the XT-block address before XMSG is called.

XFSTD does not set the appropriate bit in the PIE. Nor does it load or fix any segments. This should be done using FIXC and ENTSG - see the SINTRAN III Reference Manual - ND-60.128.

This privileged function is only allowed for RT-programs (i.e., not allowed from drivers).

A P P E N D I X B

XROUT SERVICES

1 General

The calls in this manual marked with 'XROUT Services', do not require any formatting of the messages to XROUT. In these cases the necessary formatting automatically is performed by the routines called. However, certain service functions, like 'Get Name from Magic Number' (XSGNM), require that you format the messages yourself. This can be done using the routines marked with 'Buffer Formatting'. Even so, you still need to know the XROUT message format.

You should note that many of the XROUT services are privileged. This implies that they can only be successfully executed by a privileged task. See the XMSG call XMPFPRV (Planc) or XMFFPRV (Fortran).

2 XROUT Message Format

The messages that users send to XROUT have a standard format:

Byte 0 - a serial number returned unchanged by XROUT in order to allow users, who may have many outstanding requests, to recognize particular replies. Note that messages sent from XROUT also return a special message type (XMROU) in the msgType parameter as a result from a receive call, like XMPFRCV, XMPFRRH or XMPFRRE. Thus they can be distinguished from messages originating from other tasks.

In order to comply with the ND standard message format, the high order bit of byte 0 should be zero.

Byte 1 - the service number (symbol XSxxx) of the service being requested. XROUT overwrites this with the return status from the request: 0 is OK, while other values are errors as defined by the XR... symbols. These are defined in appendix D. Note that XROUT service values and result/error codes are always in the range 0..255B, so that the user may set the high order bit (bit 7) to indicate user services and/or result status.

Bytes 2-3 - length of remainder of message in bytes, followed by a sequence of parameter blocks.

Each parameter block has the form:

Byte 0 - Parameter number and type (0 means: skip this byte to allow for fill). Integers have positive values, strings negative (two's complement of parameter number). If you use the buffer formatting routines listed in the reference guide part of this manual, in chapter 2 and 3, you do not have to observe this sign; The routines will take care of it for you.

Byte 1 - Length of parameter in bytes.

Byte 2 ... Parameter data.

The number and type of parameters are dependent on the particular service. All parameter blocks must start on even byte boundaries in the message. If you use the above mentioned formatting routines, they will do this for you. The message buffer sent to XROUT must be large enough for the reply, to be prepared if the latter is longer than the request. This is because XROUT uses the same buffer for the reply. If the buffer is too small for the reply, an error message will be returned to the originating task.

3 Services in Detail

The following is a list of XROUT services. The symbolic names XS... are defined in the files XMP:DEFS for Planc, and XMF:DEFS for Fortran.

3.1 Name a Port (XSNAM)

In order to name a port, the name must be declared to XROUT. This is done by sending the XSNAM service request from the port that is to be named.

Parameter	No	Type	Meaning
In:	1	String	Name of port

If any other open port already has the specified name, an error status is returned. Otherwise the sending port is given the specified name. If it already had a name, the port is renamed with the new name.

The maximum name length accepted (default = 32 bytes) is defined when the XMSG system is generated. You should note that XROUT will discard characters of an eventual excess.

3.2 Create Connection Port (XSCRS)

This service is very similar to XSNAM, but allows XROUT to control the number of users that a port can handle simultaneously. It may also distribute users among server ports.

Parameter	No	Type	Meaning
In:	1	String	Name of connection port
	2	Integer	Max. no of connections accepted
	3	Integer	Uniqueness flag

XROUT first handles the message like an XSNAM service request would do (see XSNAM above), except that connection ports are allowed to have identical names, unless the uniqueness parameter is specified and is non-zero. It then sets a counter (the free connection counter) associated with that port to the value specified in parameter 2. For remainder of specification, see the Send Letter service (XSLET) below.

3.3 Increment or Decrement Free Connection Count (XSNSP)

After opening a connection port (see XSCRS above), a task can later increment when connections becomes available, or decrement when the number of connections need to be reduced. For this free connection count (when connections become available) you may use the XSNSP service.

Parameter	No	Type	Meaning
In:	1	Integer	Number of extra connections

If parameter 1 is positive, the free connection counter is incremented. If the parameter is negative, the free connection counter is decremented. (If the resulting number of free connections becomes negative, an error is returned.)

3.4 Send Letter (XSLET)

This service is used to contact a remote port.

Parameter	No	Type	Meaning
In:	1	String	Port or Connection name
	2	String	System name
	4	Integer	Local Area Only flag (optional)

If parameter 2 is specified, XROUT will search in the name table, and if this has been defined as a remote name (see Define Remote Name, XSDRN, below), the letter is forwarded to the XROUT in the specified system. If the 'LAN only' flag is set and does not equal zero, the letter is only forwarded if contact has been established, and the remote system lies on the local network.

Otherwise XROUT extracts the identifier (parameter 1) and looks up the string in its name table. If a match is found, XROUT looks at the port type: If this is a normal named port (named by using the XSNAM service), the whole message is forwarded (option XFFWD) to the matching magic number.

If it is a connection port (named by using XSCRS), XROUT looks at the free connection count; If the count is greater than zero, it decrements it and forwards the letter. If not, it tries to find another port with the same name. If no match is found, the function code is set to an error value, and the message returned to the sender.

The remainder of the message can contain data for the receiving task (user name, password, ...). This will allow the server to check that the sender is entitled to use that service, before replying and thereby giving the caller his/her magic number. If the server wants to reply to the requester without giving away his/her own magic number, he/she should reply with the forward option (XFFWD).

3.5 Send Letter and Kick (XSLEK)

This privileged service is identical to the XSLET service described above, except that it also allows an RT monitor call to be executed on a specified program.

Parameter	No	Type	Meaning
In:	1	String	Port or Connection name
	2	String	System name
	3	String	Name of an RT program
	4	Integer	Kick flag (optional)

If the destination XROUT does not know the name of the destination port, it will execute an RT monitor call on the RT program specified in parameter 3, to start it up and return the letter with status OK. Note that the requesting program can test whether the target task or XROUT returned the message by checking the message type.

If parameter 4 is specified as an integer equal to one, XROUT will precede the RT monitor call with an ABORT.

3.6 Return a Null Status Message (XSNUL)

XROUT returns a message of two bytes containing the reference number and 0 (used for testing/benchmarking).

Parameters : none

3.7 Get Name from Magic Number (XSGNM)

Any XMSG user can obtain the name of a given port by sending a message containing the magic number as parameter 1.

Parameter	No	Type	Meaning
In:	1	Integer	Magic number
Out:	1	String	Port name

The return message will contain the port name appended as parameter 2, if there was space in the message buffer (make sure there is enough!).

3.8 Get Name from Magic Number not Less than Param (XSGNI)

Any XMSG-user can obtain the name of a port or remote system from the name table by sending a message containing a magic number as parameter 1.

Parameter	No	Type	Meaning
In:	1	Integer	Magic number
Out:	1	Integer	Magic number (or 0)
	2	String	Port or System name
	3	Integer	No of free connection/service points (optional)

XROUT will return the name with the least system magic number greater than or equal to the input parameter. The name's magic number is returned as parameter 1 and the name as parameter 2. If the name was that of a service, parameter 3 will contain the number of free connection/service points.

If no name was found, satisfying the above conditions, the first parameter is zero.

If the request contained no parameter 1, the name of the local system is returned.

3.9 Clear name of a port (XSCNM)

When a name's validity has expired, this service can be used to remove the name from the name table. It is done by sending the XSCNM service from the port whose name is to be cleared.

Parameters: none

Name clearing is also done automatically by XROUT when it notices that a port has been closed.

3.10 Get Magic Number from Name (XSGMG)

This is a privileged service, which returns the magic number for a particular name (see also XSGIN below).

Parameter	No	Type	Meaning
In:	1	String	Name
Out:	1	Integer	Magic number

If the specified name is the name of a connection/service port, and there are more than one port with this name, the magic number of the most recently defined name will be returned.

3.11 Get Information about Name (XSGIN)

This service returns information about a port or system name.

Parameter	No	Type	Meaning
In:	1	String	Port or System name
Out:	1	Integer	Port number (optional)
	2	Integer	System number

This service takes as input a name as parameter 1 and returns the system number as parameter 2. If the given name is a port name and not a system name, the port number is returned as parameter 1.

If the specified name is the name of a connection port, and there are more than one port with this name, the service only returns information about the most recently created connection port wich carries that particular name.

3.12 Define Remote Name (XSDRN)

XSDRN is used for defining the names of systems (specified as parameter 2 in letters - XSLET and XSLEK services). XSDRN is normally accessed via the Define-Remote-Name command of the XMSG-COMMAND background program. XSDRN is privileged, and requires two parameters.

Parameter	No	Type	Meaning
In:	1	String	System name
	2	Integer	System number

The specified name is put into the name table (must be unique). All letters that are addressed to that system (parameter 2 in XSLET and XSLEK), will be forwarded to the specified system. Note that a system can have many names. Thus the names to be used should be those identifying functional systems rather than physical systems, whenever possible (e.g., SIBAS-BACKEND or MAIL-HANDLER rather than ND-100-377.)

If the second parameter is not specified, the name is cleared, i.e., removed from XROUT's name table.

3.13 Define Local System (XSDLO)

This defines the specified system number as the number of the local system.

Parameter	No	Type	Meaning
In:	1	Integer	System number or 0

The system number must be defined as follows:

```
        For ND-100 systems=  the serial number
            ND-500 systems=  the serial number +  5000
             ND-10 systems=  the serial number +  9000
                satellites=  the serial number + 10000
```

If parameter 1 is zero, the service will use the SINTRAN system number.

You are not allowed to redefine the local system number. The definition of the local system number is done automatically when you start XMSG.

3.14 Define System Routing (XSDSY)

Whereas Define Remote Name (XSDRN) defines the mapping of a system name to an XMSG system number, Define System Routing (XSDSY) specifies how to get to that system. This is not necessary if the system is directly connected, since XROUT will find out when the link to that system starts up. It is necessary, however, for systems connected via neighbours. The XSDSY service is privileged and takes two parameters:

Parameter	No	Type	Meaning
In:	1	Integer	System number
	2	Integer	To be routed via this system (or 0)

The function will update the routing tables thus the specified system is marked as being available via the system defined in parameter 2. If the second parameter is zero, the system specified in parameter 1 is marked as 'not available'.

If no parameter 2 is specified, the specified system is removed from the system routing table.

The size of the routing table (i.e.,the number of other systems the local system can communicate with) is defined when XMSG is generated. Note that no non-local system, except directly connected systems, can communicate with the local system unless it has been defined using the XSDSY service.

XSDSY is usually accessed by using the Define-System-Route command in the XMSG-COMMAND program.

3.15 Get Routing Information for a System (XSGSY)

XMSG-COMMAND allows you to list the routing information held by any accessible XROUT in an XMSG network. This is done by sending XSGSY messages with an integer parameter, the object system number. XROUT replies with a message containing four integer parameters:

Parameter	No	Type	Meaning
In:	1	Integer	Object system number
Out:	1	Integer	The first system number found greater than or equal to that requested (or 0 if none)
	2	Integer	Connection type
	3	Integer	Extra info depending on connection type
	4	Integer	Network info

Par. 2 - Connection type	Par. 3 contains:
0 - Unavailable	
1 - Neighbour	Link Index
2 - Via	System number
3 - Via network server	Subaddress
4 - Local	Local system number

Par. 4 - Network info
<=377B - Number of hops in right byte (hop count)
>=400B - Number of WANs in left byte and number of hops in right byte, if any.

The integer parameter 4, Network Info, is interpreted as follows. If a route to a system involves 3 LAN hops, and 1 WAN, the parameter value will be 403B, i.e., left byte 001B and right byte 003B.

3.16 Starting up/Stopping an Inter-System Link (XSLKI)

This privileged service is used by the Start-Link and Stop-Link commands in the XMSG-COMMAND program. It is used when one wants to use an HDLC link or Megalink (which must have been declared in the generation of SINTRAN) as an inter-system link. The XSLKI request requires four parameters:

Parameter	No	Type	Meaning
In:	1	Integer	Link logical unit number
	2	Integer	Timeout value in XMSG Time Units (XTU) (one XTU = 0.1 sec.)
	3	Integer	Number of frames to allocate (window+1)
	4	Integer	Number of times to repeat (if <0 then infinite)

If number of frames to allocate (parameter 3) is greater than zero, XROUT will reserve the link (both input and output data fields), check that there are enough free frame buffers and then initialize the interface. The link will then enter the 'calling' state, which means that it tries to establish contact with the adjacent system, by sending a predefined (parameter 4) maximum number of SABM frames. (SABM= Set Asynchronous Balance Mode.)

When an SABM frame is received correctly, the link will enter the 'Connected' state and send Receiver Ready (RR) frames instead. When an RR from the adjacent system is received, it will enter the RUN state. At this point, the routing table will be updated to indicate that the neighbour is available over that particular link. Note that the XSLKI reply is returned to the requester as soon as the link is put into the CALL state.

If the number of frames to allocate (parameter 3) is less than zero a 'close' link operation is performed instead; the link is disabled, released, and the routing information updated.

If parameter 3 is equal to zero, this is a status request and the link state is returned as parameter 1. Link state: 0=DEAD(crashed), 1=INIT(being initialized), 2=CALL(trying to make contact with neighbour), 3=CONN(contact made with neighbour), 4=RUN(data phase), 5=KILL(closing down).

3.17 Starting up/Stopping a Network Server (XSNET)

XMSG allows one to replace an HDLC/Megalink with any other network that offers the same facilities. This is done by implementing a special program, called a network server, that takes the frames that would have been sent by the XMSG link layer over HDLC/Megalink and sends them over the other network instead.

The privileged Start/Stop Network Server service therefore is quite similar to the Start/Stop link service:

Parameter	No	Type	Meaning
In:	1	Integer	Magic number of server port
	3	Integer	Number of buffers to allocate
	4	Integer	Not equal to zero if WAN

The server magic number must have been obtained previously by direct communication with the server, eg., using the XSLEK service. The WAN flag is used to indicate that communication over this server is not free. Thus all letters that are sent with the 'LAN only' flag should stop here (cf. XSLET service).

If the number of buffers to allocate (parameter 3) is less than zero, the specified server is stopped.

If parameter 3 is equal to zero, this is a status request and the state of the particular network server is returned as parameter 1. Network server state: 0=DEAD(crashed), 1=INIT(being initialized), 2=CALL(trying to make contact with server), 3=CONN(contact made with server), 4=RUN(data phase), 5=KILL(closing down).

3.18 Trace Initialize (XSTIN)

This privileged service is used by the Open-Trace command in the XMSG-COMMAND program to initialize the trace system. It takes as a parameter, the file name of the trace file. XROUT then opens and initializes the file and starts up the trace dump foreground program (XTRACE).

Parameter	No	Type	Meaning
In:	1	String	Name of trace file
Out:	1	Integer	SINTRAN error code (optional)

Note that the same trace systems (events) will be enabled as for the previous trace.

If a file system error occurs when opening the trace file, the SINTRAN file system error code is returned as parameter 1.

3.19 Trace Close (XSTCL)

This privileged service is the opposite to XSTIN (above). It will write the last block(s) with trace information to the trace file, close the trace file and stop XTRACE.

Parameters: none

3.20 Define Trace Conditions (XSTDC)

This privileged service takes an integer as a parameter. If it is positive, the system (event) with that number is enabled for tracing. If it is negative the actual system is disabled. If zero, all tracing systems (events) are disabled.

Parameter	No	Type	Meaning
In:	1	Integer	Trace system (event) number

The following events can be traced. (Events 0 and 1 are automatically enabled.)

0 : Clock. Only output when necessary. Body (2 words) contains ATIME (in basic time units.)
1 : Trace management. First word of body contains the function
    1: open
    2: close
    3: enable/disable (next word contains the system number - negative means disable)
8 : XMSG Calls. The 5-word body contains the T-,A- and D-register, the XT-block address and the X-register from the caller.
9 : XMSG return to user. Body is as for system 8, but with Result Registers instead.
10 : Kernel context switch - traces queue and element address.
11 : Link Layer - frame received.
12 : Link layer - bad frame received and ignored.
13 : Link Layer - send frame. Trace body: AC bytes,length etc.
14 : Network Layer - complete datagram queued to receiver queue.
15 : Network Layer - datagram fragment received.
16 : Network layer - any frame received (inc. route through)
17 : Network layer - frame sent
18 : Network layer - control frame received
19 : Link layer - updating transmitter list
20 : Link layer - start transmitter
21 : Network layer - start transmitting new message
22 : Gateway - frame dechained on input

3.21 Set Crash Information (XSSCI)

This privileged service is used for defining the names of XMSG restart files and XMSG dump files. In addition, it is used to enable/disable the auto-restart flag, dump XMSG onto files and to get information about the currently defined restart/dump files and the current auto-restart flag. XSSCI is normally accessed via the commands of the XMSG-COMMAND background program.

Since XSSCI must be able to handle different types of requests (see above), the service is divided into sub-services.

The first parameter in the message sent to XROUT determines the requested sub-service. The number and type of the other parameters, depends on the particular sub-service.

The sub-services defined and the appropriate, respective parameters are as follows.

XSDAR: Enable/disable auto-restart:
This sub-service controls the setting of the auto-restart flag. If it is zero (default), the automatic XMSG restart facility is disabled. If not zero, the restart facility is enabled. If the restart facility is enabled and the restart files have been defined (see sub-service XSDRF), XMSG will automatically be restarted if an XMSG crash occurs.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSDAR
	2	Integer	0 leads to disable, ><0 leads to enable

XSDRF: Define restart files:
The sub-service is used for defining the names of XMSG restart files. This operation must be done if the automatic restart facility is to be used.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSDRF
	2	String	Batch input file name
	3	String	Batch output file name

When the restart-files are defined and auto-restart enabled, XROUT will, if an XMSG crash occurs, append the files specified as parameter 2 and 3 to the batch input and output queues respectively.
Note that the default owner of the restart files is user SYSTEM. If the files are owned bysomeone else, the file names must be prefixed with the name of that user.

If one of the parameters 2 or 3 is not specified, the restart file names are cleared.

XSDDF: Define dump files:
The sub-service is used for defining the names of XMSG dump files. This operation must be done before XMSG can be dumped onto files (for later investigation). When XMSG is dumped, the system will be dumped onto the files specified as parameter 2, 3 and 4 in the message.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSDDF
	2	String	Dump file name for segment 33 and 34
	3	String	Dump file name for XMSG tables
	4	String	Dump file name for XMSG message buffer pool

If one of the parameters 2,3 or 4 is not specified, the current dump file names are cleared. Note that the file name never should be enclosed in quotes ("). If the file does not exist, the new file will be created by XROUT. If the file exists, it must have been created as an indexed file.

XSGDF: Get defined dump files:
The return message will contain the currently defined XMSG dump file names appended as parameter 2, 3 and 4, if there was sufficent space in the message (make sure there is enough).

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGDF
Out:	1	String	Dump file name for segment 33 and 34
	2	String	Dump file name for XMSG tables
	3	String	Dump file name for XMSG message buffer pool

XSDUX: Dump XMSG onto files:
The sub-service is used to dump XMSG onto files (for post-usus analysis). This sub-service is normally accessed via theDump-XMSG command of the XMSG-COMMAND program, and when XMSG is dumped in the case of an XMSG crash. When accessed, XROUT will dump XMSG onto the files defined via sub-service XSDDF (see above).

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSDUX

XSGRF: Get defined restart files:
The return message will contain the currently defined XMSG restart file names appended as parameter 2 and 3, if there was sufficient space in the message (make sure there is enough).

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGRF
Out:	1	String	Batch input file name
	2	String	Batch output file name

XSGAR: Get auto-restart definition:
The return message will contain the current auto-restart definition appended as parameter 2, if there was sufficient space in the message (make sure there is enough).

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGAR
Out:	1	Integer	0 if disabled, ><0 if enabled

3.22 Get/Check Attribute (XSGAT)

This service is also divided into sub-services. The requested sub-service is specified by the first parameter in the message sent to XROUT. The number and type of the other parameters in the message, depends on the requested sub-service.

The sub-services (currently) defined and the appropriate, respective parameters are as follows:

XSGXV: Get XMSG version:
The return message will contain the version, revision and patch level of the running XMSG system as parameter 1,2 and 3.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGXV
Out:	1	String	Version (e.g. 'J').
	2	String	Revision (e.g. '00').
	3	Integer	Patch level (e.g. 0).

XSCMG: Check magic number:
Any user may check the validity of a magic number by sending a message containing the magic number as parameter 2. XROUT will return information about the requested magic number as parameter 1.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSCMG
	2	Integer	Magic number to be checked.
Out:	1	Integer	Information about the magic number: If the magic number is that of an open port in the accessed XROUT system, then parameter 1 = 1. If the magic number is that of a closed port, that of a system or that of a port in another system then parameter 1 = 0.

XSGCN: Deabbreviate system or port name:
Any user may deabbreviate a system or port name by sending a message containing the abbreviated name as parameter 2. XROUT replies with a message containing the full name of the port or system as parameter 2. If the name is a port name, the port number is returned as parameter 1; if the name is that of a system, the system number is returned as parameter 3.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGCN.
	2	String	Abbreviated name.
Out:	1	Integer	Port number (if the name is a port name).
	2	String	Deabbreviated (full) name.
	3	Integer	System no. (if the name is a system name).

XSGFR: Get friend information for a system (privileged):
The List-Friend-Systems command of the XMSG-COMMAND program allows one to list all systems defined as friends to the local system. This is done by sending a message with the object system number as parameter 2. The return message will contain two parameters (see below).

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSGFR.
	2	Integer	Object friend system number.
Out:	1	Integer	The first system number found greater than or equal to that requested, or 0 if none.
	2	Integer	Options (not yet implemented).

3.23 Define/Remove Attribute (XSDAT)

This privileged service is also divided into sub-services. The requested sub-service is specified by the first parameter in the message sent to XROUT. The number and type of the other parameters in the message, depends on the requested sub-service. XSDAT is normally accessed via the commands of the XMSG-COMMAND program.

The sub-services (currently) defined and the appropriate, respective parameters are as follows:

XSDFR: Define friend system:
This service is used to define a system as a friend to the local system. This is done by sending a message containing the friend system number as parameter 2.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSDFR.
	2	Integer	System number.

Note that this friendship is not reciprocal; You cannot declare yourself as a friend of another system.

XSRFR: Remove friend system:
When a friend system's validity has expired, the service is used to remove the specified system from the friend system table. This is done by sending a message containing the system number as parameter 2.

Parameter	No	Type	Meaning
In:	1	Integer	Sub-service = XSRFR.
	2	Integer	System number.

3.24 Get Network Server Information (XSNSI)

This is a privileged service, which returns information for a network server (i.e., a server which has been started by the Start-Network-Server command of the background command program). XSNSI is normally accessed via the List-Network-Servers command of the XMSG-COMMAND background program.

Parameter	No	Type	Meaning
In:	1	Integer	Virtual system number
Out:	1	Integer	The first virtual system number found greater than or equal to that requested, or 0 if none.
	2	String	Network server name.
	3	Integer	Link index in XMSG.
	4	Integer	Network type: 0 = local area, 1 = wide area.
	5	Integer	Network server port number.
	6	Integer	Gateway port number.
	7	Integer	Number of receive buffers allocated to the server.
	8	Integer	Number of transmit messages allocated to the server.

(Make sure that the message buffer, which is sent to XROUT, is big enough to contain the response parameters.)

A P P E N D I X C

The ND-100 XMSG System From PIOC

The XMSG system running in the ND-100 may be used by processes in PIOCOS through a system call. In this way processes in the PIOC may communicate with tasks in the ND-100. Processes in the PIOC may also use the XMSG in the ND-100 for internal communication, but in this case there is considerable overhead involved.

For a complete understanding of programming the PIOC, please see the PIOC Software Guide (ND-60.161).

XMSG record in PIOC (12-byte register block):

```
TYPE POXM = RECORD PACK
            INTEGER2 : Tregister
            INTEGER2 : Aregister
            INTEGER2 : Dregister
            INTEGER2 : Xregister
            INTEGER4 : user32bitAddress
            ENDRECORD
```

Example of use:

```
   POXM : xmsgBlock
   XFDUM =: xmsgBlock.Tregister
$* LEA xmsgBlock,A0
$* MOVE.W £FNXMSG,D0
$* TRAP £2
```

The user32bitAddress parameter is only used in the functions XFREA, XFWRI, XFRRE, and XFSMC.

All elements in a multicall from PIOCOS must consist of the 12-byte register blocks, even if the 32-bit address field is not used. Be aware that the driver edits the multi-call block and inserts XFDBK calls if the PIOC bank is changed from one call to another.

To avoid problems, you should not specify addresses in more than one bank (the large 512Byte PIOC has 4 banks).

We recommend that you do not use the XFWTF flag. If the XFWAK flag is set, the process will continue whether the function is completed or not. Upon completion, an event (BIT 31) will be generated for the appropriate process.

When the PIOC is unloaded (either by the PIOC monitor call or by the PIOC-MONITOR) all processes in the PIOC will be disconnected from XMSG.

The following calls are not allowed from the PIOC:

XFABR - Absolute read from physical memory
XFCRD - Define a driver for XMSG
XFDBK - Define a bank number for drivers
XFSTD - Start driver
XFWDF - Define wake-up context
XFDBK - Define a bank number for drivers
XFSIN - Initialize for system functions
XFDUB - Define a user buffer

A P P E N D I X D

XMSG ERROR CODES (PLANC OR FORTRAN)

General

The possible error codes together with the symbol defined in the XMP:DEFS or XMF:DEFS files are explained in this appendix. XMSG and XROUT error codes are described in different sections of the appendix.

The different XMSG and XROUT error messages are listed with their symbolic name, their SEC (Standard Error Code), and the pure XMSG error codes.

1 Error Codes Returned from XMSG Functions

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENTM	16896	XENTM	0

Description : The base value of the Standard Error System (SEC).
Explanation : This is not an error! Return status of not completed functions.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENOT	16897	XENOT	-1

Description : No more XT-blocks free.
Explanation : Each user of XMSG has, and must have, an XT-block (task block). An XT-block is automatically allocated by XMSG as soon as a task which has no task block makes an XMSG call of any kind.

The maximum number of tasks to be active, i.e., known by XMSG, at any time, is a system generation definition parameter.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIRM	16898	XEIRM	-2

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXETMM	16900	XETMM	-4

Description : Task is not allowed any more memory.
Explanation : A reserve or allocate message buffer(s) function has been issued, but the calling task is not allowed to reserve/allocate the message buffer(s) of the requested size.

The maximum number of bytes that a task can own at any time is a system generation definition parameter.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENIM	16901	XENIM	-5

Description : Facility not yet implemented.
Explanation : A function has been called, or an option requested, that is not yet implemented in XMSG.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIBP	16902	XEIBP	-6

Description : Illegal message buffer pointer.
Explanation : A function which refers to a message has been issued, but the specified message identifier is not a valid message identifier.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEBNY	16903	XEBNY	-7

Description : Message buffer not yours.
Explanation : A function which refers to a message has been issued, but the specified message is owned by another task.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIRT	16904	XEIRT	-8

Description : Illegal function for RT-programs (only drivers).
Explanation : A function has been issued which is not legal for RT-programs, eg., the functions 'Define Bank Number for Drivers' or the 'Define Wake-up Context'.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENOP	16905	XENOP	-9

Description : No more ports available.
Explanation : An open port function has been issued, but all ports are already in use.

The maximum number of ports to be used simultaneously is a parameter defined at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIDR	16906	XEIDR	-10

Description : Function not available to drivers.
Explanation : A driver has made a call which is not available for drivers, i.e., the function is only legal for RT-programs.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENDM	16907	XENDM	-11

Description : No default message.
Explanation : A function that operates on a default message or a function that has specified a message identifier of -1 has been issued which requires that a default message exists, but there is no current default message. A task may set the current (default) message with the 'Set Current Message' function.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEMCH	16908	XEMCH	-12

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEBFC	16909	XEBFC	-13

Description : Message is in a queue.
Explanation : A function which refers to a message has been issued, but the specified message is chained to a queue (e.g., chained to a port queue or to another internal XMSG queue).

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEAIN	16910	XEAIN	-14

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEBNC	16911	XEBNC	-15

Description : Return of a bounce message.
Explanation : A message sent with option XFBNC is returned to the sending task. This is not really an error.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEWNA	16912	XEWNA	-16

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENVI	16913	XENVI	-17

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEILF	16914	XEILF	-18

Description : Illegal function code in monitor call.
Explanation : An XMSG function has been requested which does not exist.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIMA	16915	XEIMA	-19

Description : Invalid magic number.
Explanation : A send function has been issued, but the destination magic number does not exist, or the receiving port is closed before the handling (destination) task has received the message, or the receiving port is closed while the message is 'port current' for that particular port.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEMFL	16916	XEMFL	-20

Description : Message space full.
Explanation : A reserve or allocate message buffer(s) function has been issued, but there is either no free message descriptor or no free space in the message buffer pool, large enough for the requested buffer size.

The error is also returned if a send function is issued when a secure message is sent to a task in another system, and the receiving (destination) task tries to reserve a message buffer for the incoming message when there is either no free message, or no free space in the message buffer pool, large enough for the incoming message.

The maximum number of messages to be reserved/allocated simultaneously is a parameter defined at system generation. This also applies for the total buffer space available for message buffers.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEILM	16917	XEILM	-21

Description : Illegal message size or not enough space left.
Explanation : A reserve/define message buffer function, an allocate message buffer(s) function, or a function which is used to write user data into a message buffer has been issued.

For the first type of function (reserve/define message buffer), the error is returned if the requested message size is greater than the maximum message size allowed, or if the option XFEXC (reserve exclusive message buffer) is specified and no exclusive message buffer of the requested size is available.

The maximum message size allowed is a parameter defined at system generation.

For the second type of function (allocate message buffers), the error is returned if the requested message size is equal to zero or greater than the maximum message size allowed, or of a different size than previously allocated messages.

For the third type of function (writing into message buffer), the error is returned if the message buffer size is too small to contain the user data.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIPN	16918	XEIPN	-22

Description : Illegal port number.
Explanation : A port referencing function has been issued, but the specified port is either not active (not in use), or the specified port number is greater than the highest port number in the system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEPRV	16919	XEPRV	-23

Description : Privileged function called without privilege.
Explanation : A privileged function has been called, but the caller is not privileged. The user may become privileged (for XMSG) by successfully executing the function 'Make Calling Task Privileged'.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEPVR	16920	XEPVR	-24

Description : Privilege request refused.
Explanation : To become privileged, the caller must be either a driver, direct task, foreground program on ring 2 or 3, or a background program logged in as user SYSTEM. The caller must also specify the correct XMSG password. The XMSG password can be obtained using the XMPCONF (XMFCONF) routine.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXERNA	16921	XERNA	-25

Description : Remote system not available.
Explanation : A send message function has been issued, but the local XMSG has no access to the remote system (e.g., if a link has not been started).

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEROV	16922	XEROV	-26

Description : Remote task space overflow.
Explanation : A send message function has been issued and a secure message was sent to another task. As the receiving, however, task was not allowed any more memory, the message was returned to the sending task.

The maximum number of bytes that a task can own at any time is a parameter defined at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEXBF	16923	XEXBF	-27

Description : Message already has XMSG buffer.
Explanation : The current message is reserved with a number of bytes not equal to zero.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEDRI	16924	XEDRI	-28

Description : Driver called XMSG before return from previous call.
Explanation : A driver is not allowed to call XMSG until a return has been made from the previous call.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENDP	16925	XENDP	-29

Description : No port open (so default port parameter invalid).
Explanation : A port referencing function, with a specified port number of zero, has been issued which requires that a default port exists, but there is no default/open port. The default port is the most recently opened port.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEITL	16926	XEITL	-30

Description : Illegal transfer length for read/write.
Explanation : A read or write function has been issued, but the requested number of bytes to transfer is less than zero, or the current message displacement plus the number of bytes to transfer becomes less than zero.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIDP	16927	XEIDP	-31

Description : Illegal displacement in read/write.
Explanation : A read or write function has been issued, but the specified message displacement is greater than or equal to the message buffer size.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEILR	16928	XEILR	-32

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENOS	16929	XENOS	-33

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENSE	16930	XENSE	-34

Description : Network sequencing error.
Explanation : A send function has been issued which resulted in the sending of a message to another system. XENSE is returned by the receiving (destination) system. For example, XENSE is returned if the received datagram identifier (sequence number) is not equal to the expected sequence number.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXERND	16931	XERND	-35

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEPCL	16932	XEPCL	-36

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENRU	16933	XENRU	-37

Description : XMSG not running.
Explanation : XMSG is not started or XMSG has crashed!

When XMSG is running, it spends a lot of its time checking itself. If one of these checks fails, XMSG will restart active XMSG tasks by means of this return status and close itself down.

If a task makes an XMSG call when XMSG is not running, SINTRAN will return this status when it is informed that XMSG has stopped.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENTO	16934	XENTO	-38

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXENUS	16935	XENUS	-39

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEREJ	16936	XEREJ	-40

Description : Network remote reject (request retransmit).
Explanation : A send function has been issued which resulted in the sending of a message to another system. The error is returned by the receiving (destination) system if the transport header, which is set up by the local XMSG, for the message is incorrect.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXEIXT	16937	XEIXT	-41

Description : Driver called XMSG with illegal XT-block.
Explanation : A driver has made an XMSG call with an illegal task block in its L-register, e.g., with an XT-block which belongs to a RT-program, with an XT-block which is not in use (not active), with an XT-block address which is outside the valid range for task blocks, or if XMSG has been stopped and restarted after the driver had received its XT-block address from XMSG.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXETMU	16938	XETMU	-42

Description : Too many multicalls.
Explanation : A 'Start Multicall' function has been issued, but the requested number of calls to execute is greater than the maximum allowed number of calls in a multicall.

The maximum number of calls in a multicall is a parameter defined at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXECRA	16959	XECRA	-63

Description : Not used!

2 Error Codes Returned from XROUT Services

The error value is always returned in byte 1 of the reply from XROUT.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRSOK	16960	XRSOK	0

Description : Ok, not an error.
Explanation : OK return from a service call.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRISN	16961	XRISN	1

Description : Illegal service number.
Explanation : A service has been requested which does not exist.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRUNN	16962	XRUNN	2

Description : No open port has this name.
Explanation : A service that refers to a port or a system name has been issued, but no port or system has the specified name. For example if one of the services. 'Send Letter', 'Send Letter and Kick', 'Get Information about Name', 'Define/Clear Remote Name', or 'Get Magic number from Name' has been issued, the error is returned if the specified port/system name (parameter 1) is unknown.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRDDF	16963	XRDDF	3

Description : Another port already has this name.
Explanation : A port or system naming service has been issued, but another port or system already has the specified name.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNSP	16964	XRNSP	4

Description : No space left for names.
Explanation : A port or system naming service has been issued. When a new port or system is to be named, XROUT must first allocate space for the name unit in its name table. The name table has a size defined at system generation. When full, this error is returned.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRIPT	16965	XRIPT	5

Description : Illegal parameter type.
Explanation : A parameter in the service request has an illegal type, i.e., if a parameter is of type 'integer' when it should have been of type 'string', or if a parameter is of type 'string' when it should have been of type 'integer'.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRMMP	16966	XRMMP	6

Description : Missing mandatory parameter.
Explanation : A mandatory parameter in the service request is missing: for example, if a send letter service call has been issued and parameter 1 (port or connection name) is missing.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRUNM	16967	XRUNM	7

Description : Unknown magic number.
Explanation : A service has been requested which requires that a specified magic number exists, or that the sending port should be named. You get this error message if either the magic number does not exist, or the sending port has not been named.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRMTL	16968	XRMTL	8

Description : Resulting message too long.
Explanation : A service call of any kind has been issued, but the message containing the service request is either less than 4 bytes, or it is too small to contain the returned XROUT parameter(s).

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRSMF	16969	XRSMF	9

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRPRV	16970	XRPRV	10

Description : Caller was not privileged.
Explanation : A privileged service has been requested, but the caller is not privileged. The user may become privileged (for XMSG) by successfully executing the function, 'Make Calling Task Privileged'.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRISY	16971	XRISY	11

Description : Illegal system number parameter.
Explanation : A service which refers to a system number has been issued, but the specified/requested system number can not be represented as a 16-bit integer word.

The error is also returned from a 'Define System Routing' service call if the first parameter (system number) is equal to zero or equal to the local XMSG system number; or if the second parameter (via system number) is equal to the first parameter or equal to the local XMSG system number; or, if it was a remove system request (parameter 2 missing), if the specified system (parameter 1) is not defined in the system routing table.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNRO	16972	XRNRO	12

Description : No access to remote system.
Explanation : A send letter, e.g., 'Send Letter' or 'Send Letter and Kick', service has been issued, but the specified remote system (parameter 2) is not available from the local XMSG system.

The error is, for instance, returned if the remote system is not defined in the system routing table, if the link has not yet been started, or if a timeout is detected by the network layer.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRIIV	16973	XRIIV	13

Description : Illegal integer value.
Explanation : A 'Increment/Decrement Free Connection Count' service has been issued, but the original (previous) number of connections plus the requested number of new connections (parameter 1) becomes less than zero.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNEI	16974	XRNEI	14

Description : Cannot define route to a neighbour.
Explanation : A 'Define System Routing' service has been issued which requires that the specified system (parameter 1) is not directly connected. It is not legal to define a route to an adjacent system (neighbour).

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNXM	16975	XRNXM	15

Description : Invalid service request - no inter-system XMSG.
Explanation : An inter-system XMSG service has been issued in a single-system XMSG system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRILN	16976	XRILN	16

Description : Illegal/Reserved Log. unit no. for link.
Explanation : A 'Start/Stop Link' or 'Start Network Server' service call has been issued.

If it is a 'Start Link' request, the error is returned if the specified link (parameter 1) is already started/active. If it is not started, the error status is returned if an error is returned from SINTRAN when XROUT tries to reserve the specified logical unit (parameter 1).

If it is a 'Stop Link' request, the error is returned if the specified link (parameter 1) is not active/started.

If it is a 'Start Network Server' request, the error is returned if the specified server (parameter 1) is already started/in use.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNXL	16977	XRNXL	17

Description : No more XL-Blocks (Link Descriptors).
Explanation : A 'Start Link' or 'Start Network Server' service call has been issued, but there is no free link descriptor.

Each inter-system link (i.e., HDLC or megalink) and each virtual link (i.e., link to a network server) has, and must have, an XL-block. An XL-block is automatically allocated by XMSG when a new link or network server is started.

The maximum number of links and/or network servers to be active, i.e., started, at any time is a parameter defined at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNXD	16978	XRNXD	18

Description : Not enough XD/XF-Blocks for start link/start server.
Explanation : A 'Start Link' or 'Start Network Server' service call has been issued. When an inter-system link (i.e., HDLC or megalink) or a virtual link (i.e., link to a network server) is started, a (user) specified number of XD-blocks (frame blocks) will be allocated for that particular link. If it is a virtual link, the same number of message headers (empty message buffers) need

to be allocated. When there are not enough free frame-blocks, or, if it is a 'Start Network Server' service, there are not enough free message headers (XM-blocks), this error is returned.

The maximum number of frame-blocks and message headers that can be allocated/reserved at any time are parameters defined at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNTR	16979	XRNTR	19

Description : No trace generated.
Explanation : A 'Trace initialize' service has been requested, but the trace facilities are not included in the current XMSG system. In order to include the trace facilities, modify the XMSG system definition file and generate a new system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTRA	16980	XRTRA	20

Description : Trace already active.
Explanation : A 'Trace initialize' service has been requested, but the trace is already initialized.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTRP	16981	XRTRP	21

Description : Trace passive.
Explanation : A 'Trace Close' or 'Define/Change Trace Conditions' service has been requested, but the trace system is not active.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTFE	16982	XRTFE	22

Description : Trace/Dump file open error (see param 1).
Explanation : A service that refers to a SINTRAN file system name has been requested, but a file system error was returned from SINTRAN when XROUT tried to open the trace file. The SINTRAN file system error is returned as parameter 1 in the response from XROUT.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTRT	16983	XRTRT	23

Description : Trace RT-prog (XTRACE) not found.
Explanation : A 'Trace initialize' service has been requested which requires that the trace program (XTRACE) exists, but it does not. XTRACE was (or should have been!) loaded when XMSG and XROUT were loaded onto their segments.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTIS	16984	XRTIS	24

Description : Illegal trace system number.
Explanation : A 'Define/Change Trace Conditions' service has been requested, but the specified trace system (event) number in parameter 1 is greater than 255.

(XMSG is able to trace up to 255 different system events, although so many are not yet implemented).

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRBLK	16985	XRBLK	25

Description : Not used!

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRSYD	16986	XRSYD	26

Description : Attempt to redefine local system number.
Explanation : A 'Define Local System' service call has been issued, but the local system number is already defined. It is illegal to redefine the local system number.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNLS	16987	XRNLS	27

Description : No local system number defined yet.
Explanation : A service call which refers to another XMSG system has been issued, which requires that the local system number is defined. For example, the services 'Define Remote Name', 'Define System Routing', 'Start Link', 'Start Network Server', etc. require that the local XMSG system number has been defined.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRTRE	16988	XRTRE	28

Description : Too many remote names applied to this system.
Explanation : A service used for defining the names of systems, e.g., 'Define Remote Name', has been issued. XMSG can handle a fixed maximum number of unique names (synonyms) for each remote system. When this limit is exceeded, this error is returned.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRRNA	16989	XRRNA	29

Description : Old service calls (<64) cannot go remote.
Explanation : A send letter service call has been issued which requires that the new type of service (service number >= 64) is requested.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRBUS	16990	XRBUS	30

Description : Service points busy.
Explanation : A send letter service, e.g., 'Send Letter', which is used to contact a port defined as a 'connection port', has been issued. If the free connection counter, for that particular connection port, is less than or equal to zero, the letter is returned to the sending task with this error code.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNSE	16991	XRNSE	31

Description : This is not a service port.
Explanation : A connection port referencing service (eg., the service 'Increment or Decrement Free connection Count') has been issued, but the referenced port is not a connection port.

(A connection port is a port named using the 'Create Connection Port' service.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRRPN	16992	XRRPN	32

Description : Remote port statically declared.
Explanation : A 'Clear Name' service call has been issued, but the referenced port name is statically declared.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRUKS	16993	XRUKS	33

Description : Unknown remote system name.
Explanation : A send letter service, e.g., 'Send Letter' or 'Send Letter and Kick', has been issued which requires that the destination system name (parameter 2) is known by the local system, but the specified name is not defined as a name of a (remote) system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRMFL	16994	XRMFL	34

Description : Remote system message table space full.
Explanation : A send letter service, e.g., 'Send Letter' or 'Send Letter and Kick', has been issued. The error is returned if the receiving task tries to reserve a message descriptor and a message buffer for the incoming message (letter), when there is either no free message descriptor, or no free space in the message buffer pool large enough for the incoming message.

The maximum number of messages that can be reserved simultaneously and the total buffer space available for message buffers are parameters at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRROV	16995	XRROV	35

Description : Remote task message space used up.
Explanation : A send letter, e.g., 'Send Letter' or 'Send Letter and Kick', service has been issued. The letter was sent to another task, but the receiving task was not allowed any more memory, so the letter (message) was returned to the sending task.

The maximum number of bytes that a task can own at any time is a parameter at system generation.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRRFU	16996	XRRFU	36

Description : Routing table full (too many systems).
Explanation : A system routing definition service call has been issued. When the system routing is defined, XROUT must allocate space (for the new system) in the system routing table. The size of the system routing table (i.e., the number of other systems that the local system can communicate with) is defined when XMSG is generated. When the routing table is full, this error is returned.

(Two typical services that could receive this error are 'Define System Routing' and 'Start Network Server'.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNRB	16997	XRNRB	37

Description : No remote batch service available.
Explanation : A 'Remote Append Batch' request has been sent to the File Transfer server, but no remote batch device has been defined. In order to include the batch definition, modify the XMSG system definition file and generate a new system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRURT	16998	XRURT	38

Description : Unknown RT name.
Explanation : A 'Send Letter and Kick' service call has been issued which requires that the specified RT program (parameter 3) exists, but the specified name is not a name of a RT-program.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRSNR	16999	XRSNR	39

Description : This server is not running.
Explanation : A 'Start Network Server' service call has been issued which requires that a network server (RT-program) is running. If the specified magic number (parameter 1) is invalid, or if the server is not running (e.g., if the server has been aborted), this error is returned.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRRND	17000	XRRND	40

Description : Netserver: remote system is not defined.
Explanation : A message should have been sent out of the local system via a network server, but the specified remote system is unknown to the network server. (The error is returned by the network server.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNNA	17001	XRNNA	41

Description : Netserver: network not available.
Explanation : A message should have been sent to a remote system via a network server, but the network server has no access to the remote system. (The error is returned by the network server.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRISE	17002	XRISE	42

Description : Netserver: internal server error.
Explanation : A message should have been sent out of the local system via a network server, but the network server is malfunctioning (due to inconsistencies). (The error is returned by the network server.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRIRQ	17003	XRIRQ	43

Description : Netserver: invalid request.
Explanation : A request is sent to a network server, but the requested facility is not (yet) implemented in the server, or the parameters in the request are not accepted by the network server. (The error is returned by the network server.)

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNGA	17004	XRNGA	44

Description : XMSG not configured with gateway code.
Explanation : A 'Start Network Server' service has been requested, but the gateway software is not included in the current XMSG system. In order to include the gateway code, modify the XMSG system definition file and generate a new system.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRRNL	17005	XRRNL	45

Description : Remote system not on same LAN.
Explanation : A 'Send Letter' service call has been issued which requires that the remote system (parameter 2) lie inside the local network, but the specified system is only available via a wide area network.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRNCO	17006	XRNCO	46

Description : No connection to this system (unknown status).
Explanation : A 'Send Letter' service call has been issued which requires that the letter should only be sent if contact has (already) been established with the remote system specified in parameter 2. If the remote system is not yet connected to the local XMSG system, this error is returned.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRAMB	17007	XRAMB	47

Description : Ambiguous name.
Explanation : A service that refers to a port or a system name has been issued, but the specified name is not unique.

SEC Symbol:	SEC Value:	XMSG Symbol:	XMSG Value:
XMXRFFU	17008	XRFFU	48

Description : Friend system table full (too many systems).
Explanation : A 'Define Friend System' service call has been issued. When a new friend system is defined, XROUT must allocate space for the new system in the friend system table. The size of the friend system table (i.e., the number of systems that can be defined as friends) is defined when XMSG is generated. When the friend system table is full, this error is returned.

A P P E N D I X E

RR-LIB ERROR CODES

General

The different RR-LIB error messages are listed with their symbolic name, their SEC (Standard Error Code), and their error description.

XMSG Error Codes Coming from Planc or Fortran Interface

Codes Specific to RRPCslct/RRPCcall/RRPCdisc:

SEC Symbol:	SEC Value:
RRERnttm	17296

Description : Not terminated (i.e., timeout).

SEC Symbol:	SEC Value:
RRERdscn	17297

Description : Disconnected (by server or RR-Lib).

SEC Symbol:	SEC Value:
RRERunev	17298

Description : Unexpected event while waiting for reply (i.e., RREVunkn or RREVothr).

Codes in General Use:

SEC Symbol:	SEC Value:
RRERntcl	17304

Description : Not initialized as client.

SEC Symbol:	SEC Value:
RRERntsr	17305

Description : Not initialized as server.

SEC Symbol:	SEC Value:
RRERntei	17306

Description : Not initialized as client or server.

SEC Symbol:	SEC Value:
RRERbdid	17307

Description : Invalid remote identifier.

SEC Symbol:	SEC Value:
RRERbdst	17308

Description : Call not valid in current state.

SEC Symbol:	SEC Value:
RRERxsin	17309

Description : Excess information in request or response.

SEC Symbol:	SEC Value:
RRERbdln	17310

Description : Parameter of invalid length.

SEC Symbol:	SEC Value:
RRERbdnm	17311

Description : Invalid name specified.

SEC Symbol:	SEC Value:
RRERbdbf	17312

Description : Bad buffer specification.

SEC Symbol:	SEC Value:
RRERbdpm	17313

Description : Other parameter error.

SEC Symbol:	SEC Value:
RRERxscn	17314

Description : Limit on connections reached.

SEC Symbol:	SEC Value:
RRERtslm	17315

Description : Task buffer space limit has been reached.

SEC Symbol:	SEC Value:
RRERmsfl	17316

Description : Communication subsystem has no more buffer space.

SEC Symbol:	SEC Value:
RRERprrf	17317

Description : Request for privilege refused.

SEC Symbol:	SEC Value:
RRERntpr	17318

Description : Function requires privilege.

SEC Symbol:	SEC Value:
RRERdcpn	17319

Description : Disconnect pending.

SEC Symbol:	SEC Value:
RRERincp	17320

Description : Incomplete received data.

SEC Symbol:	SEC Value:
RRERfatal	17343

Description : FATAL ERROR value - contact ND.

Errors Related to the Underlying Communication System

If RRERxnru or RRERxcra error codes are received, then the current RR context has been lost: all connections should be considered closed and the user is in possession of any outstanding user buffers (with indeterminate contents). The only permissible call after these errors is RRPINIT, which will return the same code until XMSG is running again, at which point new connections may be opened etc.

SEC Symbol:	SEC Value:
RRERxnru	17344

Description : XMSG not running (=XENRU)

SEC Symbol:	SEC Value:
RRERxcra	17345

Description : XMSG crash (=XECRA)

SEC Symbol:	SEC Value:
RRERxnxt	17346

Description : XMSG out of XT-blocks (=XENOT)

SEC Symbol:	SEC Value:
RRERxnpt	17347

Description : XMSG out of ports (=XENOP)

SEC Symbol:	SEC Value:
RRERxnsp	17348

Description : XMSG has no space for name (=XRNSP)

Disconnect Codes

SEC Symbol:	SEC Value:
RRDCuser	17281

Description : Normal disconnect by user.

SEC Symbol:	SEC Value:
RRDCrmcg	17282

Description : Remote congestion. All connections at the server are busy (= XRBUS = All connections at this port busy).

SEC Symbol:	SEC Value:
RRDCunsr	17283

Description : Unknown server (= XRUNN = No open port with this name).

SEC Symbol:	SEC Value:
RRDCunsy	17284

Description : Unknown system (= XRUKS = Unknown system).

SEC Symbol:	SEC Value:
RRDCngfl	17285

Description : Connection negotiation failed (incompatible RRs).

SEC Symbol:	SEC Value:
RRDCcref	17286

Description : Connection request refused on this network connection (= XRNSE = Not a connect port).

SEC Symbol:	SEC Value:
RRDCnoac	17287

Description : No access to remote system (= XRNRO or XRNCO).

SEC Symbol:	SEC Value:
RRDCrmdd	17288

Description : Remote end has terminated (= XEIMA).

SEC Symbol:	SEC Value:
RRDCxsin	17289

Description : Remote end's buffer was too small for the data sent.

SEC Symbol:	SEC Value:
RRDCprer	17290

Description : Protocol error.

A P P E N D I X F

TLIB ERROR CODES

General

The different TLIB error messages are listed with their symbolic name, their SEC (Standard Error Code), and their error description.

Errors Related to TLIB

SEC Symbol:	SEC Value:
TLERNTIN	17192

Description : The user has not done TLPINIT.

SEC Symbol:	SEC Value:
TLERBDRF	17193

Description : The user gave an invalid TLIB reference number.

SEC Symbol:	SEC Value:
TLERBDCL	17194

Description : The call is not valid in the current state.

SEC Symbol:	SEC Value:
TLERBDFM	17195

Description : Parameter of invalid format.

SEC Symbol:	SEC Value:
TLERBDLN	17196

Description : Parameter of invalid length.

SEC Symbol:	SEC Value:
TLERBDNM	17197

Description : Invalid name specified.

SEC Symbol:	SEC Value:
TLERBDPM	17198

Description : Other parameter error.

SEC Symbol:	SEC Value:
TLERNOUB	17199

Description : TLIB lacks user buffer.

SEC Symbol:	SEC Value:
TLERSMUB	17200

Description : The returned buffer is too small.

SEC Symbol:	SEC Value:
TLERDCPN	17201

Description : There is a disconnect pending.

SEC Symbol:	SEC Value:
TLERXOUT	17202

Description : Still expedited data is outstanding.

SEC Symbol:	SEC Value:
TLERNOTB	17203

Description : An attempt was made to open too many connections.
Increase TLMXconnections.

SEC Symbol:	SEC Value:
TLERNOSD	17204

Description : An attempt was made to use too many suffixes.
Increase TLMXaccessPoints.

SEC Symbol:	SEC Value:
TLERNOBD	17205

Description : An attempt was made to use too many user buffers.
Increase TLMXbuffers.

SEC Symbol:	SEC Value:
TLERNOQD	17206

Description : Insufficient internal descriptors for XMSG messages. Increase TLMXqueuedTpdus.

SEC Symbol:	SEC Value:
TLERNORM	17207

Description : TLIB cannot get more buffer space from the underlying system. The user should release some (e.g., by reading data) and try again.

SEC Symbol:	SEC Value:
TLERDPSF	17208

Description : Suffix is already in use by another task.

SEC Symbol:	SEC Value:
TLERFATAL	17215

Description : FATAL ERROR value - contact ND.

PLANC programmers can find diagnostic information on the cause of the error in the EXPORTED variables TLFATerr and TLFATinfo (see TLP:IMPT). Contact ND with the values of these variables.

Errors Related to the Underlying Communication System

If TLERXNRU or TLERXCRA error codes are received, then the current TLIB context has been lost. All connections should be considered closed and all "listens" should be considered terminated. The user is in possession of any outstanding user buffers (with indeterminate contents). The only permissible call after these errors is TLPINIT, which will return the same code until XMSG is running again, at which point new connections may be opened.

SEC Symbol:	SEC Value:
TLERXNRU	17256

Description : XMSG not running (=XMXENRU).

SEC Symbol:	SEC Value:
TLERXCRA	17257

Description : XMSG crash (=XMXECRA).

SEC Symbol:	SEC Value:
TLERXNXT	17258

Description : XMSG out of task blocks (=XMXENOT).

SEC Symbol:	SEC Value:
TLERXNPT	17259

Description : XMSG out of ports (=XMXENOP).

SEC Symbol:	SEC Value:
TLERXNSP	17260

Description : XMSG has no space for name (=XRNSP).

Disconnect Codes

Symbol:	Return Value
TLDCsesn	128

Description : Normal disconnect by user.

Symbol:	Return Value
TLDCrmcg	129

Description : Remote congestion. All connections at this suffix are busy (= XRBUS = All connections at this port are busy, or there is no outstanding TLIB listen).

Symbol:	Return Value
TLDCngfl	130

Description : Connection negotiation failed (incompatible TLIBs).

Symbol:	Return Value
TLDCprer	133

Description : Protocol error.

Symbol:	Return Value
TLDCcref	136

Description : Connection request is refused on this network connection (= XRNSE = Not a connect port).

Symbol:	Return Value
TLDCunet	137

Description : Unknown network address (= XRUKS = Unknown system).

Symbol:	Return Value
TLDCusuf	138

Description : Unknown suffix address. (= XRUNN = No open port with this name).

Symbol:	Return Value
TLDClcrs	200

Description : Local TLIB resources exceeded (e.g., lack of descriptors to queue TPDUs or XMSG space overflow).

Symbol:	Return Value
TLDCrmrs	201

Description : Remote TLIB resources exceeded (e.g., lack of descriptors to queue TPDUs or XMSG space overflow).

Symbol:	Return Value
TLDCrmdd	202

Description : Remote end has terminated (= XMXEIMA).

Symbol:	Return Value
TLDCnoac	203

Description : No access to remote system (= XRNRO).

Symbol:	Return Value
TLDCunxr	204

Description : Unknown XROUT error code received when trying to reach remote system.

A P P E N D I X G

SAMPLE PROGRAMS USING XMSG/PLANC

1 Introduction

The following set of two communicating background tasks is only intended to illustrate how contact may be established.

CLIENT takes the initiative to make contact with SERVER, and SERVER then replies: "Hello CLIENT". SERVER has to be started before CLIENT, because it has to name the port by which CLIENT wants to make contact.

The included files, XMP:DEFS and XMP:IMPT, contain some important definitions, e.g., the options (XFWTF etc.) and special data types (e.g., Xmmsgidentifier).

1.1 Brief Description of CLIENT

1) A letter to SERVER is made, consisting only of a letter header giving the name of the system where SERVER resides, and the name of a port which SERVER has opened (see the heading "Format letter to SERVER in internal buffer").

2) Resources from XMSG are requested: a port to communicate through and an XMSG buffer which the letter can be put into (see the heading "Get necessary resources from XMSG").

3) The letter is put into the XMSG buffer (XMPFWRI) and the letter is sent to XROUT (which knows it has to forward it to the port with the right magic number). The letter cannot be sent directly to SERVER, because CLIENT has not been informed of the magic number of SERVER's port at this point (see the heading "Send the letter to SERVER through XROUT").

4) A waiting state is entered, assuming that server will respond (XMPFRCV with XFWTF option). The response arrives at the port and resides there in an XMSG buffer. It is transferred to an internal task buffer (XMPFREA), and then output on the terminal (see the heading "Wait for message from SERVER, take it when it comes").

5) Finally, the XMSG resources are released before exit (see the heading "Release resources and exit").

1.2 Notes

CLIENT does not have to give a name to its port, therefore the XMPFOPN call is chosen, rather than XMPOPNM. The latter involves communication with XROUT to update XROUT's name table.

After the response came from SERVER, CLIENT could have obtained SERVER's magic number and sent messages directly. If SERVER and CLIENT was supposed to keep the communication going, XROUT would no longer be needed.

1.3 Brief Description of SERVER

1) A port is opened and given the name "s-port". This is done by the call XMPOPNM which, in addition to opening a port, also contacts XROUT, so that XROUT's name table can be updated with the port name (see the heading "Get necessary resources from XMSG").

2) A waiting state is entered, assuming that XROUT will forward a letter from client (XMPFRCV with XFWTF option). The response arrives at the port in an XMSG buffer (see the heading "Wait for message from XROUT").

3) The magic number of the CLIENT port is obtained by using the message status call, XMPFMST (see the heading "Get magic number of client").

4) The same message buffer that was received from CLIENT via XROUT is used for the message "Hello CLIENT" (XMPFWRI), and sent to CLIENT using the XMPFSND call (See the heading "Send message to client").

5) All resources are given back to XMSG.

1.4 Notes

SERVER does not need to obtain a message buffer from XMSG by using XMPFGET, because it can use the buffer that CLIENT sent via XROUT. SERVER does not need to read the letter from XROUT, unless is suppsed to check any user data beyond the letter header (the latter is not done in the example).

1.5 The Client Program

```
MODULE EX_CLIENT_PL
$LIST OFF
$INCLUDE XMP:DEFS
$INCLUDE XMP:IMPT
$LIST ON
%
  IMPORT (ROUTINE VOID,VOID: MON0)
%
  CONSTANT SBUFF = 100
  INTEGER ARRAY: STACK(0:1000)
  INTEGER: FLAGS := 0, MESSOFFSET := 0, BUFFOFFSET := 0
  INTEGER: SERIALNUMBER := 100, OFFSET :=0, MESSLENGTH := 20
  BYTES: INBUFFER(0:SBUFF-1)
  BYTES: OUTBUFFER(0:SBUFF-1)
  BYTES: PORTNAME := 'S-PORT'
  BYTES: SYSTEMNAME := 'SNORRE'
%*************************************************************************
%
%       Error handling
%
%*************************************************************************
  ROUTINE VOID,VOID(INTEGER): TERMINATE(ERRNUM)
    OUTPUT(1,'I6',ERRNUM)
    MON0
  ENDROUTINE
%*************************************************************************
%
%       Main program
%
%*************************************************************************
  PROGRAM: MAIN
%
    INTEGER: RETURNSTATUS,PORTNUMBER
    INTEGER: SIZEBUFFER,WRITTENLENGTH,MSGTYPE,REMOTEPORT
    INTEGER: LENGTHORSTAT,READLENGTH,SYSLENGTH
    INTEGER: PORTLENGTH
    XMMSGIDENTIFIER: MSGIDENT
```

```
%
    INISTACK STACK
%
%*****************************************************************
%
%       Format letter to SERVER in internal buffer
%
%*****************************************************************
    XMPBLET(OUTBUFFER,MESSLENGTH,OFFSET,SERIALNUMBER,&
            SYSTEMNAME,PORTNAME) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPBLET  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%*****************************************************************
%
%       Get necessary resources from XMSG
%
%*****************************************************************
    XMPFOPN(FLAGS,PORTNUMBER) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFOPN  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%
    OFFSET =: SIZEBUFFER
    XMPFGET(FLAGS,SIZEBUFFER,MSGIDENT) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFGET  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%*****************************************************************
%
%       Send the letter to SERVER through XROUT
%
%*****************************************************************
    SIZEBUFFER  =: MESSLENGTH
    XMPFWRI(FLAGS,BUFFOFFSET,ADDR(OUTBUFFER(0)) FORCE XMUSERADDRESS,&
            MESSOFFSET,MESSLENGTH,WRITTENLENGTH) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFWRI  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%
    0 =: FLAGS
    XMPROUT(FLAGS,MSGIDENT,PORTNUMBER) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPROUT  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
```

```
%*********************************************************************
%
%       Wait for message from SERVER, take it when it comes
%
%*********************************************************************
    2**XFWTF =: FLAGS
    XMPFRCV(FLAGS,PORTNUMBER,MSGTYPE,REMOTEPORT,MSGIDENT,&
                              LENGTHORSTAT) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFRCV  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%
    0 =: FLAGS
    XMPFREA(FLAGS,BUFFOFFSET,ADDR(INBUFFER(0)) FORCE XMUSERADDRESS,&
                  MESSOFFSET,MESSLENGTH,READLENGTH) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFREA  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
    OUTPUT(1,'A',INBUFFER)
%*********************************************************************
%
%        Release resources and exit
%
%*********************************************************************
    XMPFDCT(FLAGS) =: RETURNSTATUS
  ENDROUTINE
ENDMODULE
$EOF
```

1.6 The Server Program

```
MODULE EX_SERVER_PL
$LIST OFF
$INCLUDE XMP:DEFS
$INCLUDE XMP:IMPT
$LIST ON
%
  IMPORT (ROUTINE VOID,VOID: MON0)
%
  INTEGER ARRAY: STACK(0:1000)
  INTEGER: FLAGS := 0, MESSOFFSET := 0, BUFFOFFSET := 0
  BYTES: OUTBUFFER := 'HELLO CLIENT          '
  BYTES: PORTNAME := 'S-PORT'
%***************************************************************************
%
%       Error handling
%
%***************************************************************************
  ROUTINE VOID,VOID(INTEGER): TERMINATE(ERRNUM)
    OUTPUT(1,'I6',ERRNUM)
    MON0
  ENDROUTINE
%***************************************************************************
%
%       Main program
%
%***************************************************************************
  PROGRAM: MAIN
%
    INTEGER: SERIALNUMBER,RETURNSTATUS,PORTNUMBER
    INTEGER: SIZEBUFFER,WRITTENLENGTH,MSGTYPE,REMOTEPORT
    INTEGER: LENGTHORSTAT,LENGTH,READLENGTH
    INTEGER: PORTLENGTH
    INTEGER4: REMOTEMAGIC
    XMMSGIDENTIFIER: MSGIDENT
```

```
%
    INISTACK STACK
%
%*********************************************************************
%
%       Get necessary resources from XMSG
%
%*********************************************************************
    XMPOPNM(FLAGS,PORTNAME,PORTNUMBER) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPOPNM  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%
    OUTPUT(1,'A','PORT OPENED WITH NAME  ')
    OUTPUT(1,'A',PORTNAME)
%*********************************************************************
%
%       Wait for message from XROUT
%
%*********************************************************************
    2**XFWTF =: FLAGS
    XMPFRCV(FLAGS,PORTNUMBER,MSGTYPE,REMOTEPORT,MSGIDENT,&
                              LENGTHORSTAT) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFRCV.&
                     Expecting reply from XROUT  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
    IF MSGTYPE >< XMROU THEN
       OUTPUT(1,'A','WRONG MESSAGE TYPE  ')
       OUTPUT(1,'I6',MSGTYPE)
       MON0
    ENDIF
%*********************************************************************
%
%       Get magic number of client
%
%*********************************************************************
    0 =: FLAGS
    XMPFMST(FLAGS,MSGIDENT,MSGTYPE,REMOTEMAGIC,LENGTH) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFMST  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%*********************************************************************
%
%       Send message to client
%
%*********************************************************************
    XMPFWRI(FLAGS,BUFFOFFSET,ADDR(OUTBUFFER(0)) FORCE XMUSERADDRESS,&
              MESSOFFSET,SIZE OUTBUFFER,WRITTENLENGTH) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFWRI  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
```

```
%
    0 =: FLAGS
    XMPFSND(FLAGS,PORTNUMBER,REMOTEMAGIC) =: RETURNSTATUS
    IF RETURNSTATUS >< XMOK THEN
       OUTPUT(1,'A','Bad status from XMPFSND  ')
       TERMINATE(RETURNSTATUS)
    ENDIF
%*********************************************************************
%
%        Release resources and exit
%
%*********************************************************************
    0 =: FLAGS
    XMPFDCT(FLAGS) =: RETURNSTATUS
    MONO
  ENDROUTINE
ENDMODULE
$EOF
```

A P P E N D I X H

SAMPLE PROGRAMS USING XMSG/FORTRAN

1 Introduction

The following set of two communicating background tasks is only intended to illustrate how contact may be established.

CLIENT takes the initiative to make contact with SERVER, and SERVER then replies: "Hello CLIENT". SERVER has to be started before CLIENT, because it has to name the port which CLIENT wants to make contact.

1.1 Brief Description of CLIENT

1) A letter to SERVER is made, consisting only of a letter header giving the name of the system where SERVER resides, and the name of a port which SERVER has opened (see the heading "Format letter to SERVER in internal buffer").

2) Resources from XMSG are requested: a port to communicate through and an XMSG buffer which the letter can be put into (see the heading "Get necessary resources from XMSG").

3) The letter is put into the XMSG buffer (XMFFWRI) and the letter is sent to XROUT (which is programmed has to forward it to the port with the right magic number). The letter cannot be sent directly to SERVER, because CLIENT has not been informed of the magic number of SERVER's port at this point (see the heading "Send the letter to SERVER through XROUT").

4) A waiting state is entered, assuming that SERVER will respond (XMFFRCV with XFWTF option). The response arrives at the port and resides there in an XMSG buffer. It is transferred to an internal task buffer (XMFFREA), and then output on the terminal (see the heading "Wait for message from SERVER, take it when it comes").

5) Finally, the XMSG resources are released before exit (see the heading "Release resources and exit").

1.2 Notes

CLIENT does not have to give a name to its port, therefore the XMFFOPN call is chosen, rather than XMFOPNM. The latter involves communication with XROUT to update XROUT's name table.

After the response came from SERVER, CLIENT could have obtained SERVER's magic number and sent messages directly. If SERVER and CLIENT was supposed to keep the communication going, XROUT would no longer be needed.

1.3 Brief Description of SERVER

1) A port is opened and given the name "s-port". This is done by the call XMFOPNM, which, in addition to opening a port, also contacts XROUT, so that XROUT's name table can be updated with the port name (see the heading "Get necessary resources from XMSG).

2) A waiting state is entered, assuming that XROUT will forward a letter from client (XMFFRCV with XFWTF option). The response arrives at the port in an XMSG buffer (see the heading "Wait for message from XROUT").

3) The magic number of the CLIENT port is obtained by using the message status call, XMFFMST (see heading "Get magic number of client").

4) The same message buffer that was received from CLIENT via XROUT is being used for the message "Hello CLIENT" (XMFFWRI), and sent to CLIENT using the XMFFSND call (See the heading "Send message to client").

5) All resources are given back to XMSG.

1.4 Notes

SERVER does not need to obtain a message buffer from XMSG by using XMFFGET, because it can use the buffer that CLIENT sent via XROUT. SERVER does not need to read the letter from XROUT, unless it is supposed to check any user data beyond the letter header (the latter is not done in the example).

1.5 The Client Program

```
      PROGRAM CLIENT
$LIST OFF
$INCLUDE XMF:DEFS
$LIST ON
C
      EXTERNAL XMFBLET,XMFOPNM,XMFFGET,XMFFWRI,XMFROUT,XMFFRCV,XMFFMST
     +,XMFFSND,XMFFREA,XMFFDCT
      INTEGER XMFBLET,XMFOPNM,XMFFGET,XMFFWRI,XMFROUT,XMFFRCV,XMFFMST
     +,XMFFSND,XMFFREA,XMFFDCT
C
      INTEGER SERIALNUMBER,RETURNSTATUS,OFFSET,MESSLENGTH,PORTNUMBER,
     +SIZEBUFFER,MSGIDENT,WRITTENLENGTH,MSGTYPE,REMOTEPORT,
     +LENGTHORSTAT,LENGTH,READLENGTH,FLAGS,BUFFOFFSET,MESSOFFSET
C
      INTEGER*2 INPUTBUFFER(0:49),OUTPUTBUFFER(0:49)
C
      CHARACTER INBUF*100
C
      CHARACTER SYSTEMNAME*30,PORTNAME*30
C
      EQUIVALENCE (INBUF,INPUTBUFFER(0))
C*********************************************************************
C
C       Format letter to SERVER in internal buffer
C
C*********************************************************************
      MESSLENGTH = 20
      OFFSET = 0
      SERIALNUMBER = 100
      SYSTEMNAME = 'SNORRE '
      PORTNAME = 'S-PORT '
      RETURNSTATUS = XMFBLET(OUTPUTBUFFER,MESSLENGTH,OFFSET,
     +               SERIALNUMBER,SYSTEMNAME(1:-1),PORTNAME(1:-1))
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFBLET'
        GO TO 888
      ENDIF
C*********************************************************************
C
C       GET NECESSARY RESOURCES FROM XMSG
C
C*********************************************************************
      FLAGS = 0
      RETURNSTATUS = XMFFOPN(FLAGS,PORTNUMBER)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFOPN'
        GO TO 888
      ENDIF
```

```
C
      FLAGS = 0
      SIZEBUFFER = OFFSET
      RETURNSTATUS = XMFFGET(FLAGS,SIZEBUFFER,MSGIDENT)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFGET'
        GO TO 888
      ENDIF
C*****************************************************************
C
C       SEND THE LETTER TO SERVER THROUGH XROUT
C
C*****************************************************************
      FLAGS = 0
      MESSLENGTH = OFFSET
      BUFFOFFSET = 0
      MESSOFFSET = 0
      RETURNSTATUS = XMFFWRI(FLAGS,BUFFOFFSET,OUTPUTBUFFER,MESSOFFSET,
     +                       MESSLENGTH,WRITTENLENGTH)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFWRI WHEN CONTACTING XROUT'
        GO TO 888
      ENDIF
C
      FLAGS = 0
      RETURNSTATUS = XMFROUT(FLAGS,MSGIDENT,PORTNUMBER)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFROUT'
        GO TO 888
      ENDIF
C*****************************************************************
C
C       WAIT FOR MESSAGE FROM SERVER, TAKE IT WHEN IT COMES
C
C*****************************************************************
      FLAGS = 2**XFWTF
      RETURNSTATUS = XMFFRCV(FLAGS,PORTNUMBER,MSGTYPE,REMOTEPORT,
     +                       MSGIDENT,LENGTHORSTAT)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFRCV'
        GO TO 888
      ENDIF
C
      FLAGS = 0
      BUFFOFFSET = 0
      MESSOFFSET = 0
      LENGTH = 30
      RETURNSTATUS = XMFFREA(FLAGS,BUFFOFFSET,INPUTBUFFER,MESSOFFSET,
     +                       LENGTH,READLENGTH)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFREA'
        GO TO 888
      ENDIF
```

```
C*********************************************************************
C
C       OUTPUT MESSAGE FROM SERVER
C
C*********************************************************************
      WRITE (1,100)INBUF
  100 FORMAT (2X,A30)
      GO TO 999
C*********************************************************************
C
C       ERROR MESSAGE
C
C*********************************************************************
  888 WRITE (1,101)RETURNSTATUS
  101 FORMAT(2X,'RETURNSTATUS =',I6)
C*********************************************************************
C
C       RELEASE RESOURCES AND EXIT
C
C*********************************************************************
  999 FLAGS = 0
      RETURNSTATUS = XMFFDCT(FLAGS)
      END
```

1.6 The Server Program

```
      PROGRAM SERVER
$LIST OFF
$INCLUDE XMF:DEFS
$LIST ON
C
      EXTERNAL XMFOPNM,XMFFRCV,XMFFMST,XMFFREA,XMFFWRI,XMFFSND,XMFFDCT
      INTEGER  XMFOPNM,XMFFRCV,XMFFMST,XMFFREA,XMFFWRI,XMFFSND,XMFFDCT
C
      INTEGER SERIALNUMBER,RETURNSTATUS,MESSLENGTH,PORTNUMBER,
     +SIZEBUFFER,MSGIDENT,WRITTENLENGTH,MSGTYPE,REMOTEPORT,
     +LENGTHORSTAT,LENGTH,READLENGTH,FLAGS,BUFFOFFSET,MESSOFFSET
C
      INTEGER*2 OUTPUTBUFFER(0:49)
C
      INTEGER*4 REMOTEMAGIC
C
      CHARACTER OUTBUF*100
C
      CHARACTER PORTNAME*30
C
      EQUIVALENCE (OUTBUF,OUTPUTBUFFER(0))
C
C*********************************************************************
C
C       GET NECESSARY RESOURCES FROM XMSG
C
C*********************************************************************
      PORTNAME = 'S-PORT'
      FLAGS = 0
      RETURNSTATUS = XMFOPNM(FLAGS,PORTNAME(1:6),PORTNUMBER)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFOPNM'
        GO TO 888
      ENDIF
      WRITE(1,*)'PORT OPENED WITH NAME  ',PORTNAME
C*********************************************************************
C
C       WAIT FOR MESSAGE FROM XROUT
C
C*********************************************************************
      FLAGS = 2**XFWTF
      RETURNSTATUS = XMFFRCV(FLAGS,PORT,NUMBER,MSGTYPE,REMOTEPORT,
     +                       MSGIDENT,LENGTHORSTAT)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFRCV. EXPECT. REPLY FROM XROUT'
        GO TO 888
      ENDIF
```

```
      IF (MSGTYPE .NE. XMROU) THEN
        WRITE(1,*)'WRONG MESSAGE TYPE'
        GO TO 890
      ENDIF
C*********************************************************************
C
C       GET MAGIC NUMBER OF CLIENT
C
C*********************************************************************
      FLAGS = 0
      RETURNSTATUS=XMFFMST(FLAGS,MSGIDENT,MSGTYPE,REMOTEMAGIC,LENGTH)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFMST'
        GO TO 888
      ENDIF
C*********************************************************************
C
C       SEND MESSAGE TO CLIENT
C
C*********************************************************************
      FLAGS = 0
      BUFFOFFSET = 0
      MESSOFFSET = 0
      OUTBUF = 'HELLO CLIENT                '
      RETURNSTATUS = XMFFWRI(FLAGS,BUFFOFFSET,OUTPUTBUFFER,MESSOFFSET,
     +                       LENGTH,WRITTENLENGTH)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFWRI'
        GO TO 888
      ENDIF
C
      FLAGS = 0
      RETURNSTATUS = XMFFSND(FLAGS,PORTNUMBER,REMOTEMAGIC)
      IF (RETURNSTATUS .NE. XMOK) THEN
        WRITE(1,*)'BAD STATUS FROM XMFFSND'
        GO TO 888
      ENDIF
      GO TO 999
C*********************************************************************
C
C       ERROR MESSAGES
C
C*********************************************************************
  888 WRITE (1,101)RETURNSTATUS
  101 FORMAT(2X,'RETURNSTATUS =',I6)
      GO TO 999
  890 WRITE (1,102)MSGTYPE
  102 FORMAT(2X,'MSGTYPE =',I3)
C*********************************************************************
C
C       RELEASE RESOURCES AND EXIT
C
C*********************************************************************
  999 FLAGS = 0
      RETURNSTATUS = XMFFDCT(FLAGS)
      END
```

A P P E N D I X I

SAMPLE PROGRAMS USING RR-LIB

1 Introduction

The following set of communicating programs, called RR-SERVER and RR-LOW-CLIENT, gives an example of connection establishment, a few data transfer calls, and disconnection.

Notice the inclusion of the files RRP:DEFS and RRP:IMPT, which contain standard symbols and import definitions respectively. Also note that RR-SERVER has to be started before RR-CLIENT.

1.1 Brief Description of RR-SERVER

- The first call to RR-LIB is RRPBINIT.
- The loop (DO WHILE TRUE) shows a typical way of processing incoming events. The server is in a waiting state until an event occurs. Depending on the type of event, RR-SERVER responds appropriately.

  Note that since there is no exit from the loop, RR-SERVER may indefinitely serve different clients, sequentially.

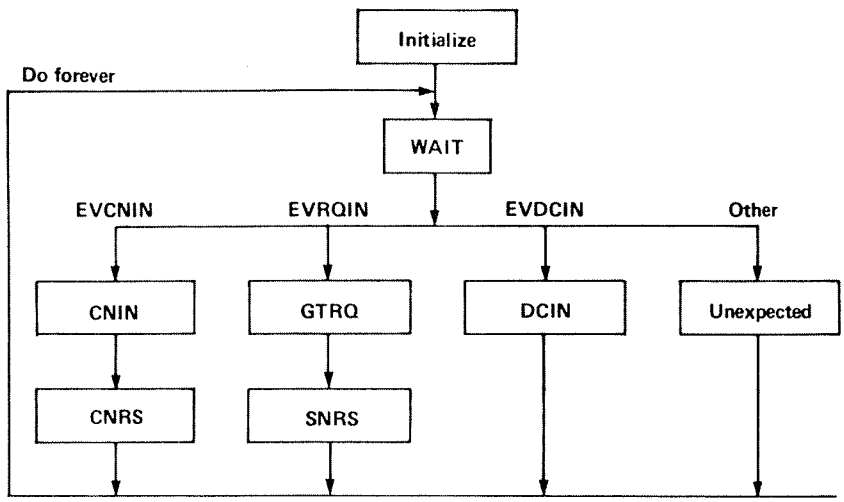


Fig. 1. Flowchart of RR-SERVER

1.2 Brief Description of RR-LOW-CLIENT

- The first call to RR-LIB is RRPBINIT.

- A connection request is sent to the server (RRPCCNRQ). This request contains a greeting, "HI this is client FRED", as information to the server.

- Loop for event processing:

  The first expected event to occur is the connection confirmation event (RREVcncf) as the result of a successful connection request. The client now sends a request to the server (RRPCSNRQ).

  The next expected event to occur is a response indication event (RREVrsin), which tells the client that a response has arrived. The client may now send another request, or he/she may disconnect by sending a RRPBDCRQ.

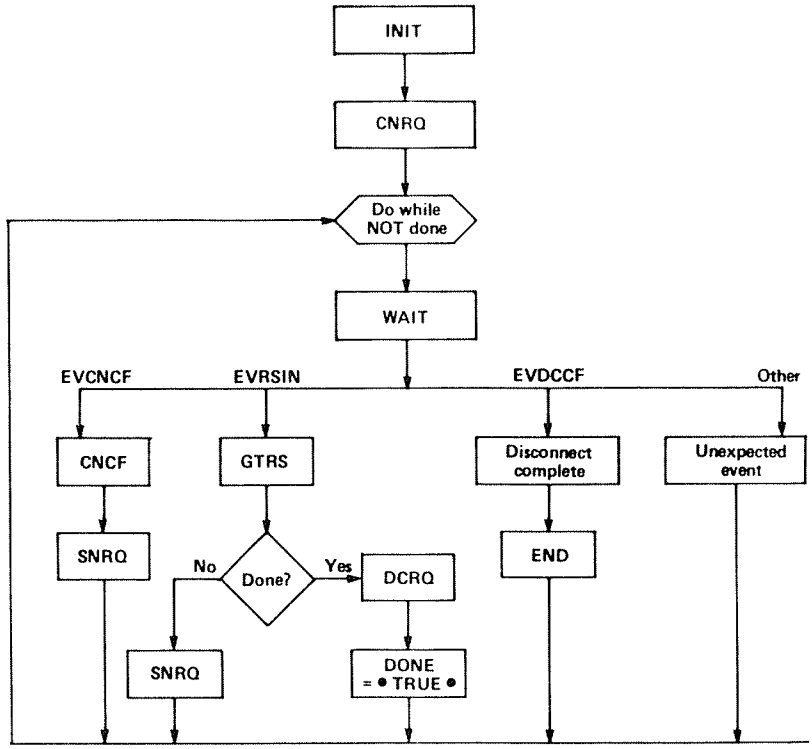


Fig. 2. Flowchart of RR-LOW-CLIENT

1.3 The RR-HIGH-CLIENT

A listing of the program RR-HIGH-CLIENT is also included in this chapter. This client is doing the same as the RR-LOW-CLIENT, but is using the high-level client calls. The program can be run with the

1.4 The Server Program

```
RR-SERVER.
MODULE X

$LIST OFF
$INCLUDE RRP:DEFS
$INCLUDE RRP:IMPT
$LIST ON

$MACRO CHKstatus(status)                        % macro for status checking
   IF "status" >< OK THEN                       % in this simple example all
       output(1,'A','$BAD status')              % errors only cause a
   ENDIF                                        % "bad-status" sentence
$ENDMACRO

ROUTINE VOID,VOID(BYTES): outtext(text)
   output(1,'A',text)
ENDROUTINE

INTEGER ARRAY: stk(0:1000)                      % program stack

INTEGER ARRAY: staticRRarea(0:RRSZstatic-1)     % fixed-size work area for RR-LIB
INTEGER ARRAY: dynamicRRarea(0:RRSZdynamic-1) % dyn.-size work area for RR-LIB
BYTES ARRAY: buffers(1:1,0:100)                 % user buffer

BYTES: serverName := 'RRSERVER'                 % server port name
RRID: clientID                                  % used to specify remote client

BYTES: serverInfo := 'Welcome to server RRSERVER' % server user data to be sent
BYTES: response := 'this is the response'          % with connect-response and
                                                   % send-response
%-------------------------------------------------------------------------------%

PROGRAM: SERVER

RRTM: waitTime                              % used to specify TIME-OUTs on wait
RREV: requestedEvent                        % used to specify requested event
RREV: nextEvent                             % used to specify occurred event
BYTES POINTER: receivedData                 % used to refer received data
BYTES POINTER: outBuffer                    % used to refer response buffer
BYTES POINTER: outData                      % used to refer response data
INTEGER: RRstatus                           % status from RR-LIB calls
INTEGER: reason                             % reason if bad status from RR-LIB
RRSP: serverParameters                      % used to specify server parameters

   INISTACK STK

   RRanyRemote =: requestedEvent.RREVremote % any remote client
   RRanyEvent =: requestedEvent.RREVevent   % any event desired

   32767 =: waitTime.RRTMlength             % specification of time-out
   4 =: waitTime.RRTMunits                  % values
```

```
TRUE =: serverParameters.RRSPisDefault   % use default server parameters
RRPBinit(staticRRarea,dynamicRRarea,&    % start use RR-LIB in server mode
        RRMDasServer,0,serverName,&      % (RRPBinit must always be the first
        buffers,1,serverParameters) =: RRstatus  % RR-LIB call)
CHKstatus(RRstatus)

DO WHILE TRUE

    RRPBwait(staticRRarea,waitTime,requestedEvent,&  % wait for an event
            nextEvent) =: RRstatus
    CHKstatus(RRstatus)

    IF nextEvent.RREVevent = RREVcnin THEN  % connection request has arrived
        outtext('$SERVER: connect attempt')
        nextEvent.RREVremote =: clientID    % specify the remote client
        RRPScnin(staticRRarea,clientID,&    % get information sent by client
                receivedData,outBuffer) =: RRstatus % in connection attempt
        CHKstatus(RRstatus)
        outtext(' with client info')
        outtext('$')
        outtext(IND(receivedData))
        ADDR(IND(outBuffer)(MININDEX(IND(outBuffer),1):&
                            MININDEX(IND(outBuffer),1)+SIZE serverInfo-1))&
                                                    =: outData
        serverInfo =: IND(outData)
        RRPScnrs(staticRRarea,clientID,IND(outData)) % accept connection
    ELSIF nextEvent.RREVevent = RREVrqin THEN  % request has arrived
        outtext('$SERVER: received request')
        RRPSgtrq(staticRRareaRRarea,clientID,&
                receivedData,outBuffer) =: RRstatus  % get request
        CHKstatus(RRstatus)
        outtext('$')
        outtext(IND(receivedData))
        ADDR(IND(outBuffer)(MININDEX(IND(outBuffer),1):&
                        MININDEX(IND(outBuffer),1)+SIZE response-1))&
                                                    =: outData
        response =: IND(outData)
        RRPSsnrs(staticRRarea,clientID,&   % send response
                IND(outData)) =: RRstatus
        CHKstatus(RRstatus)
```

```
        ELSIF nextEvent.RREVevent = RREVdcin THEN   % disconnected from client
            RRPBdcin(staticRRarea,clientID,reason,& % get information associated
                     receivedData) =: RRstatus      % with disconnect event
            CHKstatus(RRstatus)
            outtext('$SERVER: disconnect')
            IF reason = RRDCuser THEN     % normal disconnect by user
                outtext(' by client with client info')
                outtext('$')
                outtext(IND(receivedData))
            ELSE                          % another reason for disconnect
                outtext(' reason')
                output(1,'I5',reason)
            ENDIF
        ELSE
            outtext('$SERVER: unexpected event')
        ENDIF
    ENDDO
    RETURN
ENDROUTINE

ENDMODULE

%-------------------------------------------------------------------------------%
```

1.5 The Low-level Client Program

```
MODULE X

$LIST OFF
$INCLUDE RRP:DEFS
$INCLUDE RRP:IMPT
$LIST ON

$MACRO CHKstatus(status)                        % macro for status checking
   IF "status" >< OK THEN                       % in this simple example all
       output(1,'A','$BAD status')              % errors only cause a
   ENDIF                                        % "bad-status" sentence
$ENDMACRO

ROUTINE VOID,VOID(BYTES): outtext(text)
   output(1,'A',text)
ENDROUTINE

INTEGER ARRAY: stk(0:1000)                      % program stack

INTEGER ARRAY: staticRRarea(0:RRSZstatic-1)     % fixed-size work area for RR-LIB
INTEGER ARRAY: dynamicRRarea(0:RRSZdynamic-1) % dyn.-size work area for RR-LIB

BYTES: destSystem := 'DONALD'                   % remote system name
BYTES:destServer := 'RRSERVER'                  % remote server port name
RRID: serverID                                  % used to specify remote server

BYTES: connectInfo := 'HI, this is client FRED' % client user data to be sent
BYTES: request := 'this is a request'             % with different kind of
BYTES: disconnectInfo := 'goodbye'                % requests

BYTES: replyBuffer(0:100)                       % reply buffer

%-------------------------------------------------------------------------%

PROGRAM: THECLIENT

RRTM: waitTime                          % used to specify TIME-OUTs on wait
RREV: requestedEvent                    % used to specify requested event
RREV: nextEvent                         % used to specify occurred event
BYTES POINTER: reply                    % used to refer reply buffer
INTEGER: numberRequests                 % number of requests client want to send
INTEGER: requestsCompleted              % number of requests client has sent
INTEGER: RRstatus                       % status from RR-LIB calls
BOOLEAN: done
BYTES: nullName(0: -1)                  % for use
BYTES ARRAY: nullBuffers(0: -1,0: -1)   % in
RRSP: nullRRSP                          % RRPBinit call

   INISTACK STK
```

```
RRanyRemote =: requestedEvent.RREVremote % any remote server
RRanyEvent =: requestedEvent.RREVevent   % any event desired

5 =: waitTime.RRTMlength                 % specification of time-out
2 =: waitTime.RRTMunits                  % values

2 =: numberRequests

RRPBinit(staticRRarea,dynamicRRarea,&    % start to use RR-LIB in client mode
        RRMDasClient,1,nullName,&        % (RRPBinit must always be the first
        nullBuffers,0,nullRRSP) =: RRstatus   % RR-LIB call)
CHKstatus(RRstatus)

RRPCcnrq(staticRRarea,destsystem,destServer,&  % attempt to open a connection
        connectInfo,ADDR(replyBuffer),&        % to server
        serverID) =: RRstatus
CHKstatus(RRstatus)

0 =: requestsCompleted
FALSE =: done

DO WHILE NOT done

    RRPBwait(staticRRarea,waitTime,&                % wait for an event
            requestedEvent,nextEvent) =: RRstatus
    CHKstatus(RRstatus)

    IF nextEvent.RREVevent = RREVcncf THEN      % connection accepted
        outtext('$CLIENT: connect confirmation event')
        RRPCcncf(staticRRarea,&                % get information sent by server
                serverID,reply) =: RRstatus % when connection was accepted
        CHKstatus(RRstatus)
        outtext(' with server info')
        outtext('$')
        outtext(IND(reply))
        RRPCsnrq(staticRRarea,serverID,request,&  % send request
                ADDR(replyBuffer)) =: RRstatus
        CHKstatus(RRstatus)
    ELSIF nextEvent.RREVevent = RREVrsin THEN  % response has arrived
        outtext('$CLIENT: received response')
        RRPCgtrs(staticRRarea,serverID,reply) =: RRstatus  % get response
        CHKstatus(RRstatus)
        outtext('$')
        outtext(IND(reply))
        IF ((requestsCompleted+1) =: requestsCompleted)<numberRequests THEN
            RRPCsnrq(staticRRarea,serverID,request,&  % send request
                    ADDR(replyBuffer)) =: RRstatus
            CHKstatus(RRstatus)
        ELSE
            RRPBdcrq(staticRRarea,serverID,&       % terminate communication
                    disconnectInfo) =: RRstatus   % with server
            CHKstatus(RRstatus)
        ENDIF
```

```
        ELSIF nextEvent.RREVevent = RREVdeef THEN   % user-initiated disconnect
            outtext('$CLIENT: disconnect complete') % is completed
            TRUE =: done
        ELSE
            outtext('$CLIENT: unexpected event response')
        ENDIF


    ENDDO

    RETURN

ENDROUTINE

ENDMODULE

%-------------------------------------------------------------------------------%
$EOF
```

1.6 The High-level Client Program

```
MODULE X

$LIST OFF
$INCLUDE RRP:DEFS
$INCLUDE RRP:IMPT
$LIST ON

$MACRO CHKstatus(status)                        % macro for status checking
   IF "status" >< OK THEN                       % in this simple example all
       output(1,'A','$BAD status')              % errors only cause a
   ENDIF                                        % "bad-status" sentence
$ENDMACRO

ROUTINE VOID,VOID(BYTES): outtext(text)
   output(1,'A',text)
ENDROUTINE

INTEGER ARRAY: stk(0:1000)                      % program stack

INTEGER ARRAY: staticRRarea(0:RRSZstatic-1)     % fixed-size work area for RR-LIB
INTEGER ARRAY: dynamicRRarea(0:RRSZdynamic-1) % dyn.-size work area for RR-LIB

BYTES: destSystem := 'DONALD'                   % remote system name
BYTES:destServer := 'RRSERVER'                  % remote server port name
RRID: serverID                                  % used to specify remote server

BYTES: connectInfo := 'HI, this is client FRED' % client user data to be sent
BYTES: request := 'this is a request'             % with different kind of
BYTES: disconnectInfo := 'goodbye'                % requests

BYTES: replyBuffer(0:100)                         % reply buffer

%-------------------------------------------------------------------------%

PROGRAM: THECLIENT

RRTM: waitTime                          % used to specify TIME-OUTs on wait
BYTES POINTER: reply                    % used to refer reply buffer
INTEGER: numberRequests                 % number of requests client want to send
INTEGER: requestsCompleted              % number of requests client has sent
INTEGER: RRstatus                       % status from RR-LIB
INTEGER: reason                         % reason if bad status from RR-LIB
BYTES: nullName(0: -1)                  % for use
BYTES ARRAY: nullBuffers(0: -1,0: -1)   % in
RRSP: nullRRSP                          % RRPBintit call
```

```
   INISTACK STK

   5 =: waitTime.RRTMlength              % specification of time-out
   2 =: waitTime.RRTMunits               % values

   2 =: numberRequests

   RRPBinit(staticRRarea,dynamicRRarea,&   % start use RR-LIB in client mode
            RRMDasClient,1,nullName,&      % (RRPBinit must always be the first
            nullBuffers,0,nullRRSP) =: RRstatus    % RR-LIB call)
   CHKstatus(RRstatus)

   RRPCslct(staticRRarea,destsystem,destServer,&   % select server for future
            connectInfo,ADDR(replyBuffer),&        % interaction
            waitTime,serverID,reply,reason) =: RRstatus
   CHKstatus(RRstatus)
   outtext('$CLIENT: server selected with server info')
   outtext('$')
   outtext(IND(reply))

   0 =: requestsCompleted

   DO WHILE (requestsCompleted < numberRequests)

       RRPCcall(staticRRarea,serverID,request,&   % send request and wait for
                ADDR(replyBuffer),&               % response
                waitTime,reply,reason) =: RRstatus
       CHKstatus(RRstatus)
       outtext('$CLIENT: received response')
       outtext('$')
       outtext(IND(reply))
       requestsCompleted+1 =: requestsCompleted

   ENDDO

   RRPCdisc(staticRRarea,serverID,&   % terminate connection to server
            disconnectInfo,waitTime,reason) =: RRstatus
   CHKstatus(RRstatus)

   outtext('$CLIENT: disconnect complete')

   RETURN

ENDROUTINE

ENDMODULE

%-------------------------------------------------------------------------------%
$EOF
```

A P P E N D I X J

SAMPLE PROGRAMS USING TLIB/PLANC

1 Introduction

The following set of communicating programs, called TLP-SERVER and TLP-CLIENT, just gives an example of how a connection is established, and then disconnected. The connection establishment and disconnection in the example in section 6.7 serve as an outline of what happens at runtime. It shows the interaction between the two programs.

Notice the inclusion of the files TLP:DEFS and TLP:IMPT, which contain standard symbols and import definitions respectively. Also note that TLP-SERVER has to be started before TLP-CLIENT.

1.1 Brief Description of TLP-SERVER

- The first call to TLIB is TLPINIT.

- The next call to TLIB is TLPSTLS. This allows possible clients to get a connection request through.

- The loop (DO FOREVER) shows a typical way of processing incoming events. The server is in a waiting state until an event occurs. Depending on the type of event, TLP-SERVER responds appropriately.

  Note that since there is no exit from the loop, TLP-SERVER may indefinitely serve different clients, sequentially, on this same connection.

1.2 Brief Description of TLP-CLIENT

- The first call to TLIB is TLPINIT.

- A connection request is sent to the server (TLPCNRQ). This request contains a greeting, "greetings from client", as user data.

- Loop for event processing:

  The first expected event to occur is the connection confirmation event (TLEVcncf), as the result of a successful connection request. If the server sends user data in response to the greeting, which TLP-SERVER actually does, then a buffer is provided for the user data (TLPPRBF). Since there is no ordinary data to send, i.e., no data transfer phase, a disconnect request (TLPDCRQ) with "farewell" user data is sent back to the server.

The next expected event to occur is a disconnection confirmation (TLEVdccf) which completes the disconnection, and thus terminates the dialogue.

1.3 The Server Program

```
%-------------------------------------------------------------------------------%
%                                                                               %
%                              TLP-SERVER                                       %
%                                                                               %
%     A simple server illustrating the use of the PLANC interface to TLIB.      %
%     Note that the client is not expected to send any data after the           %
%     connection is established.                                                %
%                                                                               %
%-------------------------------------------------------------------------------%
MODULE TLPEXAMPLE

$LIST OFF
$INCLUDE TLP:DEFS
$INCLUDE TLP:IMPT
$LIST ON

IMPORT (ROUTINE VOID,VOID: MONO)

$MACRO CHKstatus(text,status)          % define a macro for status checking
   IF status >< OK THEN                % in this simple example, all
       output(1,'A','$BAD STATUS from ') % errors cause termination
       output(1,'A',"text")
       output(1,'I6',"status")
       MONO
   ENDIF
$ENDMACRO

$MACRO FOREVER
   WHILE TRUE
$ENDMACRO

%-------------------------------------------------------------------------------%
%                                                                               %
%                              GLOBAL DATA                                      %
%                                                                               %
%-------------------------------------------------------------------------------%

INTEGER ARRAY: stack(0:1000)                    % program stack
INTEGER ARRAY: TLdynamic(0:TLSZDYNAMIC-1)       % storage for TLIB
BYTES: serverName := 'SERVER'                   % server port name
BYTES: greetings := 'welcome to the server'     % server user data to be
                                                % sent with connect response
```

```
%-------------------------------------------------------------------------%
%
%       Main program
%
%-------------------------------------------------------------------------%

PROGRAM: SERVER

INTEGER: TLIBreference                  % connection identifier required by TLIB
INTEGER: myReference                    % connection identifier used by program
INTEGER: disconnectReason               % reason for disconnect
INTEGER: credit                         % initial credit for sending data
INTEGER: status                         % status from TLIB calls
INTEGER: lengthData                     % length of data received
BYTES: inputBuffer(0:99)                % buffer for received data
TLMD: mode                              % used to specify TLIB mode
TLTM: maxwaitTime                       % used to specify max. wait time
TLEV: requestedEvent                    % used to specify events to wait for
TLEV: actualEvent                       % gives actual event that occurred
TLBFLOGICAL: buffer                     % used to describe a buffer
TLAP2STRING: serverAddress              % address of server
TLAPXMSG: clientAddress                 % address of client as magic number
TLQS: quality                           % quality of service given by connection

   INISTACK stack
                                                % initialize TLIB
   TLXMinUserMode =: mode.TLMDxmsgmode          % XMSG in user mode
   TRUE =: mode.TLMDuniqueSuffix                % only ONE server active
   TLMXconnections =: mode.TLMDmxConnections    % max. simultaneous connections
   TLMXaccessPoints =: mode.TLMDmxAccessPoints  % max. no of TLAPs (ports)
   TLMXbuffers =: mode.TLMDmxBuffers            % max. buffers owned by TLIB
   TLMXqueuedTpdus =: mode.TLMDmxQueuedTpdus    % max. Q'd TPDUs in XMSG space

   TLPINIT(mode,TLdynamic,status)               % Always first TLIB call
   CHKstatus('TLPINIT',status)

   1 =: myReference                             % only one connection at a
                                                % time thus not used anyway!

                                                % start listening
   TLtlap2string =: serverAddress.TLAPformat    % address specified as string
   0 =: serverAddress.TLAPnetAddress.TLNMlength % don't specify own system
   SIZE serverName =: serverAddress.TLAPsuffix.TLNMlength  % server name
   serverName =: serverAddress.TLAPsuffix.TLNMstring(0:SIZE serverName-1)

   TLPSTLS(serverAddress,myReference,status,TLIBreference)
   CHKstatus('TLPstls',status)
   output(1,'A','Server is listening - name is  ')
   output(1,'A',serverName)

   TLinfiniteTime =: maxwaitTime.TLTMLENGTH     % wait forever for next event
   TLTMhrs =: maxwaitTime.TLTMUNITS

   TLanyRefno =: requestedEvent.TLEVrefno       % Allow any event to occur
   TLanyEvent =: requestedEvent.TLEVcode        % on any connection
```

```
  DO FOREVER
                                          % wait for next event
      TLPwait(maxwaitTime,requestedEvent,status,actualEvent)
      CHKstatus('TLPwait',status)
                                          % analyse next event
      IF actualEvent.TLEVcode = TLEVcnin THEN
          output(1,'A','$connection indication')
          IF (actualEvent.TLEVdata(0) =: lengthData) > 0 THEN % if user data
              TLlogicalAddress =: buffer.TLBFformat          % sent by client
              ADDR(inputBuffer(0)) FORCE INTEGER =: buffer.TLBFaddress % then
              TLPprbf(TLIBreference,buffer,lengthData,status) % provide an
              CHKstatus('TLPprbf',status)                    % input buffer
          ENDIF                                              % for it
          TLPcnin(TLIBreference,status,clientAddress,quality,&
                  credit,buffer,lengthData)
          CHKstatus('TLPcnin',status)
          IF lengthData > 0 THEN                          % if user data sent
              output(1,'A',' with user data "')           % by client then
              output(1,'A',inputBuffer(0:lengthData-1))   % display it
              output(1,'A','"')
          ENDIF
          TLlogicalAddress =: buffer.TLBFformat           % accept connection
          ADDR(greetings(0)) FORCE INTEGER =: buffer.TLBFaddress  % sending
          TLPcnrs(TLIBreference,quality,buffer,lengthData,status) % some user
          CHKstatus('TLPcnrs',status)                     % data
      ELSIF actualEvent.TLEVcode = TLEVdcin THEN
          output(1,'A','$disconnect reason')
          IF (actualEvent.TLEVdata(0) =: lengthData) > 0 THEN  % if user data
              TLlogicalAddress =: buffer.TLBFformat            % then provide
              ADDR(inputBuffer(0)) FORCE INTEGER =: buffer.TLBFaddress% buffer
              TLPprbf(TLIBreference,buffer,lengthData,status)
              CHKstatus('TLPprbf',status)
          ENDIF
          TLPdcin(TLIBreference,status,disconnectReason,& % get information
                  buffer,lengthData)                      % associated with
          CHKstatus('TLPcnin',status)                     % disconnection
          output(1,'I4',disconnectReason)
          IF lengthData > 0 THEN                          % if user data sent
              output(1,'A',' with user data "')           % by client, then
              output(1,'A',inputBuffer(0:lengthData-1))   % display it
              output(1,'A','"')
          ENDIF
      ELSE
          output(1,'A','$UNEXPECTED EVENT type')
          output(1,'I6',actualEvent.TLEVcode)
      ENDIF

  ENDDO

ENDROUTINE
ENDMODULE
%-------------------------------------------------------------------------------%
$EOF
```

1.4 The Client Program

```
%-------------------------------------------------------------------------------%
%                                                                               %
%                              TLP-CLIENT                                       %
%                                                                               %
%     A simple client illustrating the use of the PLANC interface to TLIB.      %
%     The client merely opens a connection to a server and then closes it.      %
%                                                                               %
%-------------------------------------------------------------------------------%
MODULE TLPEXAMPLE

$LIST OFF
$INCLUDE TLP:DEFS
$INCLUDE TLP:IMPT
$LIST ON

IMPORT (ROUTINE VOID,VOID: MON0)

$MACRO CHKstatus(text,status)          % define a macro for status checking
   IF status >< OK THEN                % in this simple example, all
       output(1,'A','$BAD STATUS from')% errors cause termination
       output(1,'A',"text")
       output(1,'I6',"status")
       MON0
   ENDIF
$ENDMACRO

%-------------------------------------------------------------------------------%
%                                                                               %
%                              GLOBAL DATA                                      %
%                                                                               %
%-------------------------------------------------------------------------------%

INTEGER ARRAY: stack(0:1000)                    % program stack
INTEGER ARRAY: TLdynamic(0:TLSZDYNAMIC-1)       % storage for TLIB
BYTES: ownName := 'CLIENT'                      % own port name
BYTES: serverSystem := 'SNORRE'                 % system on which server runs
BYTES: serverName := 'SERVER'                   % server port name
BYTES: greetings := 'greetings from client'     % data to go with connect
                                                % request
BYTES: farewell := 'bye'                        % data to go with disconnection

%-------------------------------------------------------------------------------%
%                                                                               %
%                              MAIN PROGRAM                                     %
%                                                                               %
%-------------------------------------------------------------------------------%

PROGRAM: CLIENT

INTEGER: TLIBreference                  % connection identifier required by TLIB
```

```
INTEGER: myReference                    % connection identifier used by program
INTEGER: credit                         % initial credit when connected
INTEGER: lengthData                     % length of data received
INTEGER: status                         % status from TLIB call
BYTES: inputBuffer(0:99)                % buffer for incoming data
TLMD: mode                              % used to specify TLIB operating mode
TLTM: maxWaitTime                       % used to specify max wait time
TLEV: requestedEvent                    % used to specify events of interest
TLEV: actualEvent                       % gives actual event that occurred
TLBFLOGICAL: buffer                     % used to describe a buffer
TLAP2STRING: myAddress                  % address of client
TLAP2STRING: serverAddress              % address of server as system/name
TLQS: quality                           % quality of service on connection
BOOLEAN: done                           % used to indicate when finished

   INISTACK stack
                                                % initialize TLIB
   TLXMinUserMode =: mode.TLMDxmsgmode          % XMSG in user mode
   TRUE =: mode.TLMDuniqueSuffix                % want own name to be unique
   TLMXconnections =: mode.TLMDmxConnections    % Max simultaneous connections
   TLMXaccessPoints =: mode.TLMDmxAccessPoints  % Max no of TLAPs (ports)
   TLMXbuffers =: mode.TLMDmxBuffers            % Max owned by TLIB
   TLMXqueuedTpdus =: mode.TLMDmxQueuedTpdus    % Max q'd TPDUs in XMSG space

   TLPINIT(mode,TLdynamic,status)               % Always first TLIB call
   CHKstatus('TLPINIT',status)

                                                % connect to server
                                                % set up own address
   TLtlap2string =: myAddress.TLAPformat        % address specified as string
   0 =: myAddress.TLAPnetAddress.TLNMlength     % do not specify own system
   SIZE ownName =: myAddress.TLAPsuffix.TLNMlength  % own port name
   ownName =: myAddress.TLAPsuffix.TLNMstring(0:SIZE ownName-1)
                                                % set up server address
   TLtlap2string =: serverAddress.TLAPformat    % address specified as string
   SIZE serverSystem =: serverAddress.TLAPnetAddress.TLNMlength
   serverSystem =: serverAddress.TLAPnetAddress.TLNMstring(0:SIZE serverSystem-1)
   SIZE serverName =: serverAddress.TLAPsuffix.TLNMlength
   serverName =: serverAddress.TLAPsuffix.TLNMstring(0:SIZE serverName-1)

   TLlogicalAddress =: buffer.TLBFformat        % buffer address is logical
   ADDR(greetings(0)) FORCE INTEGER =: buffer.TLBFaddress  % address in user
   TRUE =: quality.TLQSisDefault                           % space
   1 =: myReference                             % only one connection at a
                                                % time, thus not used anyway!
   output(1,'A','$connecting to server...')
   TLPCNRQ(myAddress,serverAddress,myReference,quality,buffer,&
           SIZE greetings,status,TLIBreference)
   CHKstatus('TLPcnrq',status)

                                                % set up to wait for events
   2 =: maxWaitTime.TLTMlength                  % max wait time of 2 minutes
   TLTMmins =: maxWaitTime.TLTMunits
                                                % wait for ANY event
   TLanyRefno =: requestedEvent.TLEVrefno
   TLanyEvent =: requestedEvent.TLEVcode
```

```
    FALSE =: done

    DO WHILE NOT done
                                                % wait for next event
        TLPWAIT(maxWaitTime,requestedEvent,status,actualEvent)
        CHKstatus('TLPwait',status)
                                                % analyse event that occurred
        IF actualEvent.TLEVcode = TLEVcncf THEN
            output(1,'A','connected')
            IF (actualEvent.TLEVdata(0) =: lengthData) > 0 THEN  % if user sent
                TLlogicalAddress =: buffer.TLBFformat            % data
                ADDR(inputBuffer(0)) FORCE INTEGER =: buffer.TLBFADDRESS
                TLPprbf(TLIBreference,buffer,lengthData,status)  % then provide
                CHKstatus('TLPprbf',status)                      % buffer for it
            ENDIF
            TLPcncf(TLIBreference,status,quality,credit,buffer,lengthData)
            CHKstatus('TLPcncf',status)
            IF lengthdata > 0 THEN                               % display user data
                output(1,'A',' with user data "')                % sent by server,
                output(1,'A',inputBuffer(0:lengthData-1))        % if any
                output(1,'A','"')
            ENDIF
                                    % no data to send, therefore disconnect
            output(1,'A','$disconnecting...')
            TLlogicalAddress =: buffer.TLBFformat
            ADDR(farewell(0)) FORCE INTEGER =: buffer.TLBFaddress
            TLPdcrq(TLIBreference,buffer,SIZE farewell,status)
            CHKstatus('TLPdcrq',status)
        ELSIF actualEvent.TLEVcode = TLEVdccf THEN           % disconnection
                                                             % complete
            output(1,'A','disconnect complete')
            TRUE =: done
        ELSE
            output(1,'A','$UNEXPECTED EVENT type')
            output(1,'I6',actualEvent.TLEVcode)
            TRUE =: done
        ENDIF
    ENDDO

ENDROUTINE

ENDMODULE
%-------------------------------------------------------------------------------%
$EOF
```

A P P E N D I X K

SAMPLE PROGRAMS USING TLIB/FORTRAN

1 Introduction

The following set of communicating programs, called SERVER and CLIENT, just gives an example of how a connection is established, and then immediately disconnected. The connection establishment and disconnection in the example in section 6.7 serve as an outline of what happens at runtime. It shows the interaction between the two programs.

Notice the inclusion of the file TLF:DEFS, which contains standard symbols. Also note that SERVER has to be started before CLIENT.

1.1 Brief Description of SERVER

The sequence of TLIB call are as follows:

1) The first call to TLIB is TLFINIT.

2) TLFSTLS starts listening on one connection. This allows possible clients to get a connection request through.

3) The TLFWAIT call brings SERVER into a waiting state. SERVER is now in principle open for any event (see "requested event definition"), but is really prepared for a connection indication from a client.

4) Assuming there is user data in the connection request from the client, SERVER invokes the TLFPRBF to provide a buffer. The connection indication event is processed by the TLFCNIN call.

5) SERVER will acknowledge the client's connection request by sending back a connection response (TLFCNRS). 'OK' is sent as user data.

6) SERVER goes into a waiting state (TLFWAIT), expecting a disconnect indication with user data.

7) A buffer is provided for the user data (TLFPRBF) and the disconnect indication is processed by invoking TLFDCIN.

1.2 Brief Description of CLIENT

The sequence of TLIB calls is as follows:

1) The first call to TLIB is TLFINIT.

2) A connection request is sent to the server (TLFCNRQ). This request contains a greeting, 'HELLO', as user data.

3) CLIENT enters a waiting state (TLFWAIT), expecting a connection confirmation back from the server.

4) Assuming that the server sends user data in the connection response, CLIENT provides a buffer (TLFPRBF). Then the connection confirmation is processed by the TLFCNCF call.

5) CLIENT finally disconnects by sending 'BYE' as user data in the TLFDCRQ call.

1.3 The Server Program

```
      PROGRAM SERVER
$LIST OFF
$INCLUDE TLF:DEFS
$LIST ON

C      declarations
      EXTERNAL TLFINIT,TLFCNRQ,TLFWAIT,TLFCNCF,TLFDCIN
C define array for use by TLIB
      INTEGER TLDYNAMIC(0:TLSZDYNAMIC-1)
C define array to store TLIB mode
      INTEGER MODE(0:TLALMODE-1)
C define array for timeout period on wait
      INTEGER MAXWAIT(0:TLALTIME-1)
C define arrays to store requested and actual events
      INTEGER REQUEST(0:TLALEVENT-1)
      INTEGER ACTUAL(0:TLALEVENT-1)
C define array to store user data on connection open and close
      INTEGER USERDATA(0:(TLMXUSERDATA/2)-1)
C storage for return status from TLIB calls
      INTEGER RETSTAT
C TLAP definitions
      INTEGER*4 USADD(0:TLALTLAP-1)
      INTEGER*4 IRMADD(0:TLALTLAP-1)
C network address and suffix definitions
      CHARACTER UN*(TLMXNAMELENGTH),US*(TLMXNAMELENGTH)
      EQUIVALENCE (USADD(TLAPNETADDRESS),UN),(USADD(TLAPSUFFIX),US)
C reply definition
      EQUIVALENCE (USERDATA(0),REPLY)
      CHARACTER REPLY*(6)
C quality of service
      INTEGER QOS(0:TLALQOS-1)

C initialize TLIB:
C XMSG is called in user mode
      MODE(TLMDXMSGMODE) = TLXMINUSERMODE
C and require the T-suffix names to be unique
      MODE(TLMDUNIQUESUFFIX) = TLISUNIQUE
C
      MODE(TLMDMXCONNECTIONS) = TLMXCONNECTIONS
      MODE(TLTLMXACCESSPOINTS) = TLMXACCESSPOINTS
      MODE(TLMDMXBUFFERS) = TLMXBUFFERS
      MODE(TLMDMXQUEUEDTPDUS) = TLMXQUEUEDTPDUS
      CALL TLFINIT(MODE,TLDYNAMIC,RETSTAT)
      IF (RETSTAT .NE. OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFINIT'
        GO TO 999
      ENDIF
C start listening on one connection
      IUSREF=1
C local address definition
      USADD(TLAPFORMAT)=TLTLAP2STRING
```

```
      USADD(TLAPLNNET)=6
      UN='SNORRE'
      USADD(TLAPLNSUFFIX)=6
      US='S-PORT'
C timeout definition
      MAXWAIT(TLTMUNITS)=TLTMMINS
      MAXWAIT(TLTMLENGTH)=10
C requested event definition
      REQUEST(TLEVREFNO)=TLANYREFNO
      REQUEST(TLEVCODE)=TLANYEVENT

      CALL TLFSTLS(USADD,IUSREF,RETSTAT,ITLREF)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFSTLS'
        GO TO 999
      ENDIF
      WRITE(1,*)'SERVER IS LISTENING'

C wait for TLEVCNIN
      CALL TLFWAIT(MAXWAIT,REQUEST,RETSTAT,ACTUAL)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFWAIT'
        GO TO 999
      ENDIF
      IF(ACTUAL(TLEVCODE).EQ.TLEVTIME) THEN
        WRITE(1,*)'TIMEOUT'
        GO TO 999
      ENDIF
      IF(ACTUAL(TLEVCODE).NE.TLEVCNIN) THEN
        WRITE(1,*)'UNEXPECTED EVENT CODE ',ACTUAL(TLEVCODE)
        GO TO 999
      ENDIF
      LDATA=ACTUAL(TLEVODATA)
      LDATA=((LDATA+1)/2)*2

C provide buffer and get user data
      CALL TLFPRBF(ITLREF,USERDATA,LDATA,0,RETSTAT)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFPRBF, CONNECTION  DATA'
        GO TO 999
      ENDIF
      CALL TLFCNIN(ITLREF,RETSTAT,IRMADD,QOS,ICRED,0,LENDAT)
      WRITE(1,*)'USERDATA:  '
      WRITE(1,100)REPLY
100   FORMAT(2X,A6)
C send reply
      REPLY='OK    '
      LDATA=6
      CALL TLFCNRS(ITLREF,QOS,USERDATA,LDATA,RETSTAT)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFCNRS'
        GO TO 999
      ENDIF
```

```
C wait for disconnect data
      CALL TLFWAIT(MAXWAIT,REQUEST,RETSTAT,ACTUAL)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS WHEN WAITING FOR DISCONNECT'
        GO TO 999
      ENDIF
      IF(ACTUAL(TLEVCODE).NE.TLEVDCIN) THEN
         WRITE(1,*)'UNEXPECTED EVENT CODE ',ACTUAL(TLEVCODE)
         GO TO 999
      ENDIF
      LDATA=ACTUAL(TLEVODATA)
      LDATA=((LDATA+1)/2)*2

C get buffer and user data
      CALL TLFPRBF(ITLREF,USERDATA,LDATA,0,RETSTAT)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLPRBF, DISCONNECT DATA'
        GO TO 999
      ENDIF
      CALL TLFDCIN(ITLREF,RETSTAT,IDCRSN,IDUM,LENDAT)
      WRITE(1,*)'USERDATA:  '
      WRITE(1,100)REPLY
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFDCIN'
      ENDIF

999   END
```

1.4 The Client Program

```
      PROGRAM CLIENT
$LIST OFF
$INCLUDE TLF:DEFS
$LIST ON

C      declarations
      EXTERNAL TLFINIT,TLFCNRQ,TLFWAIT,TLFCNCF,TLFDCIN
C define array for use by TLIB
      INTEGER TLDYNAMIC(0:TLSZDYNAMIC-1)
C define array to store TLIB mode
      INTEGER MODE(0:TLALMODE-1)
C define array for timeout period on wait
      INTEGER MAXWAIT(0:TLALTIME-1)
C define arrays to store requested and actual events
      INTEGER REQUEST(0:TLALEVENT-1)
      INTEGER ACTUAL(0:TLALEVENT-1)
C define array to store user data on connection open and close
      INTEGER USERDATA(0:(TLMXUSERDATA/2)-1)
C storage for return status from TLIB calls
      INTEGER RETSTAT
C TLAP definitions
      INTEGER*4 REMADD(0:TLALTLAP-1)
      INTEGER*4 USADD(0:TLALTLAP-1)
C network address and suffix definitions
      CHARACTER RN*(TLMXNAMELENGTH),RS*(TLMXNAMELENGTH),
     +UN*(TLMXNAMELENGTH),US*(TLMXNAMELENGTH)
      EQUIVALENCE (REMADD(TLAPNETADDRESS),RN),(REMADD(TLAPSUFFIX),RS)
      EQUIVALENCE (USADD(TLAPNETADDRESS),UN),(USADD(TLAPSUFFIX),US)
      CHARACTER NETNAME*(TLMXNAMELENGTH),SUFFNAME*(TLMXNAMELENGTH)
      CHARACTER NETNAM1*(TLMXNAMELENGTH),SUFFNA1*(TLMXNAMELENGTH)
C message definition
      CHARACTER MESS*(6)
      EQUIVALENCE (USERDATA(0),MESS)
C quality of service
      INTEGER QOS(0:TLALQOS-1)

C initialize TLIB:
C XMSG is called in user mode
      MODE(TLMDXMSGMODE) = TLXMINUSERMODE
C and require the T-suffix names to be unique
      MODE(TLMDUNIQUESUFFIX) = TLISUNIQUE
C
      MODE(TLMDMXCONNECTIONS) = TLMXCONNECTIONS
      MODE(TLTLMXACCESSPOINTS) = TLMXACCESSPOINTS
      MODE(TLMDMXBUFFERS) = TLMXBUFFERS
```

```
      MODE(TLMDMXQUEUEDTPDUS) = TLMXQUEUEDTPDUS
      CALL TLFINIT(MODE,TLDYNAMIC,RETSTAT)
      IF (RETSTAT .NE. OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFINIT'
        GO TO 999
      ENDIF
C remote address definition
      REMADD(TLAPFORMAT)=TLTLAP2STRING
      REMADD(TLAPLNNET)=6
      RN='SNORRE'
      REMADD(TLAPLNSUFFIX)=6
      RS='S-PORT'
C local address definition
      USADD(TLAPFORMAT)=TLTLAP2STRING
      USADD(TLAPLNNET)=6
      UN='SNORRE'
      USADD(TLAPLNSUFFIX)=6
      US='C-PORT'
C timeout definition
      MAXWAIT(TLTMUNITS)=TLTMMINS
      MAXWAIT(TLTMLENGTH)=10
C requested event definition
      REQUEST(TLEVREFNO)=TLANYREFNO
      REQUEST(TLEVCODE)=TLANYEVENT

C send connection request
      MESS='HELLO '
      IUSREF=1
      LENDAT=6
      QOS(TLQSISDEFAULT)=TLISDEFAULT
      CALL TLFCNRQ(USADD,REMADD,IUSREF,QOS,USERDATA,LENDAT,RETSTAT,
     +ITLREF)
      IF(RETSTAT.NE.OK)THEN
         WRITE(1,*)'BAD STATUS FROM TLFCNRQ'
        GO TO 999
      ENDIF
C wait for connection confirmation
      CALL TLFWAIT(MAXWAIT,REQUEST,RETSTAT,ACTUAL)
      IF(RETSTAT.NE.OK) THEN
        WRITE(1,*)'BAD STATUS FROM TLFWAIT'
        GO TO 999
      ENDIF
      IF(ACTUAL(TLEVCODE).EQ.TLEVTIME) THEN
        WRITE(1,*)'TIME OUT'
        GO TO 999
      ENDIF
      IF(ACTUAL(TLEVCODE).NE.TLEVCNCF) THEN
        WRITE(1,*)'UNEXPECTED EVENT CODE',ACTUAL(TLEVCODE)
          IF(ACTUAL(TLEVCODE).EQ.TLEVDCIN) THEN
            CALL TLFDCIN(ITLREF,ISTAT,IDCRSN,IUBFID,LENDAT)
            WRITE(1,*)'DISCONNECT REASON',IDCRSN
          ENDIF
        GO TO 999
      ENDIF
      LDATA=ACTUAL(TLEVDATA)
      LDATA=((LDATA+1)/2)*2
```

```
C provide buffer and get user data
      CALL TLFPRBF(ITLREF,USERDATA,LDATA,0,RETSTAT)
      IF(RETSTAT.NE.OK)THEN
         WRITE(1,*)'BAD STATUS FROM TLFPRBF'
        GO TO 999
      ENDIF
      CALL TLFCNCF(ITLREF,RETSTAT,IQOS,ICRED,0,LENDAT)
      IF(RETSTAT.NE.OK)THEN
         WRITE(1,*)'BAD STATUS FROM TLFCNCF'
        GO TO 999
      ENDIF
      WRITE(1,*)'USERDATA '
      WRITE(1,200)MESS
200   FORMAT(2X,A6)

C send disconnect request
      MESS='BYE    '
      LENDAT=6
      CALL TLFDCRQ(ITLREF,USERDATA,LENDAT,RETSTAT)
      IF(RETSTAT.NE.OK)THEN
         WRITE(1,*)'BAD STATUS FROM TLFDCRQ'
      ENDIF

999 END
```

Index

********** **SEND US YOUR COMMENTS!!!** **********

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you

* find errors
* cannot understand information
* cannot find information
* find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

********** **HELP YOURSELF BY HELPING US!!** **********

Manual name: COSMOS Programmer Guide

Manual number: ND–60.164.3 EN

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

_____

Do you have suggestions for improving this manual ? _____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for ? _____

_____

NOTE!

This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

→

Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

Answered by _____ Date _____

Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway