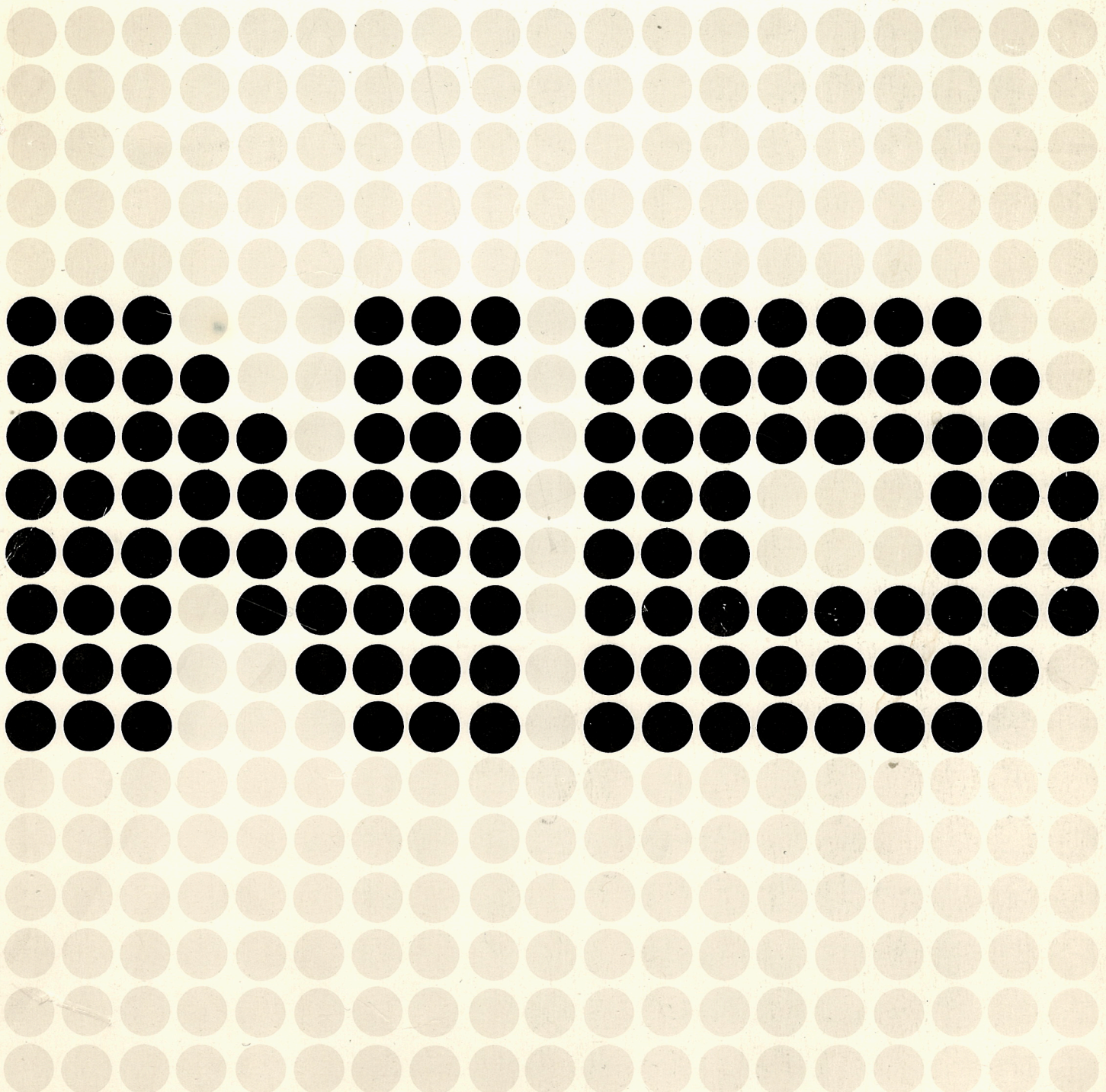


ND TPS
User's Guide

ND-60.111.03

NORSK DATA A.S



ND TPS User's Guide

ND-60.111.03

NOTICE

The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

The information described in this document is protected by copyright. It may not be photocopied, reproduced or translated without the prior consent of Norsk Data A.S.

Copyright © 1982 by Norsk Data A.S.

ND TPS User's Guide
Publ. No. ND-60.111.03

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 ND Bulletin 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, together with all types of inquiry and requests for documentation should be sent to the local ND office or (in Norway) to:

Documentation Department
Norsk Data A.S
P.O. Box 4, Lindeberg gård
Oslo 10

PREFACE

THE PRODUCT

This manual describes the ND Transaction Processing System ND TPS, version D

ND TPS ND-10101 D

ND TPS is a general transaction processing system that initiates and controls transactions between various input/output devices and application programs. TPS provides facilities for handling terminals, data communications, data bases, etc.

ND TPS systems are tailored for individual user configurations, specified when ordering ND TPS. However, all systems contain the basic TPS modules.

Special systems may contain such additional features as:

ND-500 TPS System Modules	ND-10342
Input/output Modules	ND-10105 to ND-10108
Multi - CPU Systems	
Non-standard System Generation Parameters	

THE READER

ND TPS User's Guide is written for programmers who write application programs to be run under ND TPS. These programs can be written in FORTRAN, COBOL, PLANC, NPL and MAC.

System designers who design application systems to be run under TPS will also find the material in this manual of interest.

PREREQUISITE KNOWLEDGE

Chapter 1 of this manual is an introduction to ND TPS and should give the necessary background in TPS to go on to the following chapters. A more detailed description may be found in

ND TPS General Description, ND-60.105

In addition, the reader should also be familiar with the SINTRAN III operating system, the SIBAS data base system and the screen handling systems. General descriptions of these systems are found in:

Introduction to SINTRAN III, ND-60.125

SIBAS II User's Manual, Chapter 1, ND-60.127

The NORD Screen Handling System, chapter 1, ND-60.088

FOCUS Screen Handling System, chapter 1, ND-60.137

THE MANUAL

ND TPS User's Guide can be divided into three parts. Chapter 1 is an introduction which should be read first if the reader is not already familiar with ND TPS. The main body of the manual consists of chapters 2—8. These chapters may be read independently of each other and in any order; each chapter treats one topic in a tutorial manner and should be read sequentially. Finally, the appendices, especially appendix H, TSR call formats, are reference material.

The manual covers all aspects of TPS of interest to the application programmer, both special TPS features, such as session communication, and the interface to other systems used by TPS application programs, such as SINTRAN, SIBAS, FOCUS, NSHS, FORTRAN, COBOL, PLANC, NPL and MAC. The manual, however, does not go into the details of these other systems and the reader is referred to the manuals for the individual systems for these details.

RELATED MANUALS

The following manuals describe the systems of greatest interest to the TPS application programmer:

- SINTRAN III Timesharing Guide, ND-60.132
- SINTRAN III Reference Manual, ND-60.128
- SINTRAN III Real-Time Guide, ND-60.133
- NORD Screen Handling System, ND-60.088
- FOCUS Screen Handling System, ND-60.137
- NORD FORTRAN Reference Manual, ND-60.145
- NORD COBOL Reference Manual, ND-60.144
- NORD-PL User's Guide, ND-60.047
- MAC User's Guide, ND-60.096
- SIBAS II User's Manual, ND-60.127
- Symbolic Debugger - User's Guide, ND-60.158

For ND-500:

- ND-500 LOADER/MONITOR, ND-60.136

Other manuals describing ND TPS are:

- ND TPS General Description, ND-60.105
- ND TPS System Supervisor's Guide, ND-30.006

TABLE OF CONTENTS

+ + +

<i>Section:</i>	<i>Page:</i>
1 INTRODUCTION	1-1
1.1 What is ND TPS?	1-1
1.2 The Structure of TPS	1-2
1.2.1 Transaction Control.....	1-4
1.2.1.1 Transaction Control Modules	1-4
1.2.1.2 Transaction Processing Tasks	1-4
1.2.1.3 Transaction Service Routines.....	1-5
1.2.1.4 Application Programs	1-6
1.2.1.5 Special Applications.....	1-7
1.2.2 Handling Input/Output.....	1-9
1.2.2.1 Standard Devices	1-9
1.2.2.2 Special TPS Devices - Input/Output Modules.....	1-10
1.2.2.3 The NSHS Screen Handling System.....	1-11
1.2.2.4 The FOCUS Screen Handling System	1-11
1.2.2.5 The SIBAS Data Base Management System.....	1-11
1.2.2.6 Checkpoint and Restart.....	1-12
1.2.3 Operator Communication	1-13
1.2.4 Message Routing and Queuing	1-13
1.3 Controlling Transactions	1-14
1.3.1 Type 1: Permanent Terminal Transactions	1-14
1.3.2 Type 2: Short Terminal Transactions.....	1-18
1.3.3 Type 3: Short Local Terminal Transactions.....	1-18
1.3.4 Type 4: Concurrent Transactions.....	1-19
1.3.5 Type 5: Future and Periodic Transactions	1-19
1.4 Processing a Transaction	1-20
1.4.1 Starting the Transaction	1-20
1.4.2 Processing the Transaction	1-22
1.4.3 Terminating the Transaction.....	1-22
2 ADMINISTRATING TASKS.....	2-1
2.1 Tasks, Transactions and Applications.....	2-1
2.2 Starting Tasks and Switching Applications.....	2-4
2.2.1 Immediate Task Activation	2-5
2.2.1.1 TACTV - The Activate Concurrent Task TSR	2-5

<i>Section:</i>	<i>Page:</i>
2.2.2	Future and Periodic Task Activation.....2—7
2.2.2.1	TASET — The Set Execution Time TSR.....2—7
2.2.2.2	TINTV — The Set Interval TSR2—9
2.2.2.3	TDCNT — The Disconnect Application TSR.....2—10
2.2.3	Switching to Another Application2—11
2.2.3.1	TSWAP—The Switch Application Program TSR2—11
2.2.4	The SIGNON and SELECT Special Applications.....2—12
2.3	Terminating Transactions.....2—13
2.3.1	Normal Termination2—13
2.3.1.1	TSTOP - The STOP Transaction TSR2—13
2.3.1.2	TTERM - The Terminate Task TSR2—14
2.3.1.3	The SIGNOFF Special Application2—14
2.3.1.4	TSTST - The Set Termination Strategy TSR.....2—15
2.3.1.5	TSCST - The Set Close Strategy TSR.....2—16
2.3.2	Abnormal Termination2—17
2.3.2.1	The ABEND Special Application2—17
2.3.2.2	TSAST - The Set Abend Strategy TSR.....2—17
2.3.2.3	Illegal Monitor Calls2—18
2.3.2.4	Timeout.....2—18
3	INPUT/OUTPUT PROCESSING.....3—1
3.1	SIBAS Under TPS3—1
3.1.1	Data Definition and Manipulation3—1
3.1.2	The SIBAS Interface Routine3—2
3.1.3	Opening and Closing the Data Base3—3
3.1.4	Using More Than One Data Base.....3—3
3.1.5	SIBAS in ND-500 Multi-CPU TPS3—5
3.1.6	Restricted SIBAS Calls3—6
3.2	NSHS and FOCUS Under TrS.....3—8
3.2.1	Handling Display Terminals3—8
3.2.2	The NSHS System3—8
3.2.2.1	Defining and Using Pictures.....3—8
3.2.2.2	Q °Q°Q° and Restart.....3—9

<i>Section:</i>	<i>Page:</i>
3.2.3	FOCUS Level 13—11
3.2.3.1	Defining and Using Forms3—11
3.2.3.2	Local or Remote Asynchronous Terminals3—12
3.2.3.3	Synchronous/Buffered Terminals Using FOCUS3—13
3.2.3.4	ND-100 — ND-500 incompatibilities in FOCUS3—13
3.3	Special TPS Devices3—14
3.3.1	Session Request from a Device3—15
3.3.2	TSOPN — The Open - Session TSR3—17
3.3.3	Session Request from an Application3—19
3.3.4	TSCLO — The Close Session TSR3—22
3.3.5	TSEST — The Session Status TSR3—23
3.3.6	TSMMSG — The Send-Message TSR3—24
3.3.7	TRMSG — The Read Message TSR3—25
3.3.8	TPASZ — The Set Packet Size TSR3—26
3.3.9	Restart3—26
3.3.10	Available Input/Output Modules3—27
3.3.10.1	X25LAPB3—27
3.3.10.2	IBM—3270—CU3—32
3.3.10.3	IBM—3270—HOST3—34
3.3.10.4	ISO—1745—HOST3—35
3.4	Standard Devices and Files3—36
3.4.1	Allocating Standard Devices and Files3—36
3.4.2	Unavailable Devices and Files3—37
3.4.3	Accessing Standard Devices and Files3—39
3.4.4	Restart3—40
4	OTHER TPS AND SINTRAN FACILITIES4—1
4.1	Message Handling4—2
4.1.1	TWMSG and CWMSG - The Write Message to Operator TSR4—3
4.1.2	TBRDC - The Broadcast Message TSR4—4
4.1.3	TTEXT - The Send Text Message TSR4—5
4.1.4	TGBRD and CGBRD - The Get Broadcasted Message TSR ...4—6
4.1.5	Monitor Calls (ERMSG,QERMS,ERMON)4—7
4.2	Clock Routines4—9
4.3	The HOLD Monitor Call.....4—9
4.4	Semaphores and Internal Devices4—10

<i>Section:</i>	<i>Page:</i>
5	CHECKPOINT—RESTART5—1
5.1	Protecting the Database5—1
5.2	Preventive Facilities5—3
5.2.1	Backup5—3
5.2.2	Data Base Logging5—4
5.2.3	Synchronised Checkpoints5—5
5.2.3.1	TTSYN - The Allow-Synchronised Checkpoint TSR5—5
5.2.3.2	THSYN - The Hold Synchronised Checkpoint TSR5—6
5.2.3.3	TCHK - The Take Synchronised Checkpoint TSR5—6
5.2.4	Transaction Checkpoints5—7
5.2.4.1	TTRAN - The Take Transaction Checkpoint TSR5—8
5.3	Restart Facilities5—10
5.3.1	Rollback and Recovery5—10
5.3.2	Restarting TPS5—13
5.3.2.1	The RESTART Special Application5—15
5.3.2.2	TSRST - The Set Restart Strategy TSR5—16
6	SPECIAL APPLICATIONS6—1
6.1	SIGNON and SELECT6—3
6.1.1	SIGNON6—3
6.1.2	SELECT6—4
6.1.3	The Access Control System6—6
6.2	SIGNOFF, ABEND and RESTART6—7
6.2.1	SIGNOFF6—7
6.2.2	ABEND6—8
6.2.2.1	The Abend Error Message6—9
6.2.3	RESTART6—10
6.2.4	Summary of Termination, Abend and Restart Strategies6—13
6.3	TPOPEN and TPCLOSE6—14
6.4	CHECKPOINT, ROLLBACK and RECOVER6—14

<i>Section:</i>	<i>Page:</i>
7	SPECIAL CONSIDERATIONS7-1
7.1	Data Areas in the ND1007-1
7.1.1	The Variable Data Area in the ND1007-3
7.1.2	The Task Common Data Area in the ND1007-3
7.1.3	The Local Data Area in the ND1007-5
7.1.4	The Size of the Data Area in the ND1007-7
7.2	Data areas in the ND5007-8
7.3	Language Dependent Considerations7-9
7.3.1	FORTTRAN/PLANC in ND1007-10
7.3.2	COBOL in ND1007-11
7.3.3	MAC-NPL7-11
7.4	Program Structure7-12
7.4.1	Application Names and Numbers7-12
7.4.2	Subroutines7-13
7.5	Efficiency7-15
7.5.1	ND500 Efficiency7-15
7.5.2	Taking Checkpoints7-16
7.5.3	Opening and Closing the Data Base7-16
7.5.4	The Working Set7-17
7.6	Real Time Versus Background7-18
7.7	Pictures for NSHS in ND1007-19
7.7.1	Defining Private Pictures for NSHS7-19
7.7.2	Producing Public Pictures for NSHS7-21
7.7.3	Loading Public Pictures for NSHS7-22
7.8	Pictures for FOCUS7-23
7.8.1	Public pictures for FOCUS7-23
8	COMPILING AND LOADING PROGRAMS8-1
8.1	Testing of ND500 applications8-1
8.2	Background Testing of ND100 applications8-2

<i>Section:</i>		<i>Page:</i>
8.2.1	The TPS Background System	8—2
8.2.2	Available Facilities in the TPS Background System	8—3
8.2.3	The Load-Common and Save-Common Routines	8—4
8.2.4	Running the Background System	8—5
8.2.5	Testing in Background Mode	8—6
8.2.6	Compile and Load Examples	8—8
8.3	Real Time Programs	8—10
8.3.1	The Loading Procedure	8—11
8.3.2	Programs and Files Required	8—16
8.3.3	Compile and Load Example	8—17

<i>Appendix:</i>		<i>Page:</i>
A	APPLICATION NUMBERS FOR SPECIAL APPLICATIONS	A—1
B	SAMPLE PROGRAMS	B—1
C	ERROR MESSAGES	C—1
D	TPS SEGMENT STRUCTURE IN ND100	D—1
E	MONITOR CALLS AND LIBRARY CALLS	E—1
F	SCREEN—HANDLING CALLS	F—1
G	SIBAS CALLS	G—1
H	TSR CALL FORMATS	H—1
I	TSR CALLS—FUNCTIONAL LIST	I—1
J	TPS ON ND-500	J—1
K	GLOSSARY	K—1
	INDEX	

NOTATIONS

TSR Calls

The detailed formats of these calls, with complete parameter descriptions are given in appendix H. Chapters 2—5 discuss the calls and their use in a tutorial manner but do not contain detailed parameter descriptions. On the other hand, examples of the use of these TSRs are given in greater detail in chapters 2—5 than in appendix H. In both places the call formats and the examples are given first in FORTRAN and then in COBOL.

Examples

As for the TSR examples mentioned above, all examples are given first in FORTRAN and then in COBOL. Comments are included in some of the examples and these are written on the same line as program statements to save space, although this may not be allowed by the compiler. FORTRAN parameters are given sometimes as variables, sometimes as literals where this is allowed.

In the examples of conversational interaction with a program, input to the program is underlined.

Symbols

ESC indicates the escape key on a terminal

 indicates carriage return, line feed

< > indicates a parameter. Optional parameters or parameters with default values are indicated in the same way, but the default value is given under "rules". If one of several alternative values is to be given, this is also indicated under "rules".

1 INTRODUCTION

1.1 WHAT IS ND TPS?

ND TPS is a transaction processing system for the NORD family of computers. A transaction processing system may be defined as a computerised on-line system that allows the user to process data and update a data base as soon as information arrives and to retrieve the information as soon as he needs it.

The user will normally have a terminal available that is online to the data base. He will enter the transaction input on the terminal, the system will start the processing program (application program), the program will access the data base and send the user a response within seconds.

A transaction may be an inquiry which only reads the data, formats it and sends it to the terminal (*inquiry* transaction). The transaction may update the data base, perhaps after a conversational interaction between the program and the user (*up-date* transaction). The transaction may gather in data interactively and store it in a temporary file for later batch updating (*data entry*). A transaction may also generate a relatively large amount of output to a printer (*report generation*).

Most transactions are characterised by a fairly small amount of input and output, conversational interaction, short duration and fast response times, although none of these characteristics are absolute.

To accomplish this, a transaction processing system must provide facilities for handling the following main tasks:

- starting, controlling and terminating transactions
- communicating with terminals and other I/O devices belonging to the external environment, including routing messages to the correct destinations
- accessing the data base, including reading it and updating it

A short description of the structure of TPS and how it handles these tasks follows.

1.2 THE STRUCTURE OF TPS

The main tasks of TPS are, as mentioned above, controlling transaction start/termination, controlling the external environment and accessing the data base. In addition, provision must be made for starting, stopping and controlling the TPS system itself. Also, facilities must be available for simple and efficient communication between modules of TPS. The TPS system is therefore composed of the following types of modules:

(See figure 1.1)

- transaction control and service routines
- SIBAS data base control routines (separate subsystem)
- NSHS or FOCUS screen handling system (separate subsystem)
- input/output modules
- operator communication
- message queuing and routing routines

The application programs themselves are mainly user-written, but TPS includes a number of

- special application programs

to carry out user-oriented system functions, such as terminal operator sign-on, program abnormal-end, etc.

TPS exists as a basic modules plus a number of options. The basic modules consist of the transaction control and service routines, the SIBAS data base system, the NSHS or FOCUS screen handling system, operator communication, the message queuing and routing routines and a standard set of special applications. In the default version of TPS, all of these are run in a single CPU.

TPS options include multi-CPU systems, using both ND-100s and ND-500s, several transaction control modules and input/output modules for special devices, networks and distributed processing.

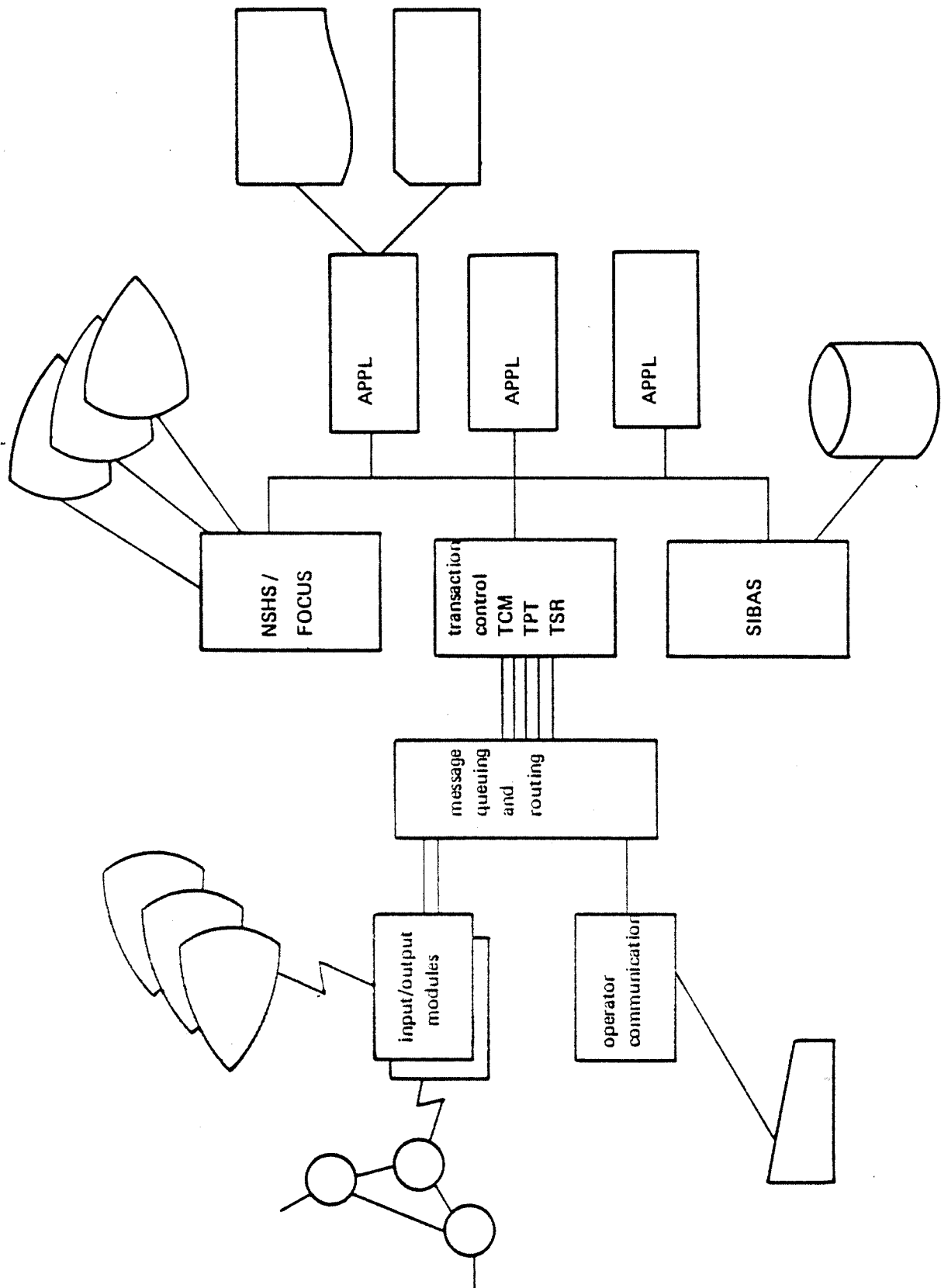


Figure 1.1: TPS Modules

1.2.1 Transaction Control

1.2.1.1 Transaction Control Modules

Transaction control is handled by one or more *transaction control modules* (TCMs). The TCM supervises the application programs belonging to it and controls system functions such as start, stop and checkpoint.

1.2.1.2 Transaction Processing Tasks

Each TCM has a number of *transaction processing tasks* (TPTs). (See Figure 1.2) The TPTs are a set of identical programs belonging to pools, one pool for each TCM. Each TPT is one unit with a TPT unit number. When a transaction is started, TCM may allocate a free TPT to the transaction from the pool and start the TPT and when the transaction is finished, TCM may free the TPT. Some TPTs may be permanently allocated to terminals and can process many transactions in a row.

The TPT has several functions:

- to start the application program used by the transaction
- to terminate the application program when it is done and either switch to a new application program or terminate the transaction
- to provide the application program with data areas (all application programs are reentrant and thus may not be written into)
- to provide checkpoint/restart facilities for the application program in case of system failure

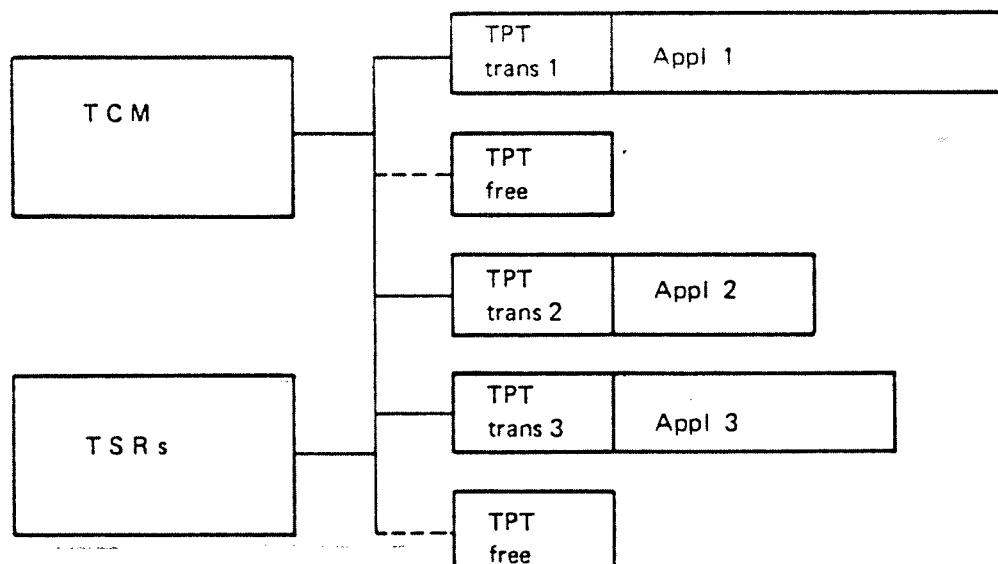


Figure 1.2: Transaction Control

1.2.1.3 Transaction Service Routines

The *transaction service routines* (TSRs) are a set of routines supplied with TPS to assist the application programmer in performing functions such as administrating task control, communicating with I/O devices and sending messages. These routines allow the programmer to concentrate on the applications as such, without having to be concerned with the complex details of a real-time environment. The routines represent a clean and logical interface between application programs and ND TPS (*See Figure 1.3*).

TSRs may be arranged in groups as follows:

- 1 Administrative services.
 - Switch control to another application.
 - Activate concurrent application.
 - Stop transaction.
 - Set termination/abend/restart strategy/close strategy
 - Set execution time/interval
- 2 Session services.
 - Read message.
 - Send message.
 - Open/close session.
- 3 Checkpoint control.
 - Take checkpoint.
 - Allow/prevent checkpoint.
- 4 Message services.
 - Write message on operator console.
 - Broadcast message to terminals.
- 5 Special Application TSRs.
 - Restart
 - Read status
 - Read configuration information
 - Operator functions

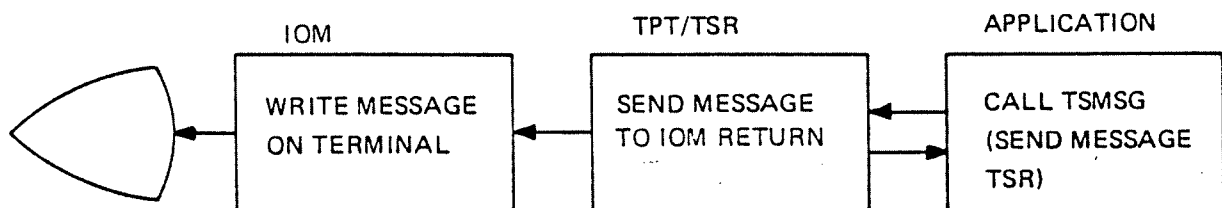


Figure 1.3: Using a TSR

1.2.1.4 Application Programs

The *application programs* do the actual processing of the transactions. They are mainly user-written, with the exception of a number of special applications supplied with the TPS system. They may be written in FORTRAN, COBOL, PLANC (a system oriented high level language), NPL (a ND-100 machine oriented medium level language) or MAC (assembly language) and they may use most of the facilities available to these languages.

Application programs are reentrant and may be used by several transactions simultaneously without having more than one copy. To achieve this, the ND100-FORTRAN compiler must be set in reentrant mode when compiling, and the ND100-COBOL compiler must be set in separate code-data mode when compiling. No special commands have to be given to the ND500-compilers.

The TPTs are also mainly reentrant, with only a small non-reentrant part for each TPT plus the data areas. Thus, there may be many TPTs without taking more than a minimum of space.

The maximum size of application programs and data areas in the ND-100 are user-dependent configuration parameters. Typical sizes are 24K words for programs and 5K words for data. The corresponding limits for the ND-500 are up to 134 megabytes of program and data.

Application programs in ND-100 may be tested as timesharing programs under SINTRAN before being run under TPS. A special set of routines is available to simulate a real-time TPS environment. SIBAS can be accessed by both TPS, timesharing and batch programs at the same time. In the ND-500, the application programs may be tested by running the ND Symbolic Debugger «live» in an ordinary TPS-run. (See Chapter 8.1.)

1.2.1.5 SPECIAL APPLICATIONS

Certain functions of the TPS system are not handled by the internal modules of TPS but are carried out as applications. They are started and terminated as any other application, under the control of a TPT.

The main reason is that application programs are easy to write, modify and load. Users can easily tailor these programs to their own needs. Functions that are common to many users, but where the detailed processing may vary from user to user, such as signon, transaction abend and restart, are carried out by special applications. Also, special system functions such as checkpoint, rollback and recovery are controlled by these applications.

A complete set of special application programs is supplied with TPS and many users will find that their needs are fully satisfied by these standard versions. Other users will modify the standard versions, while some users may wish to write their own versions.

The main purpose of the following special applications is communication with SIBAS. They are called by only one TPT (a special TPS system TPT) when the system uses them (*See Figure 1.4*)

- TPOPEN, called when TPS is initially started. This application may open the data base for general use. It may also start up transactions and broadcast a start message to terminals controlled by IOMs.
- TPCLOSE, called when TPS is closed or abnormally ended. The application may close the data base.
- CHECKPOINT, called when a synchronised checkpoint is taken. The application calls the SIBAS checkpoint routine.
- ROLLBACK, called when a system failure has occurred and the system is to be rolled back to a synchronised checkpoint. The application supervises the SIBAS rollback routine.
- RECOVER, called when a system failure has occurred to restore the system to its state at the latest transaction checkpoints. The application supervises the SIBAS recover routine.

Additional special applications, activated for each individual TPT, are:

- SIGNON, called to check the terminal operator's status and password and to reserve the terminal
- SELECT, called to determine which processing application is to be given control.

- TPMON, called when an ND-500 application is started. Thereafter TPMON administrates the ND-500 process. Further description can be found in Appendix J.
- SIGNOFF, called when a transaction terminates.
- ABEND, called when a transaction terminates abnormally due to an error situation in the program itself or an error return from a system routine.
- RESTART, called after a rollback or recovery has been performed. The application is called by each active TPT for the purpose of restarting the TPT's application at the correct point.

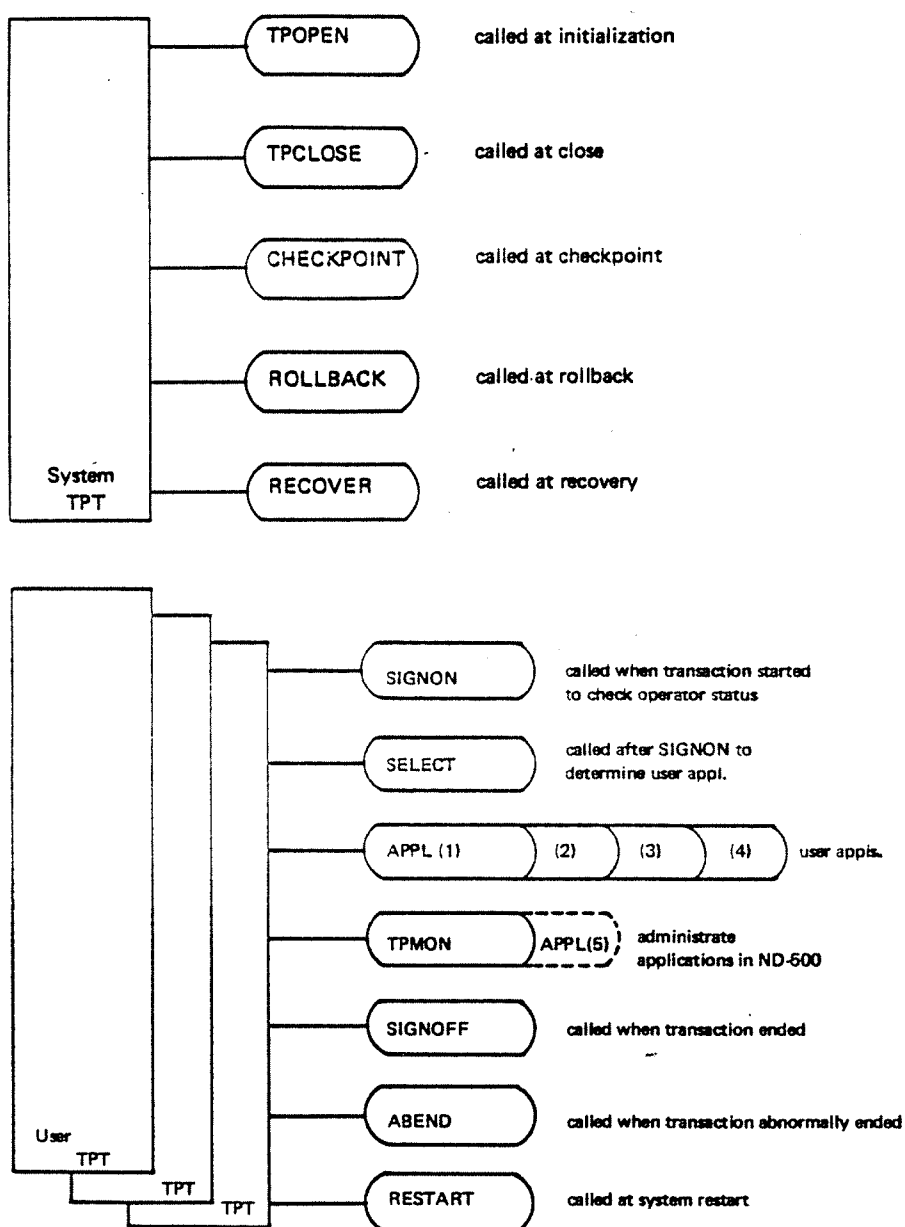


Figure 1.4: Special Applications

1.2.2 Handling Input/Output

TPS has facilities for handling input/output from many types of terminals and I/O devices and from data bases.

The data bases are controlled through the SIBAS data base management system.

Most terminal types can be handled through the screen handling system.

Other I/O devices can be divided into two main types:

- *standard devices* allocated to and controlled by a particular application program. Examples are standard SINTRAN terminals, printers, card readers and non-SIBAS files.
- *special TPS devices* that are allocated to TPS modules, not individual applications. Examples are networks and synchronous terminals.

1.2.2.1 Standard Devices

These devices are controlled by application programs directly through SINTRAN. Devices belonging to this group are generally available to all users of the computer system, also non-TPS users. They may include printers, spooling files, card readers, magnetic tapes, disk files etc. They are allocated to an application program when the program requests them and released by the program when it no longer needs them. The application program will access the devices through the standard routines available in the language it is written in, such as OPEN, CLOSE, READ, WRITE, etc.

1.2.2.2 Special TPS Devices - Input/Output Modules

Devices in this group are controlled by special programs called *Input/Output Modules* (IOMs). There is one IOM for each type of device, although each IOM may control many devices. Each device is one unit with a device unit number.

Input/output modules are used to control devices that cannot be controlled directly by an application program. This may be because their control is too difficult for the application program, for example non-standard devices or devices with complicated communication protocols. However, the main use for IOMs is in connection with *networks* and other types of multiplexed connections.

A network does not belong to any one application program but may have many terminals connected to many application programs (See Figure 1.5). The connection between the terminal and the application program is not direct, but goes through modems, concentrators, etc. Another type of network connection may be the connection between two TPS systems at different processing sites (See Figure 1.6). Application programs at one site may communicate over the network with application programs at the other site.

An application program communicates with these devices through the transaction service routines provided by TPS. A *session* is established between the device and the application program and the application program can then send messages to and receive messages from its session partner by calling these routines. Sessions may also be established between two application programs, either at the same processing site or across an external network.

Sessions may be broken and new sessions established. A transaction is only allowed to have one session at a time but it may possess several local devices, including local terminals, at the same time.

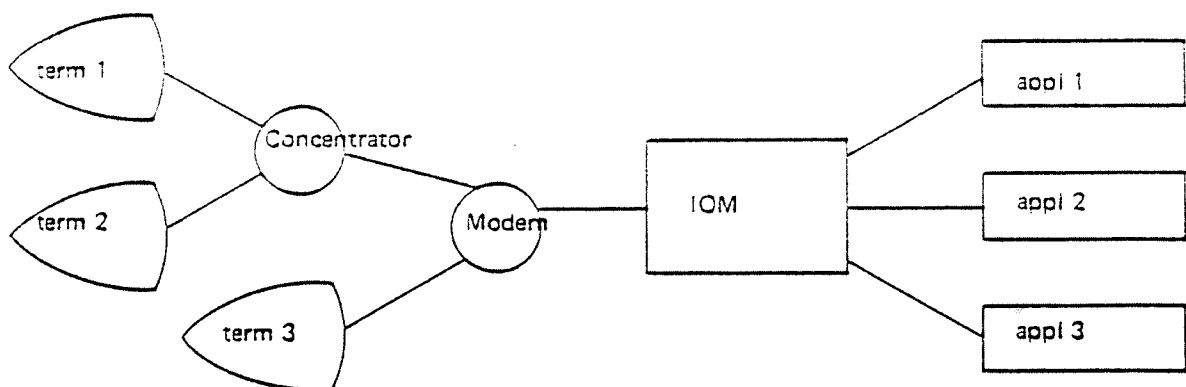


Figure 1.5: A Terminal Network

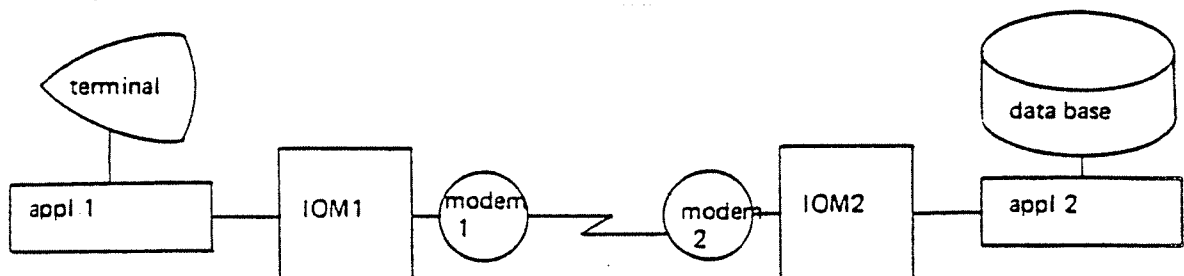


Figure 1.6: Accessing a Remote Database

1.2.2.3 The NSHS Screen Handling System

Terminals can be controlled by application programs through the NORD screen handling system, NSHS, which contains routines for formatting output pictures, reading input, field definition, cursor control, etc.

Terminals controlled by NSHS may be standard terminals or they may be special devices controlled by input/output modules. The applications programmer, however, does not have to know which group a terminal belongs to since the same NSHS calls are used for all types of terminals.

1.2.2.4 The FOCUS Screen Handling System

The Focus Screen Handling System can be used to control asynchronous as well as synchronous terminals. The processing part can be distributed to one or several Front End CPU's residing in one TPS system. The communication is transparent to the user programs (*See Section 3.2*)

1.2.2.5 The SIBAS Data Base Management System

SIBAS is a DBMS that provides most of the capabilities specified by the CODASYL committee for a data base facility in COBOL. Similar facilities are available to FORTRAN, PLANC, NPL and MAC programmers.

SIBAS allows direct and fast access to all data. It provides several methods of file organisation and access, separation of physical and logical organisation, concurrent or exclusive access and data independence. It has facilities for backup and restart to insure data integrity and privacy locks to prevent unauthorised access.

The data base is defined and created using the SIBAS data definition/redefinition language DRL. This is done independently of TPS.

The data base is accessed from application programs using the SIBAS data manipulation language DML. The DML used by a program running under TPS is, with a few exceptions, the same as for a program running in a different environment, such as timesharing or batch.

It is, in fact, possible to access the same SIBAS data base from TPS, timesharing and batch programs at the same time. All SIBAS calls will go to a common SIBAS interface under the control of TPS. The timesharing or batch user will be unaware of TPS control over the data base unless a TPS restart should happen to change its contents.

1.2.2.6 Checkpoint and Restart

An on-line transaction processing system should have adequate facilities for protection of the data base. If a system failure occurs in a *batch* system, a backup copy of the data base can be mounted and the whole job run once more without too much inconvenience or waste of time. If a failure occurs in an *online* transaction system, reentering and reprocessing the transactions may be very inconvenient, time-consuming or even impossible.

Facilities should be available to assure the integrity of the data base, to minimise the amount of data lost and to restart the system automatically at a well-defined point.

TPS makes use of the extensive checkpoint/restart facilities of SIBAS. These are mainly transparent to both the user and the application program, TPS itself controlling them. *Synchronised* checkpoints of the whole system are taken automatically according to the load on the system. In addition, the application program can take individual *transaction* checkpoints.

If a system failure occurs, the latest transaction checkpoint can be used to restore the system to its status at or near the point of failure (recovery). If the data base is incorrect at this point, synchronised checkpoints can be used to restore the system to a previous state (rollback). (See Figure 1.7) In both cases, those transactions which were active can be restarted automatically at the correct point.

How often checkpoints are taken and what types of backup/restart facilities are used are system parameters controlled by the user. He must weigh the advantages of assuring the protection of data in the data base against the overhead needed to accomplish this.

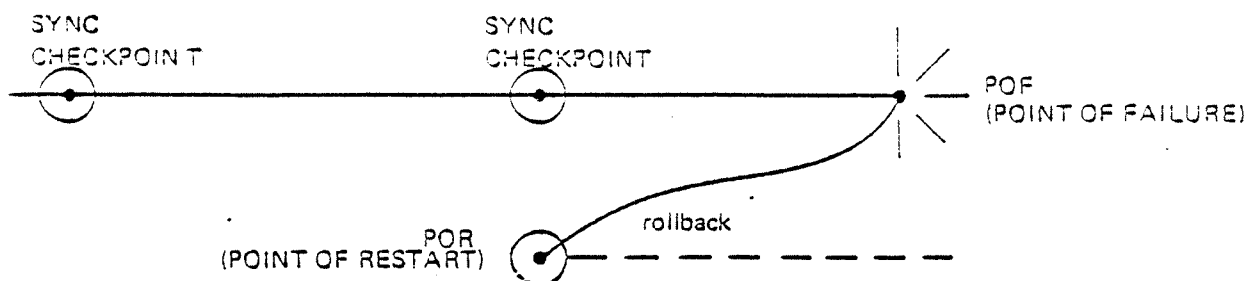


Figure 1.7A: Rollback without Recovery

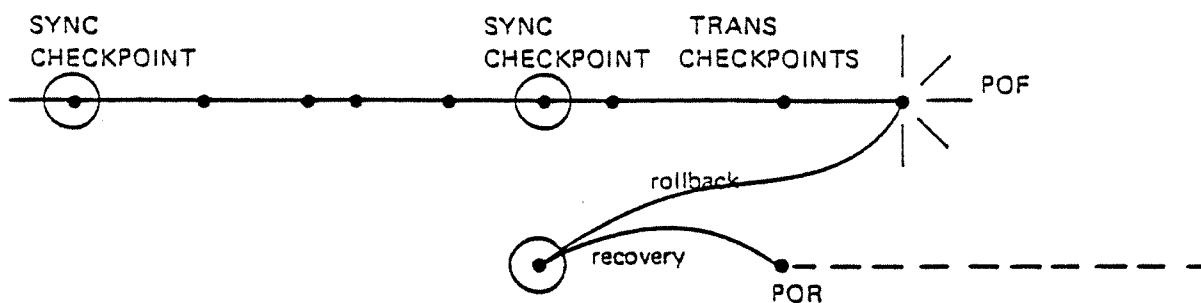


Figure 1.7B: Rollback with Recovery

1.2.3 **Operator Communication**

A special terminal, the operator terminal, is used for starting, stopping and controlling the TPS system. A set of commands is available for interaction with the whole system (system directives) and for interaction with individual modules.

The system directive commands consist of the basic command functions used in connection with system start, stop, pause, checkpoint and rollback/recovery. These commands serve vital functions in connection with normal processing and in case of hardware or software malfunctioning.

Other commands are available for such tasks as starting and stopping individual TPS modules, starting and aborting transactions, changing system parameters, and broadcasting messages.

System messages, both error and informative, will be written on the operator terminal. Application programs may also send messages to this terminal. In a multi-CPU system, those CPUs that do not have an operator terminal will have a log-writer terminal for special error messages.

1.2.4 **Message Routing and Queuing**

Communication between the individual TPS modules is done by messages using a buffer pool and queuing system controlled by the main dispatcher (MD). If the modules are spread across more than one ND100-CPU, there is one MD for each ND100-CPU and they will send messages to the correct CPUs. TPS may thus make use of multiprocessing facilities in a single system.

Every TPS module has a queue for messages to that module. When a message is sent, it goes first to the main dispatcher. MD will put the message in the queue for that module and it will then start the module. The module will read its queue and process the message there. When it is done, it will usually read the queue again in case any messages have arrived in the meantime.

Routines are also available for putting messages that have been received into a waiting queue if they are not to be processed immediately. They can be read from the waiting queue later and processed.

Application programs do not have to concern themselves with these queues. They are controlled by the application program's TPT.

1.3 CONTROLLING TRANSACTIONS

There are two main ways of setting up a transaction, depending on whether the connection between the terminal and TPS is permanent or only lasts as long as a single transaction.

1.3.1 Type 1: Permanent Terminal Transactions

Permanently connected terminals are the simplest to handle and usually give faster response time, because the overhead in setting up the connection between the terminal and TPS is avoided. In addition, TPS provides some special applications (SIGNON and SELECT) designed mainly for permanently connected terminals.

SIGNON helps the terminal user to sign on to the system by writing a picture on the terminal asking for the user's name and password. A typical SIGNON picture is shown in figure 1.8. When the user has written his name and password, SIGNON will check them, and if they are accepted, control will be given to SELECT.

ND TPS ON LINE AT 15.45 ON MARCH 1, 1982

TTTTTTTT	pppppppp	SSSSSSSS
TTTTTTTT	pppppppp	SSSSSSSS
TT	pp pp	SS
TT	pppppppp	SSSSSSSS
TT	pppppppp	SSSSSSSS
TT	pp	SS
TT	pp	SS
TT	pp	SSSSSSSS
TT	pp	SSSSSSSS

PLEASE ENTER YOUR NAME:

PASSWORD:

Figure 1.8: A SIGNON Picture

SELECT will help the user to start the transaction program (application) he wants by writing a menu picture on the terminal, as shown in figure 1.9A. The user just has to select the item he wants and enter its number. The application will then be started.

It is also possible to have sub-menus, i.e. one menu choice will give a new menu, as shown in figure 1.9B. It is possible to have as many sub-levels of menus as desired. The user can also go from a sub-menu back to the master-menu, back to SIGNON, or even into SINTRAN as a timesharing user.

ND TPS	MASTER MENU
1	ACCOUNTING
2	PAYROLL
3	INVOICE
4	INVENTORY
5	TEXT PROCESSING
6	STOP
ENTRY CHOICE:	

Figure 1.9A: A Master Menu

ND TPS	ACCOUNTING
1	BOOKKEEPING
2	ACCOUNTS RECEIVABLE
3	GENERAL LEDGER
4	REGISTER UPDATE
5	REPORTS
6	MASTER MENU
7	STOP
ENTRY CHOICE:	

Figure 1.9B: A Sub-Menu

However, sooner or later a user application will be started. The user will then carry out his transaction, probably involving interaction with both terminal and data base.

When he is done, control will be given to another special application, SIGNOFF. The main task of this application is to find out what type of transaction it is, in this case a "never-ending" transaction, and to terminate it accordingly, in this case by giving control to SIGNON.

The terminal is now ready to accept a new user and start a new transaction. If it is not necessary to go through the SIGNON procedure again, SIGNOFF could have given control directly to SELECT.

In both cases, the terminal will not be released from TPS, i.e. the Transaction Processing Task (TPT) in TPS controlling this terminal will not release the terminal. There is a permanent connection between the terminal and the TPT. There is a never-ending succession of applications running on this terminal (SIGNON, SELECT, user application, SIGNOFF, SIGNON, SELECT, user application, etc.).

This is illustrated in figure 1.10, Type 1.

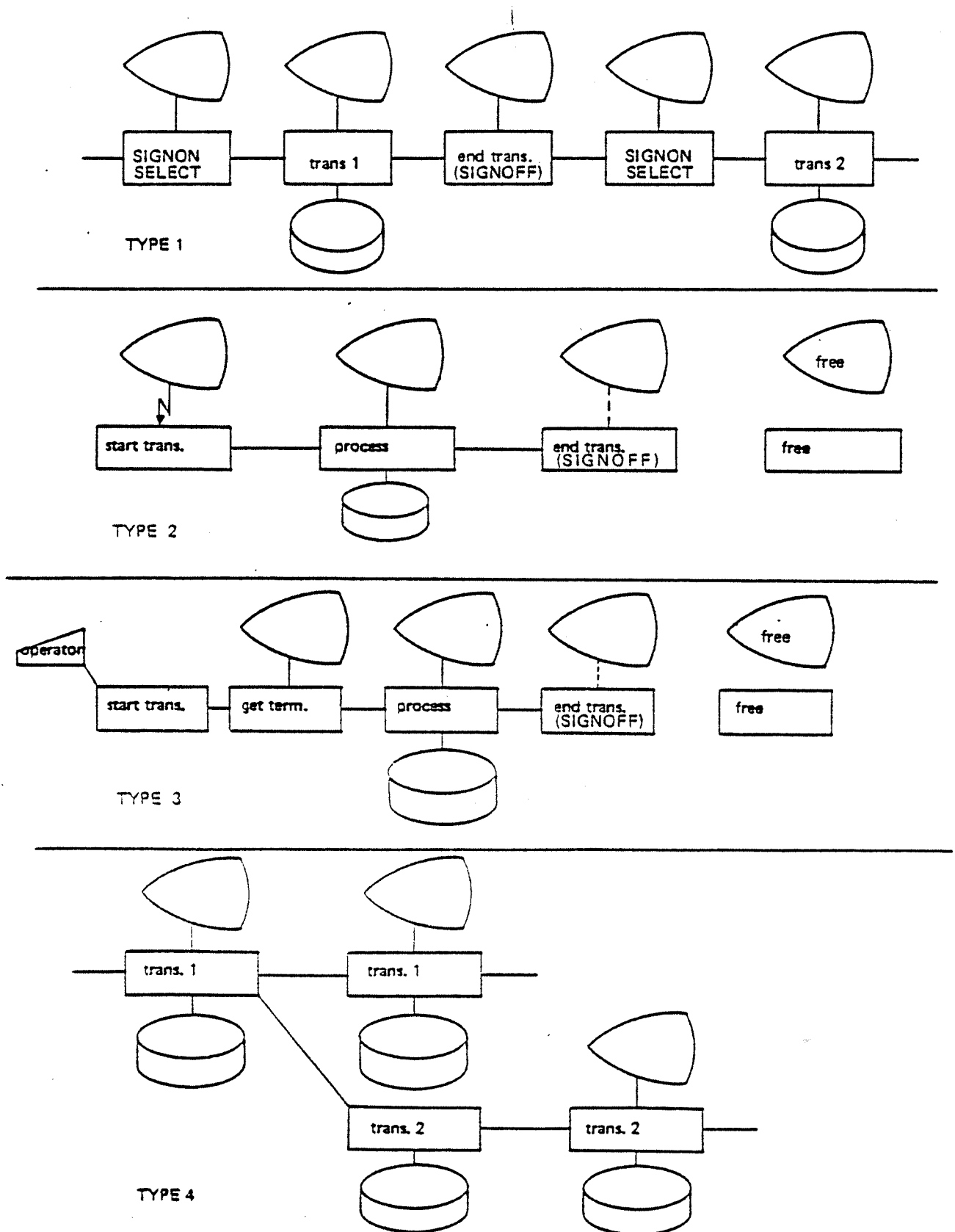


Figure 1.10: Types of TPS Transactions

1.3.2 **Type 2: Short Terminal Transactions**

The other main way of setting up a transaction is by connecting a terminal to TPS whenever a transaction is to be carried out on the terminal and releasing the terminal when the transaction is done. This frees the terminal for other use and, possibly even more importantly, also frees the Transaction Processing Task (TPT) in TPS for use with another terminal.

This method should be used if there are many terminals that are not in constant use. They may then share a limited number of TPTs among themselves. The disadvantage is that there is more overhead in starting up the transaction.

The procedure would be as follows:

A terminal connected to a network is inactive. The user presses a special function key that causes a message to be sent over the network to TPS. TPS allocates a TPT to the terminal and the TPT starts an application program that can converse with the terminal. A dialogue follows between the user and the application program, and the data base is read and updated. When the user indicates to the application program that he is done, the application terminates and both the terminal and the TPT are freed.

1.3.3 **Type 3: Short Local Terminal Transactions**

A terminal is connected locally to the computer and can be used for both TPS processing and other processing. The terminal can be brought into the TPS system by issuing a command at the TPS *operator console*, a special terminal devoted to operational control of TPS. A TPT will be allocated and the application program indicated in the command started. When the application program terminates, both the TPT and the terminal will be freed.

1.3.4 **Type 4: Concurrent Transactions**

A transaction that is active can start another transaction to run *concurrently*. A new TPT will be allocated and the new application started. Data can be sent from the mother task to the daughter task. The daughter task will not have a terminal — if it needs one, it must set up the connection itself.

1.3.5 **Type 5: Future and Periodic Transactions**

A transaction that is active can start another transaction at some time in the future, either at a given absolute time or periodically. When the time comes, the transaction will be started as for Type 4. The TPS operator can also use commands to start a single or periodic application at a specified time. TPS need only be informed once of a periodic application. It will then be started periodically at the correct times.

1.4 PROCESSING A TRANSACTION

This section describes in some detail the steps involved in processing a typical transaction using the standard version of TPS. It follows the transaction sequentially through the TPS system from its initiation by the user until it is terminated.

The transaction described is a transaction of Type 1, a permanently-connected terminal transaction. A connection is established between a terminal and a TPT when SIGNON is first started by TPOPEN (*See Figure 1.11*). SIGNON checks the terminal operator's status and SELECT calls the correct user application. A conversation is carried on between the user and the application program and the data base is read and updated. When the user has no more input and has received all output, he indicates that the transaction is done. The application program will terminate and the SIGNOFF application will give control back to SIGNON to wait for the next transaction.

1.4.1 Starting the Transaction

The TPS system has been started initially and is in normal running state.

TPOPEN has started a number of transactions using the activate-task TSR. The first application to be given control is the SIGNON special application.

The terminal in our example is a standard terminal controlled by the NSHS or FOCUS screen handling system. The first time SIGNON is started, it will reserve the terminal. After that it will call NSHS or FOCUS routines to display a picture asking the user to enter his name, and then wait for an answer in the NSHS or FOCUS input routine.

To start the transaction, the user will enter his name on the terminal and press the return key. SIGNON will be started up with the reply in the input data area. It will check the user's identification and perhaps whether that user is allowed to use that particular terminal. SIGNON may then ask for a password, again using NSHS or FOCUS routines. When this has been checked, SIGNON will switch to SELECT, using the switch-application TSR. SELECT will display the user's master menu and wait for the user to enter the number of the entry he wants. This may result in displaying a sub-menu, going back to SIGNON or switching to a user application. If it is the latter, the transaction has been started.

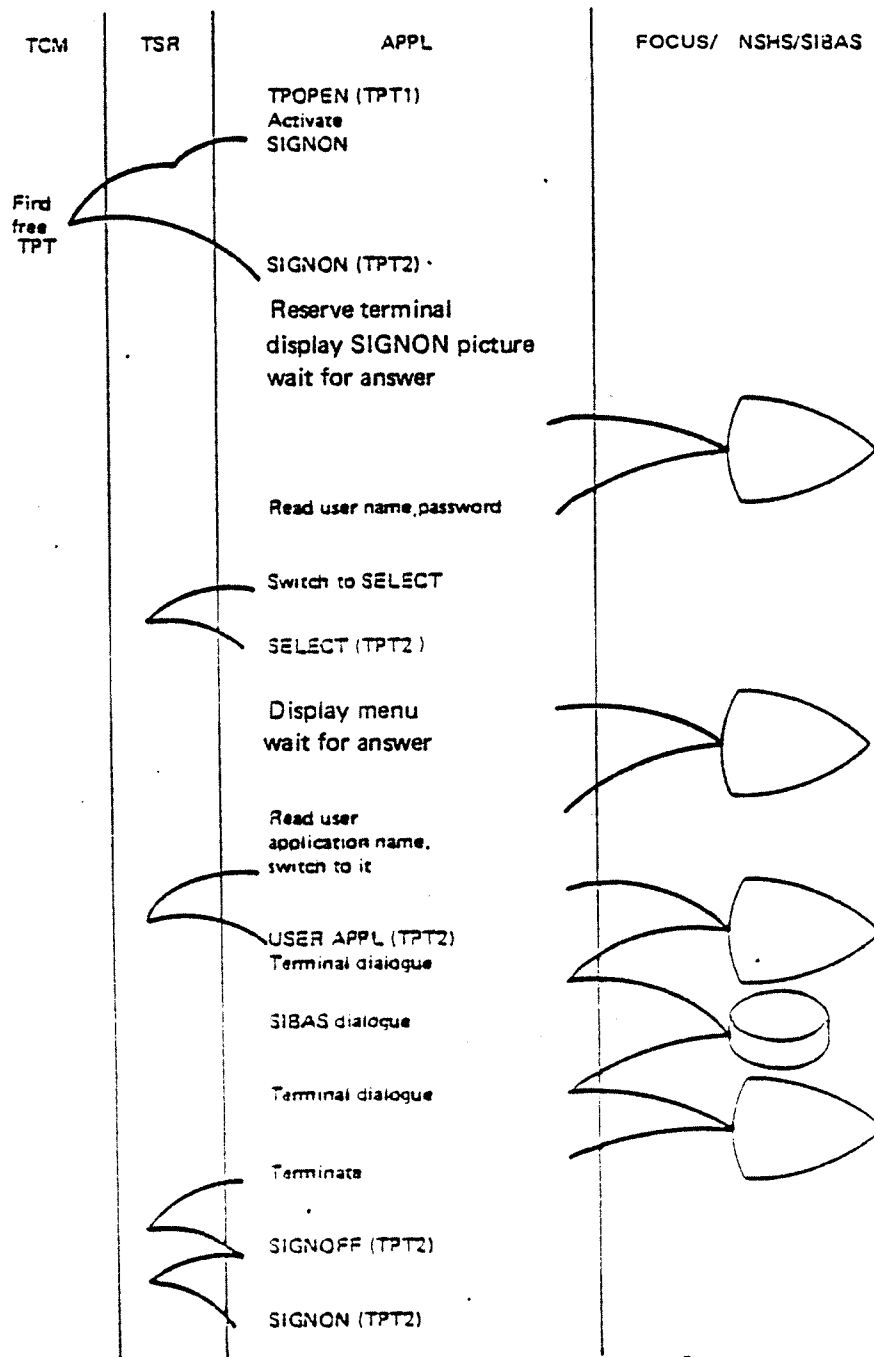


Figure 1.11: A Typical Transaction

1.4.2 Processing the Transaction

Normally, one may envisage the transaction processing in one or more sequences each consisting of a dialogue between the application program and the user, with some activity on the data base as the result of most dialogues.

The transaction may start by asking for some information, for example a transaction type. When the user has answered (register a new customer), a dialogue can follow, prompting the user to enter the details of the transaction (name, address, telephone number, account number, etc.), the data base will be updated and the user notified (the customer has been registered). The transaction may then start a new dialogue.

The data base is accessed through normal SIBAS calls in the application program. All of the common SIBAS data manipulation calls are available to TPS programs, such as OPEN DATA BASE, CLOSE DATA BASE, FIND, GET, MODIFY, STORE, ERASE, REMEMBER, FORGET, etc. It is possible to access records from outside the data base (out-of-the-blue access) in several ways, to conduct searches and to access records via their relationships to other records (relative accesses).

The transaction may be an inquiry transaction, an update transaction, data entry, report generation or any combination of these. It may use TSRs to create a session with an I/O device or application program. It may access local devices by reserving them and communicating directly with them. It may start concurrent or future tasks or switch to another application, again using TSR routines.

1.4.3 Terminating the Transaction

The transaction will be terminated when the logical end of the program is reached, when the user indicates that there is no more processing to be done, etc. When the transaction terminates, the SIGNOFF application will be activated. SIGNOFF may gather some statistics or do other processing and will end by switching to the SIGNON application.

2

ADMINISTRATING TASKS

2.1

TASKS, TRANSACTIONS AND APPLICATIONS

A *task* in TPS can be defined as the processing done by a Transaction Processing Task (TPT) from the time it is allocated by the Transaction Control Module (TCM) until it is freed again. The number of concurrent tasks at any time is thus the same as the number of allocated TPTs.

A task may be either a *short* task that only lasts for one transaction or it may be a *long* task that handles many transactions in a row (but only one at a time). (See Figure 2.1).

A long task will return to the SIGNON or SELECT application between transactions, instead of completely terminating by releasing the terminal and the TPT. This saves the overhead of allocating a TPT every time a new transaction is started and assures that a TPT is available for that terminal. This method should be used mainly for terminals that are in more or less constant use, since the terminal will be permanently connected to that one TPT as long as the task lasts.

A *transaction* can then be defined as the processing done either from the time a TPT is allocated until it is freed for a short task, or from the time control is given to the user application until return to SIGNON or SELECT for a long task.

An *application* program is a user written program linked to and started by the TPT. The application program will run under the control of the TPT and do the actual transaction processing. When it is done processing, it can either switch to a new application or terminate.

A task may thus consist of the sequential processing of one or many transactions and a transaction may consist of one or more application programs.

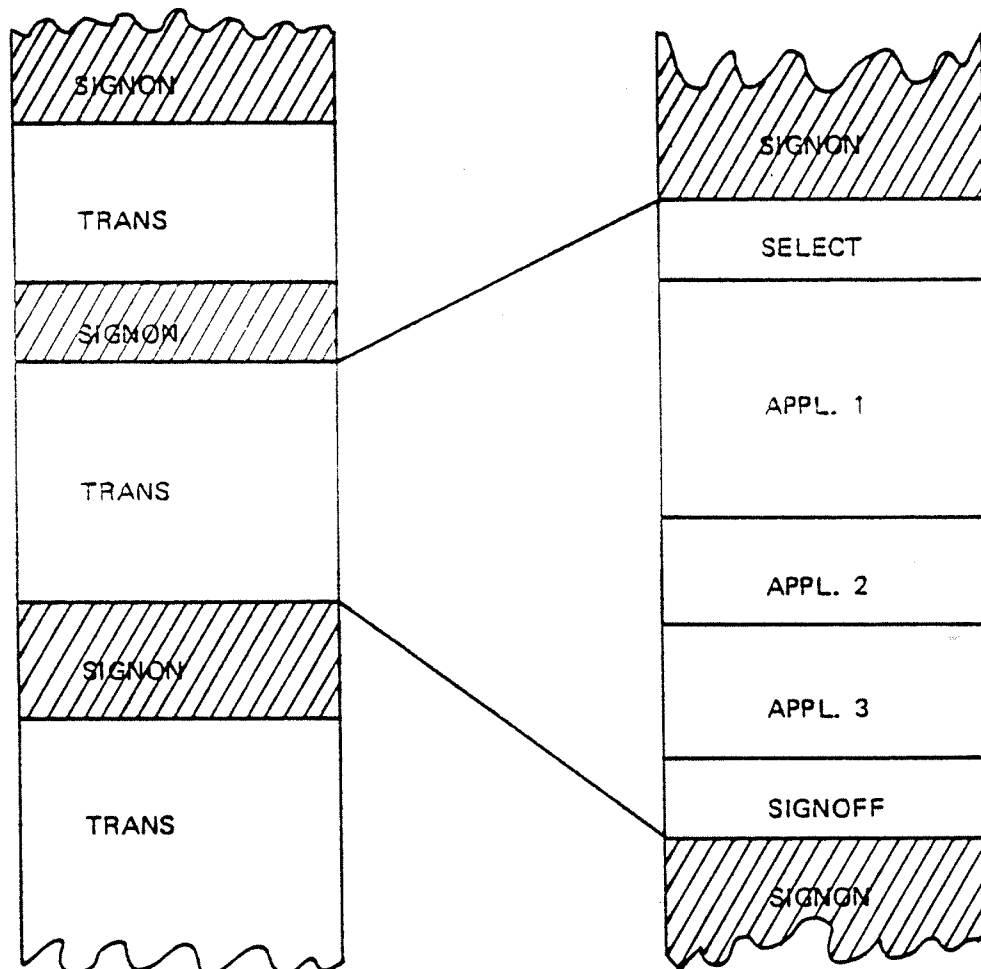
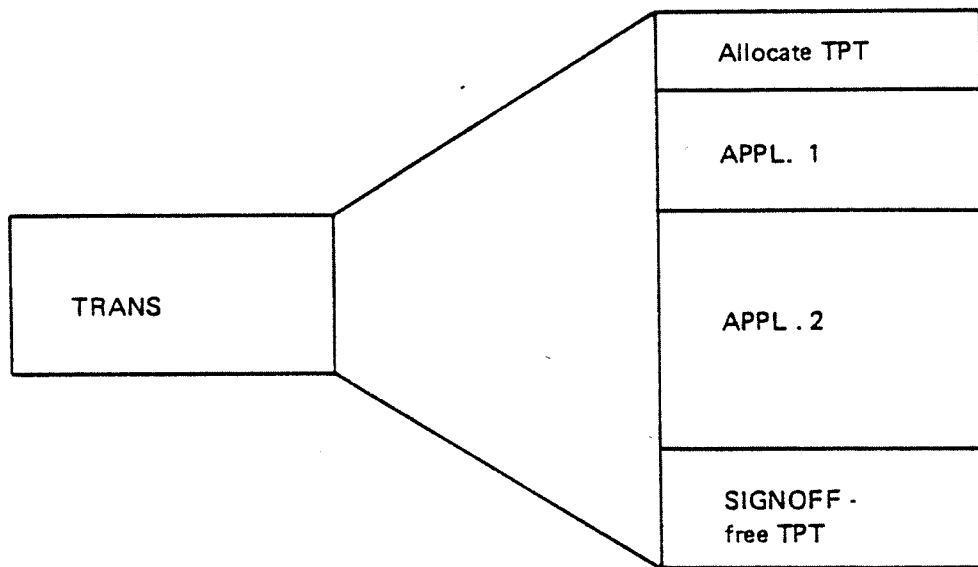


Figure 2.1 Short and Long Tasks

Task administration includes starting tasks, terminating tasks and transactions, and switching application programs. (See Figure 2.2.) Task administration can be done by application programs and by other TPS components, such as input/output modules and the system operator. This chapter will only discuss the task administration that can be done by application programs.

<i>TACTV</i>	ACTIVATE A CONCURRENT TASK
<i>TASET</i>	SET THE EXECUTION TIME FOR A FUTURE TASK
<i>TINTV</i>	SET THE EXECUTION INTERVAL FOR A PERIODIC TASK
<i>TDCNT</i>	DISCONNECT THE EXECUTION TIME/INTERVAL
<i>TSWAP</i>	SWITCH TO ANOTHER APPLICATION PROGRAM
<i>TSTOP</i>	TERMINATE THE TRANSACTION
<i>TTERM</i>	TERMINATE THE TASK COMPLETELY
<i>TSTST</i>	SET THE TERMINATION STRATEGY
<i>TSCST</i>	SET THE CLOSE STRATEGY
<i>TSAST</i>	SET THE ABEND STRATEGY

Figure 2.2: Task Administration TSRs

2.2 STARTING TASKS AND SWITCHING APPLICATIONS

Tasks may be started in several ways:

- a session request from an IOM to a TCM will start a new task (*See Section 3.3.1*)
- a session request from an application program to a TCM, requesting a session with another application program, will start a new task. This is done with the TSOPN TSR and is described in section 3.3.3.
- the operator can start a task using the activate application command.
- the special application program TPOPEN can start tasks when the TPS system is initially started (*See Section 6.3*)
- an application program can start both concurrent, future and periodic tasks

Thus an application program is allowed to start both concurrent and future tasks and to set the execution interval for periodic tasks. These functions are carried out through TSRs.

2.2.1 Immediate Task Activation

2.2.1.1 TACTV - The Activate Concurrent Task TSR

An application program may activate a new task on the same TCM to run concurrently with itself. A new TPT will be allocated if one is available and the given application program started. Up to 2000 bytes of data can be transferred to the activated task. The activated task will receive the data in the beginning of the task common data area(See Section 7.1.2).

```
CALL TACTV ( <application number> , <record> , <size> , <status> )
CALL 'TACTV' USING <application number> <record> <size> <status> .
```

If no TPT is available, an error code is returned in the status parameter. The availability of a TPT is determined by the number of free TPTs and the priority of the new application.

The old task and the new one will run independently and have no common data area. If they want to communicate, one way is to use the TSOPN TSR instead of the TACTV TSR, since a session will then be established between them. Another way of communicating is through internal devices (See Section 4.4).

Example - FORTRAN

PROGRAM 1

```
COMMON/Private/ITERM(128),IPRIV(2000)
DIMENSION IREC(20)           data record to be sent to
                              program 2
.
.
CALL TACTV(52,IREC,40,ISTAT)  activate program 2 (appl 52),
                              send 40 bytes of data to it
IF (ISTAT.LT.0) GO TO error routine check return status
.
.
```

PROGRAM 2 (APPLICATION 52)

```
COMMON/Private/ITERM(128),IPRIV(5)
DIMENSION IDATA(20)          define data area
.
.
DO 10 I=1,20                  move data from beginning of
10 IDATA(I)=ITERM(I)          common area to right area
.
.
```


Example - COBOL

PROGRAM 1

WORKING-STORAGE SECTION.

```

01  NSHS-AREA.
    02  ITERM COMP OCCURS 128.
    02  IPRIV COMP OCCURS 2000.
01  DATA-REC COMP OCCURS 20.    data record to be sent to
    .                               program 2
    .
MOVE 52 TO APPL-NR.              activate program 2 (appl 52),
                                send 40 bytes of data to it
CALL 'TACTV' USING APPL-NR DATA-REC C40 STATUS-CODE.
IF STATUS-CODE < 0 GO TO ERROR-ROUTINE.  check return status
    .

```

PROGRAM 2 (APPLICATION 52)

WORKING-STORAGE SECTION.

```

01  NSHS-AREA.
    02  ITERM COMP OCCURS 128.
    02  FILLER REDEFINES ITERM.  data from program 1 put at
        03  PROG1-DATA COMP OCCURS 20.  beginning of common area
        03  FILLER PIC X(216).
01  DATA-REC COMP OCCURS 20.    define data area
    .
    .
MOVE PROG1-DATA TO DATA-REC.    move data to right area
    .

```

2.2.2 Future and Periodic Task Activation

The following timing functions are available to application programs

- set the absolute execution time for starting an application
- set the interval for a periodic application
- remove (disconnect) both absolute and periodic timing for an application

Up to 16 applications can be in the time queue and up to 16 in the interval list. Only 1 absolute start time can be given for an application.

Timing information is stored on the disk when synchronized checkpoints are taken and is therefore restored at rollback.

2.2.2.1 TASET — The Set Execution Time TSR

An application program may activate a task, on the same TCM or a different one, to be started at a specified absolute time. Two parameter values may also be specified.

```
CALL TASET (<module>, <application number>, <parameter 1>,
            <parameter 2>, <time>, <status>)
```

```
CALL 'TASET' USING <module>, <application number>, <parameter 1>,
                  <parameter 2>, <time>, <status>.
```

When the specified time (second, minute, hour, day, month, year) has been reached, a new TPT will be allocated on the specified TCM and the application program will be started. The two parameter values (for example a terminal number and type) will be placed in the beginning of the task common data area.

If the specified time has already been reached when the TSR is issued, the new task will be started immediately.

If no TPT is available when the task is to be started, an error message will be written on the TPS operator console.

The time resolution is 5 seconds.

Example - FORTRAN

```

      DIMENSION ITIME(6)
      .
      .
      ITIME(1)=0           start appl 8 on TCM0
      ITIME(2)=0           at absolute time 10am
      ITIME(3)=10          on December 31, 1980
      ITIME(4)=31          with parameters 46 and 4
      ITIME(5)=12
      ITIME(6)=1980
      CALL TASET(32,8,46,4,ITIME,ISTAT)
      IF (ISTAT.NE.0) GO TO error routine

```

Example - COBOL

```

      MOVE '0' TO ABS-TIME(1) ABS-TIME(2).  start appl 8 on TCM0
      MOVE '10' TO ABS-TIME(3).             at absolute time 10am on
      MOVE '31' TO ABS-TIME(4).             December 31, 1980
      MOVE '12' TO ABS-TIME(5).
      MOVE '1980' TO ABS-TIME(6).
      CALL 'TASET' USING TCM-0 APPL-8 TERM-NR TERM-TYPE
                          ABS-TIME STATUS-CODE.
      IF STATUS-CODE NOT = 0 GO TO ERR-ROUTINE.

```

2.2.2.2 TINTV — The Set Interval TSR

An application program can set the execution interval for an application. The next time the application is activated, it will become periodic.

```
CALL TINTV (<module>, <application number>, <parameter 1>, <parameter
2>, <interval>, <status>)
```

```
CALL 'TINTV' USING <module>, <application number>, <parameter 1>,
<parameter 2>, <interval>, <status>.
```

The TINTV TSR will not itself start periodic execution of the specified application program. This must be done by some other means (see section 2.2). Once it has been started, however, it will continue periodically at the specified time intervals. The next interval will start at each time of activation.

A periodic application can have only one execution interval. If it already has an interval, the new interval will replace the old one.

An application can set its own interval, but if it is not already periodic, the next execution must be started by some other means.

The execution interval is specified as seconds, minutes, hours and days. Since months and years are not well-defined time spans, long intervals must be specified in days.

The time resolution is 5 seconds.

Example - FORTRAN

```
DIMENSION INTVL(4)

INTVL(1)=0           set execution interval to
INTVL(2)=30          1 hour and 30 minutes for
INTVL(3)=1           appl 5 on TCM0 (no
INTVL(4)=0           parameters)
CALL TINTV(32,5,0,0,INTVL,ISTAT)
IF (ISTAT.NE.0) GO TO error routine 1
CALL TACTV(5,0,0,ISTAT)      start it (no data)
IF (ISTAT.NE.0) GO TO error routine 2
```

Example - COBOL

```
MOVE '0' TO INTERVAL(1) INTERVAL(4).
MOVE '30' TO INTERVAL(2).      set execution interval to
MOVE '1' TO INTERVAL(3).      1 hour and 30 minutes
CALL 'TINTV' USING TCM0 APPL-5 ZERO ZERO
                              INTERVAL STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO ERR-ROUT-1.
CALL 'TACTV' USING APPL-5 ZERO ZERO STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO ERR-ROUT-2.
```

2.2.2.3 TDCNT — The Disconnect Application TSR

An application can be removed (disconnected) from the time queue and the interval table.

CALL TDCNT (<module>, <application number>, <status>)

CALL 'TDCNT' USING <module>, <application number>, <status>.

The application will be immediately removed from both time queue and interval table with this TSR.

Examples

CALL TDCNT(32,5,ISTAT) disconnect appl 5 on TCM0

CALL 'TDCNT' USING TCM0 APPL-5 STATUS-CODE.

2.2.3 Switching to Another Application

2.2.3.1 TSWAP - The Switch Application Program TSR

The processing of a transaction may involve the activation of several application programs, one at a time. When one program is done, it may switch to another program instead of terminating.

The TSWAP TSR is used to switch to another application program.

Examples

```
CALL TSWAP(25, ISTAT)           switch to appl 25
Error routine (will not return here if OK)
```

```
MOVE 25 TO NEXT-APPL.
CALL 'TSWAP' USING NEXT-APPL STATUS-CODE.
Error routine (will not return here if OK)
```

If the old application had a terminal or a session, the new application will have the same terminal or session partner and may continue to exchange messages with this partner.

The new program will have access to the data area of the old program if it is defined as belonging to the task common data area. In a FORTRAN program, this will be the COMMON/Private area. COBOL programs must contain a section of working storage which is identical in all applications which may be executed within one task.

The new program can be written in the same language as the previous one or in any other of the available languages.

When both COBOL and FORTRAN programs are to be executed in the same task, the data areas should be arranged as in Figure 2.3. It is the programmer's responsibility to make the two maps identical and avoid destruction of common data at run time. For a more detailed description of common areas see Chapter 7.

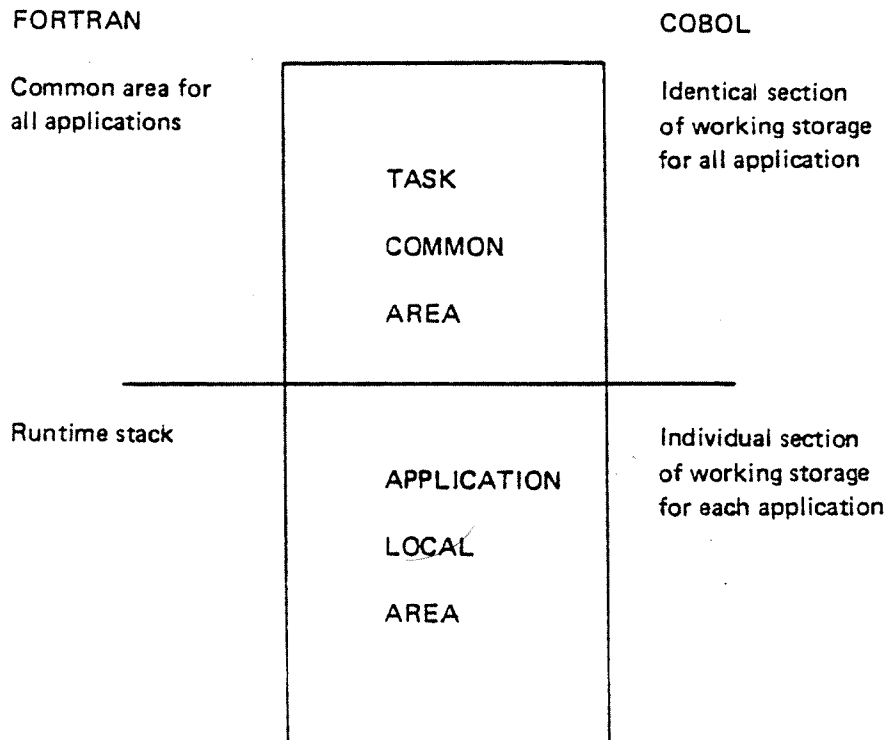


Figure 2.3: Application Data Area

2.2.4 The SIGNON and SELECT Special Applications

SIGNON and SELECT are special applications supplied with TPS. The standard versions of the special applications are discussed in chapter 6. They are mentioned here because they may play an important role in task administration.

The main function of SIGNON is to check the identity of the user and perhaps ask for a password. The standard version also reserves the terminal and initiates NSHS or FOCUS if necessary. If the user has been accepted, SIGNON ends by switching to SELECT.

The function of SELECT is to ask the user to select the application to be run. When the user has answered, the SELECT application will switch to the desired user application program.

2.3 TERMINATING TRANSACTIONS

When an application program is done, it can either switch to a new application program using the TSWAP TSR (*See Section 2.2.2*) or it can terminate the transaction. If the transaction is terminated, this can be done normally or abnormally.

2.3.1 Normal Termination

Normal termination of a transaction can be caused by:

- reaching the logical end of the program (the END or STOP RUN statement)
- using the TSTOP TSR with a stop code of 0 or using the TTERM TSR
- the LEAVE monitor call (CALL LEAVE)

2.3.1.1 TSTOP - The Stop Transaction TSR

The TSTOP TSR may be used to terminate a transaction either normally with a stop code of 0 or abnormally with any other stop code. A negative stop code can be used to give a formatted error message from NSHS (-1) or SIBAS (-2).

Normal termination will activate SIGNOFF, while abnormal termination will activate ABEND.

The application should have performed appropriate housekeeping on the data base, session partner, devices, etc.

```
CALL TSTOP ( <stop code> )
CALL 'TSTOP' USING <stop code>.
```

Examples

```
SCODE=3
CALL TSTOP(SCODE)                stop code = 3
```

```
CALL 'TSTOP' USING ZERO.         stop code = 0
```


2.3.1.2 TTERM — The Terminate Task TSR

It is also possible to terminate a task directly instead of by going to SIGNOFF. The termination will be a complete termination and the TPT will be freed.

```
CALL TTERM (<checkpoint>)  
CALL 'TTERM' USING <checkpoint>.
```

A transaction checkpoint is normally taken when a task is completely terminated (see TTRAN). However, this can be prevented by using TTERM with the checkpoint parameter set to 1. This saves the overhead of taking a transaction checkpoint, but it could create a problem if recovery is done on the TPS system.

The application program should close the data base, close a session, release resources, etc., before using the TTERM TSR.

Examples

```
CALL TTERM(0)  
  
CALL 'TTERM' USING ZERO.
```

2.3.1.3 The SIGNOFF Special Application

When a transaction terminates normally, the SIGNOFF special application is given control to carry out the actual termination. There are several ways of terminating a transaction, the main choice being between complete task termination (freeing the terminal and the TPT), and continuing the task with a new application, usually SIGNON or SELECT.

2.3.1.4 TSTST - The Set Termination Strategy TSR

SIGNOFF uses the *termination strategy* for the task to determine which course to follow. When the task is originally started, the termination strategy is set to 1, which usually indicates complete termination. The TSTST TSR can then be used to change the strategy to any other value, the meaning of each value depending on the way it is interpreted by SIGNOFF.

```
CALL TSTST ( <term. strategy>, <term. appl.> )
CALL 'TSTST' USING <term. strategy>, <term. appl.>.
```

Examples

```
CALL TSTST(1)                                complete termination, TPT
                                              released

MOVE 20 TO TERM-APPL.                        user termination application
CALL 'TSTST' USING FOUR TERM-APPL.
```

In addition to setting the termination strategy, the TSTST TSR can be used to indicate a user-written termination application. The standard termination strategies are described under the SIGNOFF application in chapter 6.

2.3.1.5 TSCST — The Set Close Strategy TSR

When the CLOSE-TPS operator command is given, a controlled stop sequence will be initiated. Normally, active tasks (TPTs) will continue until they are terminated, but no new TPTs may be allocated. When all TPTs have been freed, TPS will be stopped.

However, since long tasks do not free their TPTs between transactions, a close sequence will never be completed in systems with this type of task. To avoid this, it is possible to indicate that a task is to be terminated immediately and completely if a close command is given. This is done by setting the close strategy to immediate termination with the TSCST TSR.

On the other hand, to prevent a transaction from being terminated in the middle of processing, the close strategy can be set to normal termination while the actual processing is being done. Then, when that transaction is complete, the close strategy may be set back to immediate termination. This will also cause the task to be terminated if a close command has already been given.

The close strategy can be set in both user applications and special applications. For example, SIGNOFF may set it to immediate termination, and the last thing SELECT may do before starting a user application is to set it to normal termination. But it may also be left to the application programmer to control this.

```
CALL TSCST (<close strategy>)
CALL 'TSCST' USING <close strategy>.
```

The value of <close strategy> is either 0 (normal termination) or 1 (immediate termination). If it is 1, TSCST will also check if a close command has already been given and terminate the task if it has.

The close strategy is normally set to 1 in the beginning of SINON and changed to 0 before swapping to the SELECT application.

Examples

```
CALL TSCST(1)

CALL 'TSCST' USING ONE.
```

2.3.2 Abnormal Termination

Abnormal termination of a transaction can be caused by:

- the FORTRAN or COBOL runtime system
- the TPS system
- the TPS operator
- the application program itself by using the TSTOP TSR with a non-zero stop code.

2.3.2.1 The ABEND Special Application

When a transaction is terminated abnormally, the ABEND special application will be called before termination to write an error message, take a dump or do some other special processing. When the ABEND application is done, it will usually switch to SIGNOFF to terminate as for normal termination.

2.3.2.2 TSAST - The Set Abend Strategy TSR

ABEND uses the *abend strategy* for the task to determine what action to take in connection with abnormal termination. The default value is 1 when the task is started. The TSAST TSR can be used to change the strategy to another value, the meaning of each value depending on the way it is interpreted by ABEND.

```
CALL TSAST (<abend strategy>, <abend appl.>)
CALL 'TSAST' USING <abend strategy> <abend appl.>.
```

Examples

```
IABAPP=12                                user abend application
CALL TSAST(4,IABAPP)
```

```
CALL 'TSAST' USING THREE.                dump the data area on the printer
```

In addition to setting the abend strategy, the TSAST TSR can be used to indicate a user-written abend application. The standard abend strategies are described under the ABEND special application in chapter 6.

2.3.2.3 Illegal Monitor Calls

The monitor call routines allowed in TPS are listed in appendix E. If a FORTRAN or COBOL program calls an illegal monitor call routine, the program will be abnormally ended with the error 'illegal use of TSRs'.

If an illegal routine is called from a MAC or NPL program, the routine will be executed and no error message written. However, the programmer is strongly advised not to use routines not on the list, since they may hang the TPS system or cause unpredictable results. In addition, PLANC, MAC and NPL programs should not use the MON instruction directly, but call the corresponding FORTRAN monitor call subroutine.

2.3.2.4 Timeout

When an application is loaded, a maximum time between TSR calls is set. If the application runs longer than the given time, it will be abnormally ended with the error 'timeout'. The timeout depends on the application priority, a low priority giving a long timeout. Setting the maximum time to 0 will allow the application to run for an indefinite time. The application timeout can be turned on and off with the TTONS and TTOFF TSRs.

There is also an operator timeout. This is useful if, for example, the terminal has been turned off or the operator does not answer for some other reason. The operator timeout time can be changed with the TSOPT TSR.

A timeout is also used when restarting the system after rollback or recovery. If communication with the terminal is involved during restart (*See Section 6.2.3*), it is not certain that the terminal operator is still there or that the terminal is still turned on. If no answer is received within a specified time, the application is abnormally ended.

3 INPUT/OUTPUT PROCESSING

This chapter describes how to handle input and output processing under TPS.

The data base is controlled by the SIBAS data base management system through standard SIBAS calls. Using SIBAS in TPS application programs is discussed in section 3.1.

Display terminals will usually be controlled through the NORD Screen Handling System, NSHS or FOCUS. This system can be used for most types of display terminals, connected both locally and through input/output modules. NSHS and FOCUS are discussed in section 3.2.

Other input/output devices are of two types. The first type is special TPS devices controlled by Input/Output Modules (IOMs) and accessed by the application program through special Transaction Service Routines (TSRs). These are described in section 3.3.

Finally, TPS application programs may also use standard devices and files available to all users of the local computer system, also non-TPS users. These devices and files are controlled directly by the application program using standard input/output statements and SINTRAN monitor calls. Standard device handling is discussed in section 3.4.

3.1 SIBAS UNDER TPS

3.1.1 Data Definition and Manipulation

The data base in a TPS system is controlled by the SIBAS data base management system. The data base is defined and created using the SIBAS data definition/redefinition language DRL. This is done independently of TPS, in background (timesharing or batch) mode.

The data base is accessed from application programs using the SIBAS data manipulation language DML. A TPS application program uses the same SIBAS calls to access the data base as a program running in a different environment such as timesharing or batch. These calls are described in detail in the SIBAS User's Manual and a summary of SIBAS DML statements is found in appendix G. This section only discusses special considerations which should be taken when using SIBAS under TPS.

3.1.2 The SIBAS Interface Routine

When an application program calls a SIBAS routine, the call can not go directly to SIBAS, but will go to the SIBAS interface routine (DML simulator). (See Figure 3.1). This routine functions as a communication interface between the user and SIBAS, sending calls from one to the other via internal devices or core common.

This interface routine makes it possible to access the same SIBAS data base from TPS programs, other RT programs and background (timesharing and batch) programs at the same time. The interface is divided into a user side and a SIBAS side, and all SIBAS calls will go from the individual user interfaces to the common SIBAS interface and back again.

The ability to access a TPS data base from background programs can be an aid to program testing. The programmer must however be aware that no checkpoints are taken of background programs and a TPS rollback or recovery operation may cause the data base and the background program to be inconsistent. Normally programs testing in background will have their own test version of the data base.

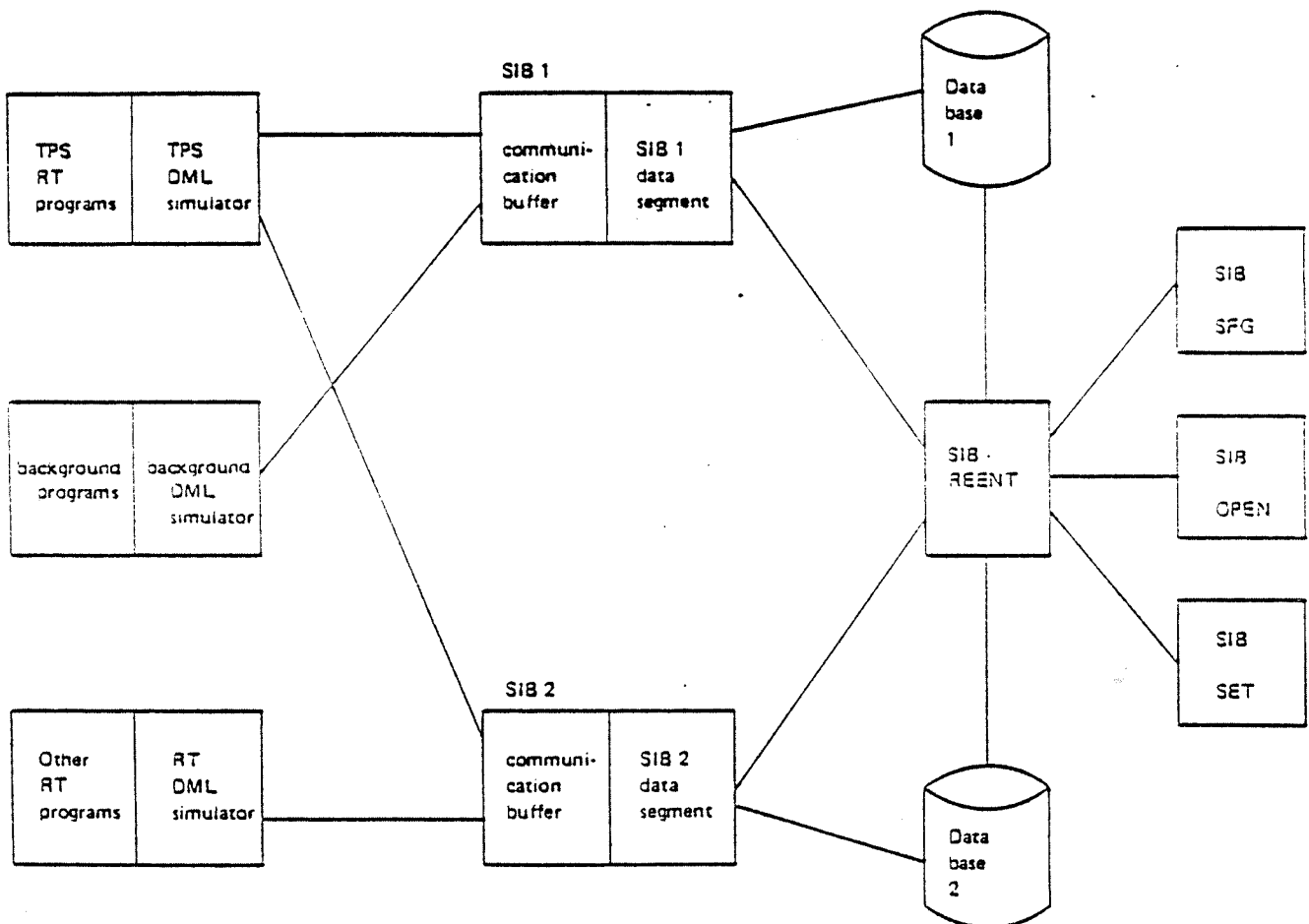


Figure 3.1: SIBAS User Interface

3.1.3 Opening and Closing the Data Base

The data base is opened and closed using the normal SOPDB, SRRLM and SCLDB calls. Opening the data base will cause it to be "physically" opened if no other program has it open at the moment, otherwise opening it will just cause the caller to be registered as a user. Since the former involves more overhead than the latter, it may be most convenient for the TPOPEN special application to open the data base physically and to leave it open until the TPCLOSE application closes it physically. Application programs can then open and close it as needed without causing unnecessary overhead.

Another factor to be considered in opening and closing the data base is that a transaction checkpoint is taken every time the data base is opened or closed. This overhead can also be avoided if the data base is only opened once for each user, for example the first time SIGNON is called. This of course is only possible for long tasks, since the 'user' is the TPT and this must not change. On the other hand, taking a transaction or synchronised checkpoint involves more overhead if the data base is open. If an application program has a long phase with no data base accesses (reading input from the terminal, for example), it may be best to close the data base and open it again afterwards, if transaction checkpoints are taken often (*See Figure 3.2*).

For a detailed discussion of checkpoint and restart see chapter 5.

3.1.4 Using More Than One Data Base

Application programs are allowed to use more than one data base, but if they do and the checkpoint/restart facility is used, an application program must only have one data base *opened* at a time. (*See Figure 3.3*) If this rule is not followed, a restart could cause problems, since checkpoints are only taken of the data base in current use (the one given in SETDV). The other data base may not only be inconsistent, but may not be closed properly at rollback or opened at restart. These problems are avoided if only one data base is open at a time for each application program (different application programs may, however, have different data bases open at the same time).

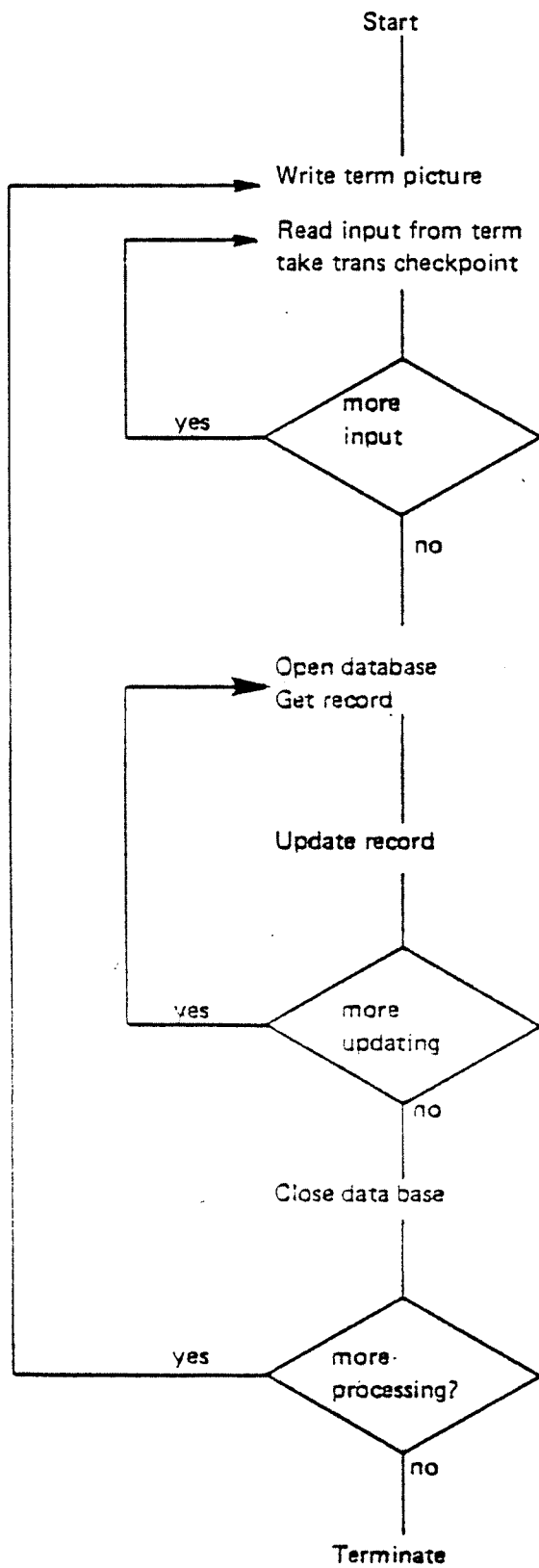


Figure 3.2 Database Open-Close

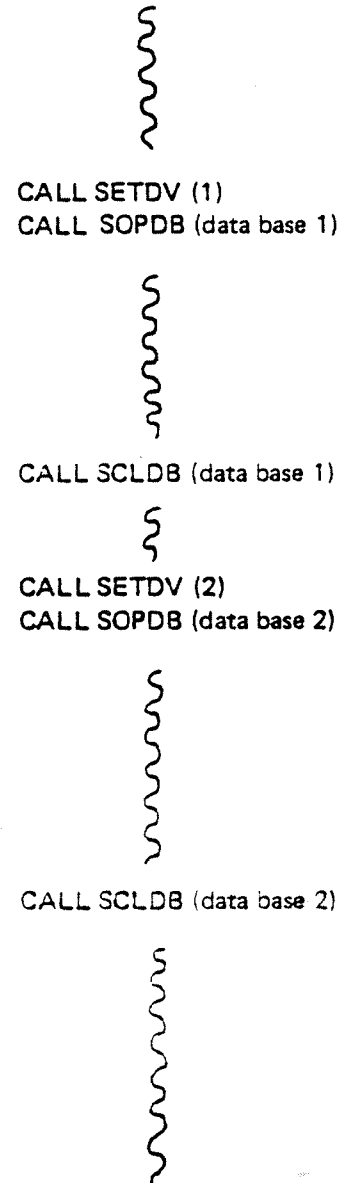


Figure 3.3 Using more than one Database

3.1.5 **SIBAS in ND-500 Multi-CPU TPS**

In ND-500 multi-CPU TPS, SIBAS process(es) may run in both the ND-100 and the ND-500. Applications running in either CPU may call any SIBAS-process, the SIBAS device number identifies the SIBAS-process, e.g. SIBAS-number 0 may run in the ND-100 and SIBAS No. 1 and No. 2 in the ND-500. Apart from the SIBAS device number, there are no other differences in calling SIBAS on another machine from the application's point of view. Note that when calling SIBAS on another machine than where the application is running (E.g. ND-100 → ND-500), you get a substantial increase in system overhead. Consult the SIBAS User's Manual (ND-60.127) for further information on overhead and on how to call SIB-DML from applications running on the ND-500.

3.1.6 Restricted SIBAS Calls

Certain SIBAS functions are not controlled by user application programs, but by special applications called by TPS when these functions are required, (*See Section 6.4*). These functions are synchronised checkpoint, rollback and recovery and they are controlled by the CHECKPOINT, ROLLBACK and RECOVER special applications respectively.

Since checkpoint/rollback/recovery affects the whole TPS system, SIBAS routines for these functions must not be called by user application programs. This holds also for logging routines such as initiating the log files and turning the routine log on and off during recovery.

The special applications also control the state SIBAS is in at any time and routines that change the state must therefore not be called from user application programs.

Other SIBAS routines which should not be called are BSEQU and ESEQU since these are used in a special way by TPS. The THSYN (hold synchronized checkpoint) and TTSYN (allow synchronized checkpoint) TSRs should be used instead to indicate critical sequences.

The reason for this is that when recovery is done, SIBAS will reprocess all calls from the last synchronized checkpoint to the end of the last critical sequence. At the same time, TPS will restore transactions to the last transaction checkpoint. If the transactions are to continue, these two points must be the same. To achieve this, the transaction checkpoint routine makes use of BSEQU and ESEQU. However, if recovery with automatic transaction restart is not used, user application programs may call TBSEQ and TESEQ. (See Appendix H.)

The restricted SIBAS calls are shown in figure 3.4.

Checkpoint/rollback/recovery

**GCHPO
SCHPO****SCROLL
SREPR****SICON**

Logging

INLOG**ONLOG****OFLOG**

Status

**START
STOPS****SRUN
SPAUS
SRECO****SFINI
STREP
SPASS**

Miscellaneous

**RESIB
RELSI
SABOR****CHCOM
SIBIO
STRLG****RBLAN
SBLAN
ZTRB***Figure 3.4: Restricted SIBAS Calls*

3.2 NSHS AND FOCUS UNDER TPS

3.2.1 Handling Display Terminals

Display terminals can be controlled through the NORD Screen Handling System (NSHS) or the FOCUS Screen Handling System. It is of course possible to write to and read from display terminals using the standard I/O facilities (as discussed in section 3.4), but NSHS or FOCUS provide more advanced facilities for screen handling. In the ND500, only FOCUS ought to be used, or you will get a great amount of system overhead.

3.2.2 The NSHS System

NSHS provides facilities for picture definition with leading texts and data fields, various field types, input control, cursor control etc. NSHS can be used for standard terminals and some terminals controlled by I/O modules (*See Figure 3.5*). The NSHS calls are the same for both types. However, each type has its own version of NSHS and the correct version must be used. This is described under loading applications.

3.2.2.1 Defining and Using Pictures

Pictures are defined using the NORD screen definition system. This is a background program which is run as a SINTRAN timesharing program or a batch job, not as a TPS program. Defining pictures is discussed in section 7.7.

After a picture has been defined, it can be used by a TPS application program via calls to the NORD screen library system. These calls are listed in appendix F. For a complete discussion of the screen definition system and the screen library system, see the NORD Screen Handling System Manual.

3.2.2.2 Q^cQ^cQ^c and Restart

Control Q (pressing the control and the Q keys simultaneously) *3 times* in a row has a special function in screen handling. It will clear the screen and write out the latest picture and any input to the latest RFLDS call. If input from one picture is read with several RFLDS calls, input to previous RFLDS calls is not shown, since it has already been sent to the application program.

This Q^cQ^cQ^c function is used normally to restore a picture if, for example, the terminal is turned off by mistake or if the picture disappears because of power failure.

Q^cQ^cQ^c is used by TPS in connection with system restart (*See Section 5.3.2*), in order to restore the screen picture, which may have been lost when the system was down, and to position the cursor correctly. The terminal operator is instructed by the RESTART application to press Q^cQ^cQ^c. When he does so, the latest picture and any input to the latest RFLDS will be restored. Input to previous RFLDS, however, which has already been processed by the application program, will not be restored. This may cause some confusion for the terminal operator as to what has been registered, even though the cursor will be positioned properly. One way to avoid this situation is to read all input from one picture with a single RFLDS call if this can be fitted to the program logic.

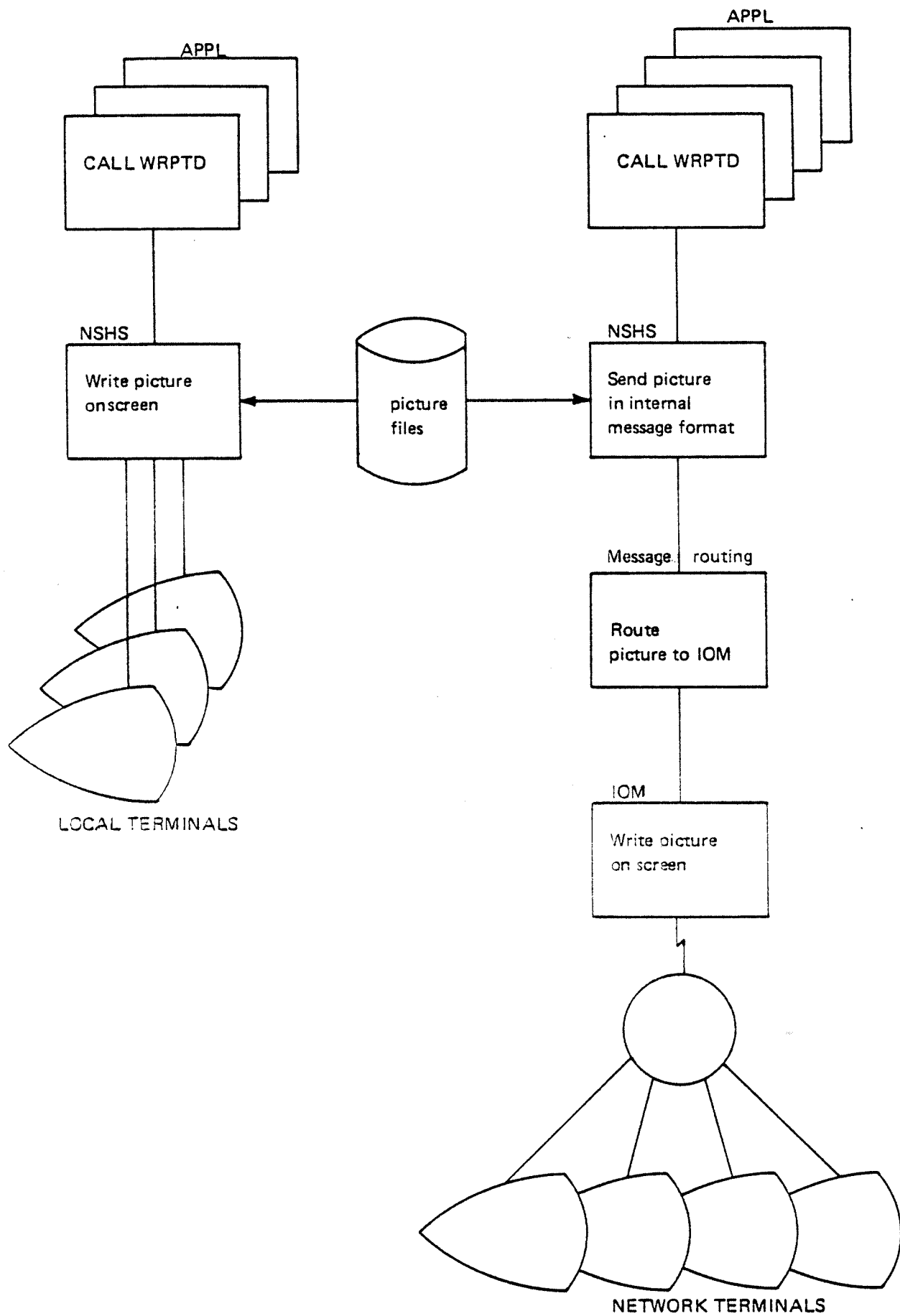


Figure 3.5.A: NSHS for Local and Special Terminals

3.2.3 FOCUS Level 1

FOCUS Level 1 is a high level screen handling system that can be used to control local or remote asynchronous terminals or synchronous (buffered) terminals using ISO 1745 or 3270* as line procedure. The calls to the screen handling system are the same for locally connected terminals and for remote terminals. The communication is carried out by the TPS system which establishes a session between the application's TPT and the remote FOCUS process, and calls "FCINITE" with specific parameters.

* *Not yet implemented.*

3.2.3.1 Defining and Using Forms

The definition of forms (pictures) is done using the FOCUS-DEFINE system. This is a background program which is run as a SINTRAN timesharing program, not as a TPS-program. Defining pictures is discussed in Chapter 7.8.

After a picture is defined, it can be used by a TPS application program via calls to the FOCUS library system. These calls are listed in Appendix F. For a complete description of the FOCUS screen handling system, see the FOCUS Screen Handling System manual.

3.2.3.2 Local or Remote Asynchronous Terminals

If the load of the system becomes too big, the processing part of the screen handling can be distributed to one or several Front End CPUs (*See Below*).

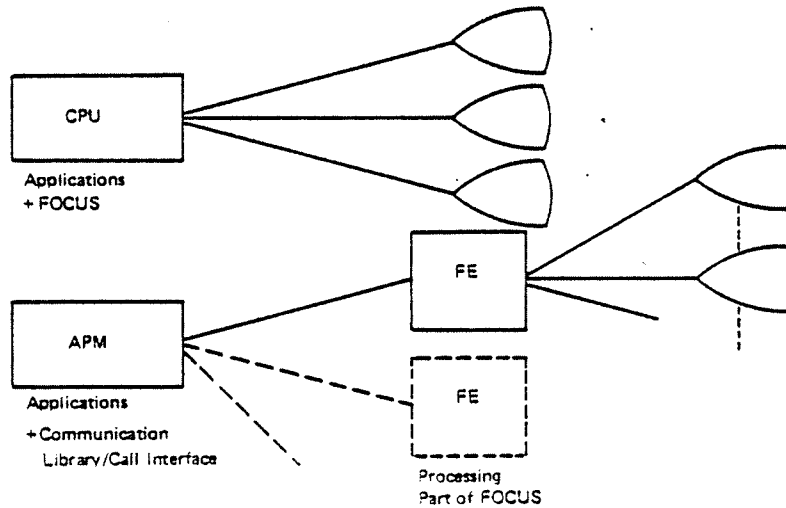


Figure 3.5B: FOCUS on One or Several CPUs

Those terminals that are to be used on a FE CPU are defined in the user configuration (*See chapter 4.2.1.6 in the TPS System Supervisor's Guide*).

For each terminal the logical device number, terminal type and CPU-number is defined. This information is sent to the FE CPU by the SIGNON special application when establishing a session between the local application and the remote CPU process.

3.2.3.3 Synchronous/Buffered Terminals Using FOCUS

A special version of FOCUS is developed for buffered terminals using I/O-modules.

The calls to FOCUS are the same, but some more calls are added to enable use of the special features of these terminals, such as set high/low intensity and lock/unlock fields. The FOCUS calls operating on just one field are not relevant for these terminal types. The forms must be compiled by a post processor before use.

3.2.3.4 ND-100 — ND-500 Incompatibilities in FOCUS

Application program using FOCUS in the ND-500 cannot switch (using the TSWAP TSR-routine) to a new application in ND-100 and continue to use FOCUS with the FOCUS-initiation set by the previous application in ND-500 and vice versa. The new application must then call 'FCINITE' as the first call to FOCUS. The FOCUS internal data format is different in the ND-100 and the ND-500 due to the difference in word-length of the two machines.

3.3 SPECIAL TPS DEVICES

Special TPS devices are devices that are controlled by I/O modules (IOMs). The main use for IOMs is in connection with networks and distributed processing, but they may also be used for any other devices that are not controlled directly by the application.

The IOM does the actual reading and writing on the device according to the protocol required by the device. After being read, data is transformed to the internal TPS message format and sent to the application program via the TPS message routing system. Data from the application program to the device is sent to the IOM in the internal protocol and written on the device by the IOM in the device format.

All handling of special message protocols, formatting, unformatting and errors is done by the IOM. The application program uses a set of simple calls to communicate with devices belonging to this group (*See Figure 3.6*). Note that some terminals controlled by IOMs may be accessed from the application program through NSHS calls. Using NSHS is discussed in section 3.2.

In addition to the TSR routines described in this chapter, the TBRDC and TTEXT TSRs can be used to broadcast a message to one or all units controlled by an IOM. These TSRs are described in chapter 4.

Input/output modules are not a part of the default TPS system, since the number of different types of devices they can handle is practically unlimited. Several IOMs have been written for the most common devices and these can be acquired as TPS options.

<i>TSOPN</i>	Open a session with device or application program
<i>TSCLO</i>	Close the session
<i>TSEST</i>	Read session status information
<i>TSMMSG</i>	Send a message to the session partner
<i>TRMSG</i>	Read a message from the session partner

Figure 3.6: Communication TSRs for Special TPS Devices

3.3.1 Session Request from a Device

The connection between a special TPS device and an application program is called a *session*. In order to establish this connection, a *session request* is sent from one of them.

A session request from a device will result in the allocation of a TPT and the starting of a transaction. The steps in accomplishing this are (See Figure 3.7A):

- the device sends a special message to the IOM requesting a session
- the IOM sends a session request to the TCM with the application name and the logical device unit as parameters
- the TCM allocates a free TPT to the session and sends the session request on to the TPT
- the TPT registers the logical unit as session partner, sends a session response back to the IOM and starts the application program
- the IOM registers the session response and connects the address of the TPT with the device unit

As long as the session lasts, the IOM will send messages from that device to the correct TPT and the TPT will send messages from the application program to the device via the IOM. The device and the application program are *session partners*. The application program communicates with its session partner through the read-message and send-message TSR routines, the device communicates through the IOM. The communication mode is half-duplex.

Note that up to 2000 bytes of data may be sent with the session request. When the application program is started, this data will be placed at the beginning of the task common data area. (See section 7.1 for a discussion of data areas.)

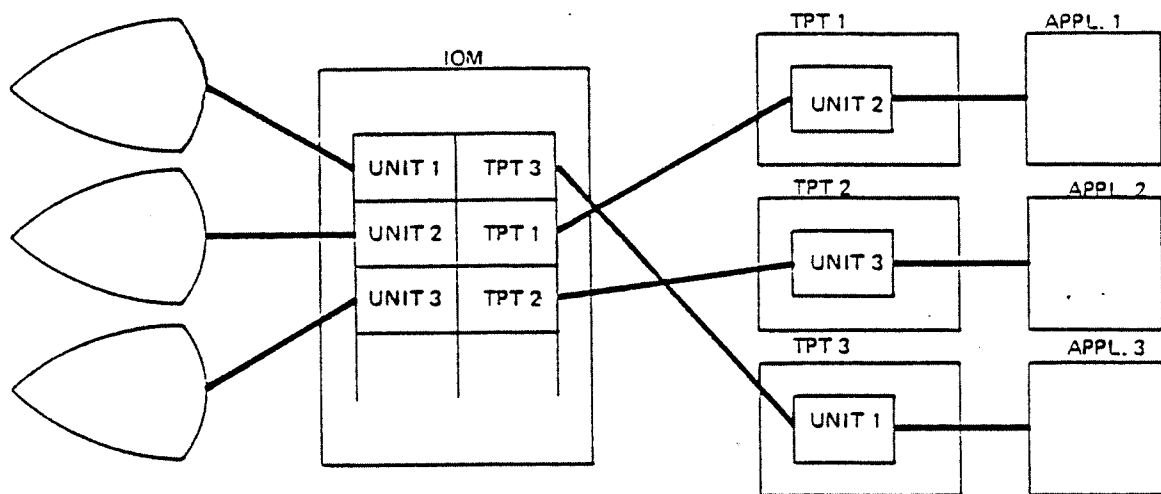


Figure 3.7A: Device-Application

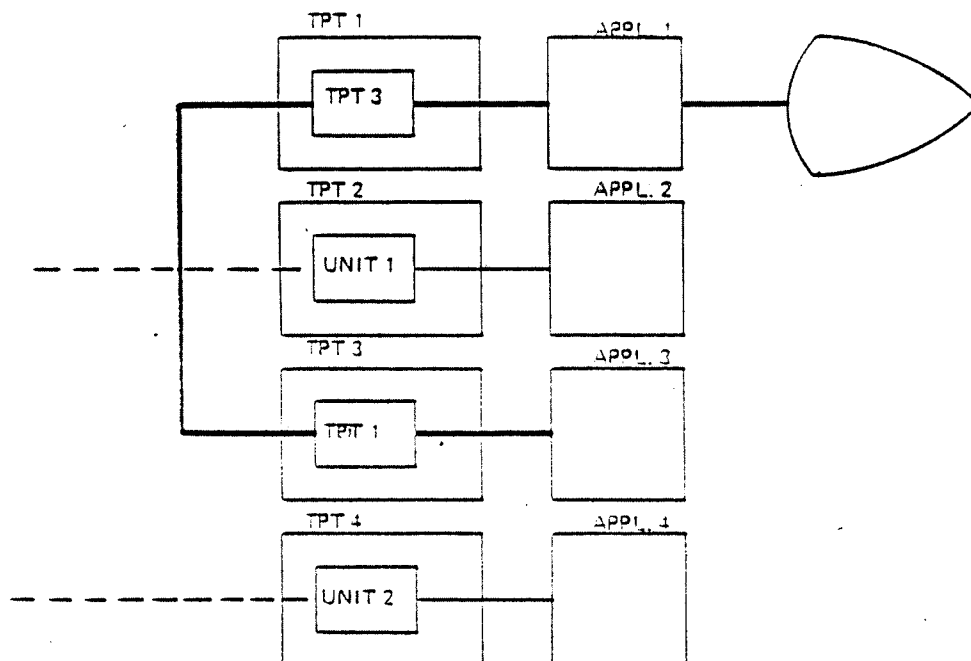


Figure 3.7B: Application-Application

SESSIONS WITHIN SAME TPS SYSTEM

3.3.2 TSOPN — The Open - Session TSR

An application program is only allowed to have one session partner at a time. If it has no session partner, it may establish a session. This session may be to a device controlled by an IOM, but it does not have to be; it could be to another application program. It could also be to a device or an application program in another TPS system (*see below*).

A session is established from an application program with the TSOPN TSR.

```
CALL TSOPN ( <module>, <sub-address>, <record>, <size>, <more>,
<status> )
CALL 'TSOPN' USING <module> <sub-address> <record> <size>
<more> <status>.
```

Within a single TPS system, the sub-address parameter is the identification of the session partner, either a device or an application program. The module is the identification of the module controlling the unit, an IOM if device, a TCM if application program.

Example of TSOPN to application program - FORTRAN

```
DIMENSION IMOD(4),IAPPL(3)
CHARACTER TCM*4
EQUIVALENCE (TCM,IMOD(3))

IMOD(1)=2          addr type is char string
IMOD(2)=4          length = 4 bytes
TCM='TCM1'         module is 'TCM1'
IAPPL(1)=1         sub-addr is appl nr
IAPPL(2)=2         length = 2 bytes (1 word)
IAPPL(3)=16        unit is appl nr 16
CALL TSOPN(IMOD,IAPPL,0,0,0,ISTAT)  open the session
IF (ISTAT.NE.0) GO TO error routine  check return status
```

Example of TSOPN to application program - COBOL

```
MOVE 2 TO MOD-ADD-TYPE.      addr type is char string
MOVE 4 TO MOD-ADD-SIZE.      length = 4 bytes
MOVE 'TCM2' TO MOD-NAME.     module is 'TCM2'
MOVE 1 TO APPL-ADD-TYPE.     sub-addr is appl nr
MOVE 2 TO APPL-ADD-SIZE.     length = 2 bytes
MOVE 22 TO APPL-NUMBER.      unit is appl nr 22
MOVE 2000 TO REC-LENGTH.     2000 bytes of data
MOVE 1 TO MORE.              and more to follow
                              open the session
CALL 'TSOPN' USING MODULE APPLICATION DATA-REC
                              REC-LENGTH MORE STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO ERR-ROUTINE.  check return status
```

Example of TSOPN to device - FORTRAN

```

DIMENSION IMOD(3),IUNIT(3)
CHARACTER IREC*80
.
.
IMOD(1)=1          addr type is the TPS module nr
IMOD(2)=2          length=2 bytes (1 word)
IMOD(3)=22B        TPS module nr of IBM3270 emulator
IUNIT(1)=1         sub-addr is the unit nr
IUNIT(2)=2         length=2 bytes (1 word)
IUNIT(3)=20        unit is channel nr 20
                   open the session and send 80 bytes
CALL TSOPN(IMOD,IUNIT,IREC,80,0,ISTAT)      of data
IF (ISTAT.NE.0) GO TO error routine  check return status

```

Example of TSOPN to device - COBOL

```

MOVE 2 TO MOD-ADD-TYPE.      addr type is a character string
MOVE 4 TO MOD-ADD-SIZE.      length=4 bytes
MOVE 'SX25' TO MOD-NAME.     module is X.25 I/O module
MOVE 3 TO SUB-ADD-TYPE.      sub-addr is in native mode for SX25
MOVE 12 TO SUB-ADD-SIZE.     length=12 bytes
MOVE X25-NUMBER TO SUB-ADD-NAME. unit is X.25 number
                           open the session
CALL 'TSOPN' USING MODULE SUB-ADDR ZERO ZERO ZERO STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO ERR-ROUTINE. check return status

```

If the session is with a device or application program in another TPS system, the module parameter will be an IOM controlling intersystem communication. Further addressing is IOM dependent and will be contained in the sub-address and/or data record.

3.3.3 Session Request from an Application

A session request to a device in the same TPS system will cause the device (if it is free) to be allocated to the application program in the same type of session as described above. The session is set up as follows (*See Figure 3.7A*):

- the application program calls TSOPN with the IOM and the device unit as parameters
- the TSOPN TSR sends a session request to the IOM
- the IOM registers the TPT as session partner for that device unit and sends a session response to the TPT
- the TPT registers the unit as session partner and returns to the application program

The device and the application program can now communicate as above.

An application program can also establish a session with another application program. The second application will be started and the session established as follows (*See Figure 3.7B*):

- the first application program (task TPTA) calls TSOPN with the TCM and the new application number as parameters
- the TSOPN TSR sends a session request to the TCM
- the TCM allocates a free TPT (TPTC) and sends the session request on to it
- TPTC registers TPTA as session partner, sends a session response to TPTA and starts its application program
- TPTA registers TPTC as session partner and returns to its application program.

The two application programs run concurrently and communicate through the send-message and read-message TSRs. They should be synchronised by using the 'more' parameter.

Sessions with programs and devices in other systems are requested in the same way by the application program, through the TSOPN TSR, but it may be necessary to specify some addressing information in the data record.

TSOPN for a session in another TPS system may look like this:

Example

```

DIMENSION IMOD(3),IUNIT(3),IBUF(7)
CHARACTER TCMX*4
EQUIVALENCE (TCMX,IBUF(3))
.
IMOD(1)=1
IMOD(2)=1
IMOD(3)=17                      intersystem IOM = X.25
IUNIT(1)=1
IUNIT(2)=1
IUNIT(3)=2                      channel 2
IBUF(1)=1                      additional addr info in IBUF
IBUF(2)=20000B                 TCMO
IBUF(3)=12                     appl 12
CALL TSOPN(IMOD,IUNIT,IBUF,6,0,IST)
IF (IST.NE.0) GO TO ERROR

```

For detailed information on establishing sessions, see the description of the particular IOM being used.

Figure 3.8A shows a session between a TPS application program and another computer system. Figure 3.8B shows one between application programs in two TPS systems.

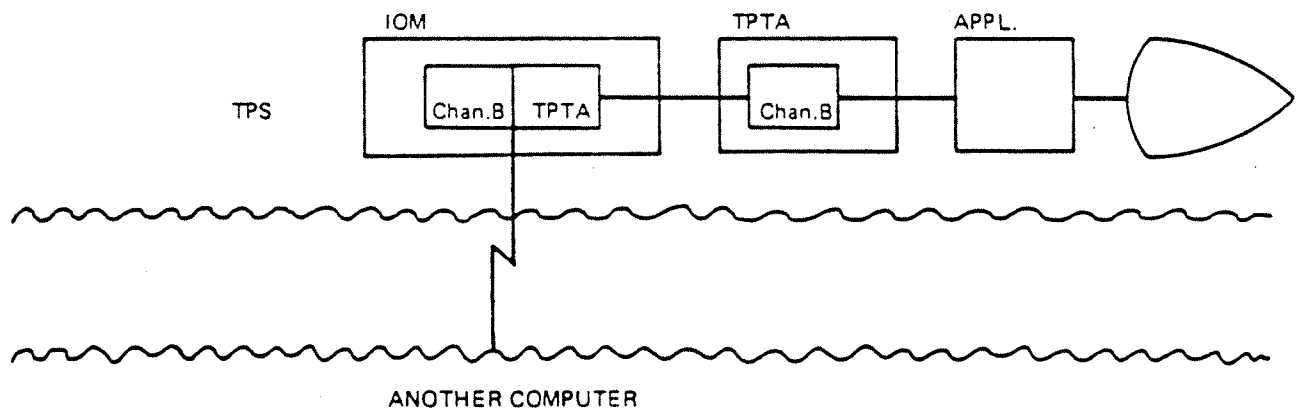


Figure 3.8A: Application-Computer

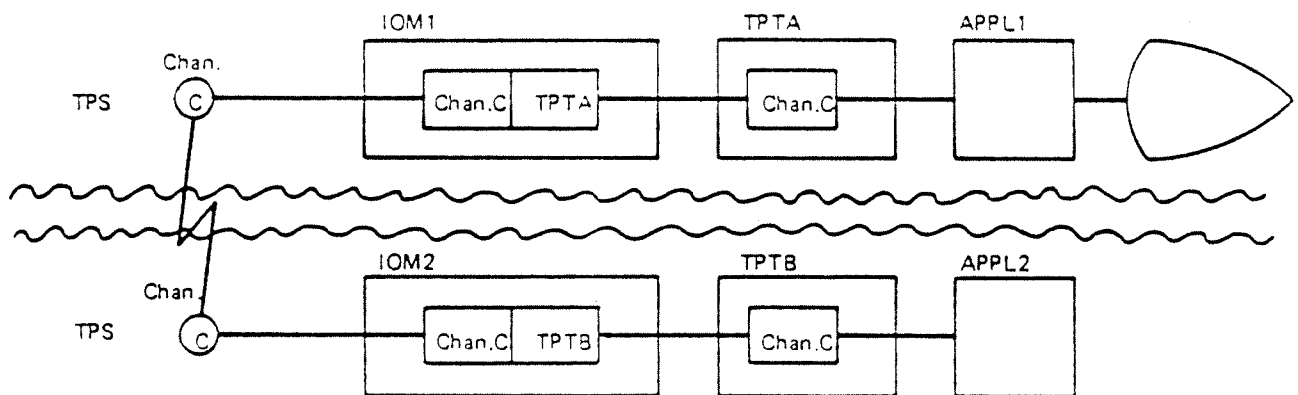


Figure 3.8B: Application-Application

SESSIONS BETWEEN TWO TPS SYSTEMS

3.3.4 TSCLO - The Close Session TSR

A session may be broken by either of the session partners by sending a session-finished message to the partner. This is done by the application program with the close-session TSR.

```
CALL TSCLO ( <status > )
CALL 'TSCLO' USING <status >.
```

Examples

```
CALL TSCLO(ISTAT)
IF (ISTAT.NE.0) GO TO session-not-closed
```

```
CALL 'TSCLO' USING STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO SESSION-NOT-CLOSED.
```

This will cause the following to happen:

- a finish-session message will be sent to the IOM or TCM
- the IOM will free the device and send a session-finished message to the TPT
or
- the TCM will send a session-finished message to the other TPT (but it will not be freed)
- the TPT will return to the application program

Note that sessions will be automatically broken when transactions terminate completely and the TPT is freed (but not when they switch application programs or return to SIGNON or SELECT).

3.3.5 TSEST - The Session Status TSR

If the application program does not know if it has a session if it has reason to believe that a session may have been broken, or if it wants information about the current session, the read-session-status TSR can be used. The TSR is used mainly by the RESTART special application, but is also available to user application programs (See Figure 3.9).

```
CALL TSEST ( <record> )
CALL 'TSEST' USING <record>.
```

Examples

```
DIMENSION IREC(20)
CALL TSEST(IREC)

CALL 'TSEST' USING SESSION-INFO-REC.
```

1	Session state
2	Current direction
3	No. of input messages
4	Time for latest input message (year, month, day, hour, minute, second, BTU)
11	No. of output messages
12	Time for latest output message
19	Session partner-module
20	Session partner-unit

Figure 3.9: Session Information

3.3.6 TSMMSG - The Send-Message TSR

When an application program wants to send data to the session partner, it will prepare an array/record in working storage and send it to the partner with the send-message TSR:

```
CALL TSMMSG ( <record > , <size > , <more > , <status > )
CALL 'TSMMSG' USING <record > <size > <more > <status>.
```

Example - FORTRAN

```
DIMENSION ITEXT(1000)           message defined as array
CHARACTER CTEXT*2000           message defined as character string
EQUIVALENCE (ITEXT(1),CTEXT)

CTEXT='MESSAGE TO SESSION PARTNER'
CALL TSMMSG(ITEXT,26,0,ISTAT) send message(26 bytes),no more to follow
IF (ISTAT.NE.0) GO TO error routine    check return status
```

Example - COBOL

```
MOVE 'MESSAGE TO SESSION PARTNER' TO MESSAGE-TEXT.
MOVE 26 TO MESSAGE-SIZE.
MOVE 0 TO MORE.
CALL 'TSMMSG' USING MESSAGE-TEXT  send message (26 bytes), no more
                                MESSAGE-SIZE MORE STATUS CODE.      to follow
IF STATUS-CODE NOT = 0 GO TO ERR-ROUTINE.  check return status
```

The message will be copied from working storage to a buffer area and sent to the session partner in the form of a data message. The TSR will then return immediately to the application program without waiting for an answer from the session partner.

Note that there is a flag, the 'more' flag, that can be used to indicate whether the application program intends to send more data before expecting an answer. The session partner can then test this flag when the data is read with the read message TSR. If a message in one direction is to be followed by another in the same direction, the more bit is set. For the last message, the more bit will be cleared. In the case of read-message, this means that if the bit is set the application program should call read-message again to get the next message before an answer is sent. In the case of send-message, the application program will set the bit if a new message is going to be sent before waiting for an answer.

3.3.7 TRMSG - The Read Message TSR

When the application program wants to receive data from the session partner, it will call the read-message TSR:

```
CALL TRMSG ( <record > , <size > , <more > , <status > )
CALL 'TRMSG' USING <record > <size > <more > <status>.
```

Example - FORTRAN

```
DIMENSION ITEXT(50)           message area defined as array
CHARACTER CTEXT*100          message area defined as
EQUIVALENCE (ITEXT(1),CTEXT)  character string

ISIZE=100                     max length 100 bytes
CALL TRMSG(ITEXT,ISIZE,MORE,ISTAT)  read message
IF (ISTAT.NE.0) GO TO error routine  check return status
IF (MORE.EQ.0) GO TO last input message  check for more input
```

Example - COBOL

```
MOVE 100 TO MESSAGE-SIZE.      max length 100 bytes
CALL 'TRMSG' USING             read message
    MESSAGE-TEXT MESSAGE-SIZE MORE STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO ERR-ROUTINE.  check return status
IF MORE = 0 GO TO LAST-INPUT-MSG.  check for more input
```

When this TSR is called, the TPT will see if any message has come from the session partner. If it has, it will copy the message to the record area in working storage and return to the application program. If none has come yet, the TPT will wait until a message arrives.

3.3.8 TPASZ — The Set Packet Size TSR

Session partners exchange messages which in turn are divided into packets by TPS. This TSR sets the size of the packets.

CALL TPASZ (<packet size>, <status>)
CALL 'TPASZ' USING <packet size>, <status>.

The packet size set with this TSR will only be used for this transaction. If the transaction does not set the packet size, the default size at system generation will be used.

The maximum packet size allowed is 2047.

Examples

```
CALL TPASZ(2000,ISTAT)

CALL 'TPASZ' USING PACKET-SIZE STATUS-CODE
```

3.3.9 Restart

If a system failure occurs, the system can be restarted again with rollback or recovery (*See Chapter 5*). After a system restart, some sessions may be intact while others may be broken. Sessions between two application programs in the same system will probably be intact, since the application programs have both been restarted at their checkpoints. IOMs, however, do *not* take checkpoints and therefore cannot be rolled back. In addition they may have been reloaded and lost all session information, or connections may have been broken externally if the system was down for any length of time.

The RESTART application may try to restore broken sessions for transactions with restart at checkpoint (*see section 5.3.2.2*) and it should break sessions for other types of transactions. The special TSR TSEST (read-session-status) is available for this. It can also use TSOPN to create a new session. Sessions may therefore be intact when the application program regains control after system restart.

The function of checking and restoring broken sessions in the RESTART application must be programmed by the user. The standard version of RESTART only restores connections with SIBAS and NSHS.

3.3.10 Available Input/Output Modules

The input/output modules that are available at present are

- ISO1745 for communicating with terminals using the ISO-1745 protocol (STANSAAB Alfaskop 3500 terminals)
- X25LAPB for communicating with other systems using the X.25 protocol
- IBM3270-HOST for communicating with terminals on a control unit using the IBM-3270 protocol, i.e. the NORD CPU communicates with the 3270s
- IBM3270-CU for *emulating* an IBM 3270 Control Unit communicating with some other equipment, i.e. the NORD CPU *is* a 3270

A brief description of how to program them is given here. They are discussed in more detail in the TPS System Supervisor's Guide.

3.3.10.1 X25LAPB

The X25LAPB module can be used for communication between two or more TPS systems or between a TPS system and another TP monitor using the X.25 communications protocol. For example, it could be used for communication between a NORD machine with TPS and a CENSOR 932 machine. Figure 3.10 illustrates two possible X.25 configurations.

The communication protocol consists of 4 levels. Levels 1 and 2 correspond to the two lowest levels of X.25. Level 3 is a subset of X.25/3. Level 4 corresponds to the TPS level. It is here that sessions are established, user data is transmitted, and sessions are terminated.

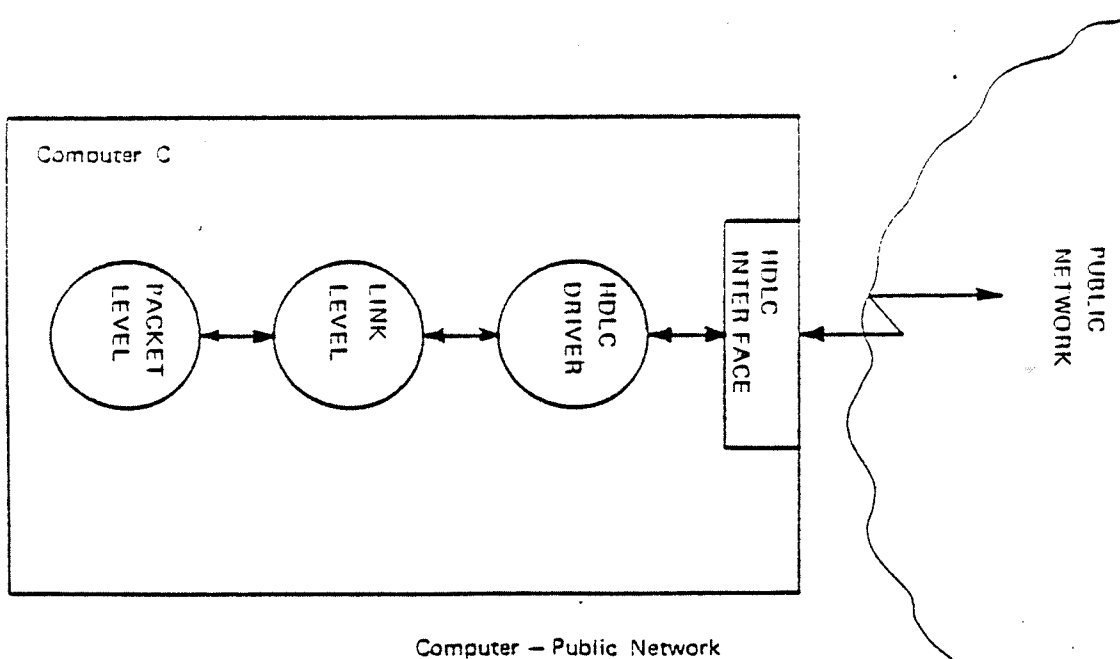
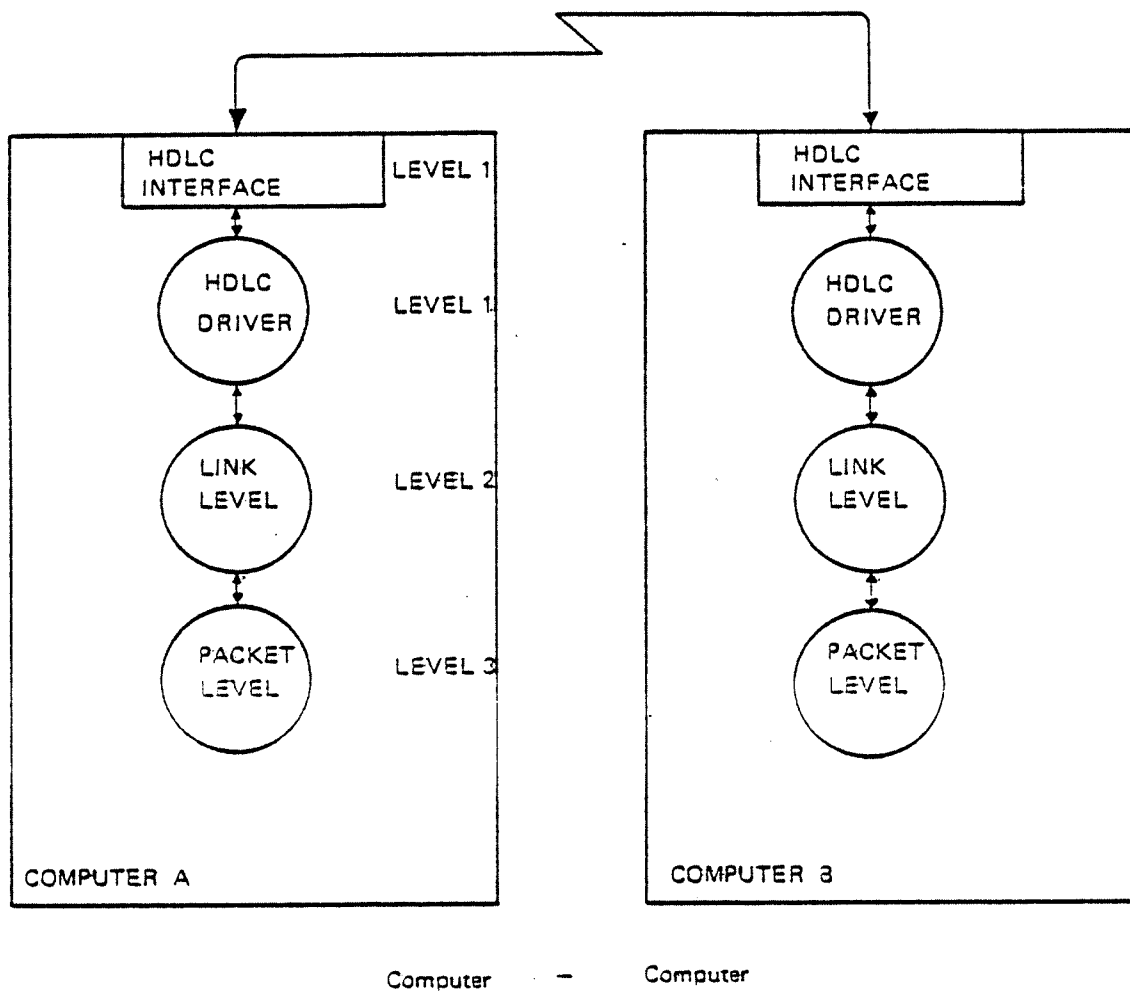


Figure 3.10: X.25 Communication

The usual TSR routines are used to communicate with a remote partner

- TSOPN Open a session
- TSCLO Close a session
- TSMSG Send a message
- TRMSG Read a message

When calling TSOPN, the number of the communication channel must be specified. If the specified channel is already in use, the call will return an error status of -3, unit temporarily not available. Usually in a 2-system configuration one system will use even numbers in TSOPN, the other one odd numbers.

In addition, the TSOPN call must specify the session partner, i.e. the TCM number and application number. This is given at the beginning of the data record sent with the TSOPN call. User data can also be sent with TSOPN, but the size is limited to the packet size that can be sent over the line (usually 128 or 256 bytes).

The application that is started up must reply with a data message (sent with TSMSG). This message may be empty (dummy reply).

Example

1. The Application Setting up the Connection

```

PROGRAM APOXX,YY
.
.
C
C Identify X.25 I/O module
C
IMODL(1)=1
IMODL(2)=1
IMODL(3)=17                                TPS module number
C
C Identify channel number
C
IUNIT(1)=1
IUNIT(2)=1
IUNIT(3)=2                                channel number
C
C Identify session partner
C
IARR(1)=1
IARR(2)=20000B                            TCMO
IARR(3)=12                                appl number
C
C Open the session
C
CALL TSOPN(IMODL,IUNIT,IARR,6,0,ISTAT)
C
C Read reply from partner
C
CALL TRMSG(.....)
C
C Dialogue
C
CALL TRMSG/TSMSG
C
C Close the session
C
CALL TSCLO
.
.
END

```

2. The Started Application

```
PROGRAM APOZZ,XX
.
.
C
C Send reply message
C CALL TSMSG(.....)
C
C Dialogue
C CALL TRMSG/TSMSG
.
.
END
```

3.3.10.2 IBM—3270—CU

The IBM—3270—CU module emulates an IBM—3270 Control Unit. A NORD machine with TPS and the IBM—3270—CU module can be connected to an IBM machine and look like terminals using the IBM 3270 communication protocol.

IBM—3270—CU was developed as part of the Nortrygd project. In its present form it is not a completely general product, but a communication module that satisfies the requirements for the Nortrygd project. It contains both simplifications and special features such as:

- ASCII/EBCDIC conversion:
All messages to the IBM machine are converted from ASCII to EBCDIC before being sent and all messages received are converted from EBCDIC to ASCII. As a consequence of this, it is impossible to send or receive messages containing IBM buffer control orders such as SBA, SF, EUA, etc., because they are followed by addresses that should not be converted.
- Generation of "ENTER" and "cursor address":
The present version of IBM3270—CU will only send the type of message resulting from pressing the ENTER key. Pressing function keys cannot be simulated, nor can different cursor addresses.
- System and operator messages:
These messages from the IBM machine are routed to the TPS operator's console.
- Messages from unused channels:
Messages received on channels not "in session" are ignored
- Long messages from TPS to IBM:
IBM—3270—CU can send messages of any length to IBM if they are received from TPS as one packet, but messages that are divided into several packets cannot be sent as one message
- Session request from IBM:
This is not possible — the NORD machine must initiate the session

Application programs communicate with the IBM machine with the usual session TSR routines.

- TSOPN Open a session
- TSCLO Close a session
- TMSG Send a message
- TRMSG Read a message

The TSOPN call specifies the module number (18) and the unit number (always 1). In addition, the first data message must be sent in TSOPN.

IBM—3270—CU can receive messages of any length from IBM and send them on to TPS modules divided into several packets if necessary. However, it is not possible to receive several packets from TPS modules and put them together into one message to be sent to IBM.

Messages received from IBM will be sent to the session partner only if the first character is "I". All other messages are considered system messages and are routed to the TPS operator's console.

There is no individual timeout for a session, but a timeout for the whole line. If the IBM machine stops sending, all sessions will be broken and the message "COMMUNICATION DEAD" written on the operator's console. At the present time, this will happen 35 seconds after the last poll or message has been received from IBM. Upon receipt of the first poll after the line has been down, the message "COMMUNICATION RUNNING" will be written on the console.

A session request (TSOPN) given when the line is down will result in a negative session response.

Example

```

      .
      .
      .
      IMOD(1)=1
      IMOD(2)=1
      IMOD(3)=18
      IUNIT(1)=1
      IUNIT(2)=1
      IUNIT(3)=1
C
C   The first message to IBM is put into IMESS
C
      ILEN=number of bytes in IMESS
      CALL TSOPEN(IMOD,IUNIT,IMESS,ILEN,0,ISTAT)
      IF (ISTAT.NE.0)THEN error
      .
      .
      CALL TRMSG(IMESS,ILEN,IMORE,ISTAT)
      .
      .
C
C   Dialogue with TSMMSG/TRMSG
C
      CALL TSMMSG/TRMSG
C
C   Done, close session
C
      CALL TSCLO(ISTAT)
      .
      .
      END

```

3.3.10.3 IBM—3270—HOST

The IBM—3270—HOST module acts as a host machine to IBM—3270 terminals. A NORD machine with TPS and this I/O module can be connected, through synchronous modem lines, with terminals using the IBM 3270 line procedure. The NORD machine will look like an IBM host machine to these terminals.

This module was originally made in order to use Alfaskop System 41 terminals, but it can also be used with other equipment using the same line procedure.

The IBM—3270—HOST module uses EBCDIC characters, but can use ASCII characters instead if modifications are made to SINTRAN III.

Communication between an application program and a terminal is done through the session TSR routines TSMSG/TRMSG.

A session will usually be set up by the terminal operator. This is done by pressing SEND with a blank screen. A session will be set up between the terminal and a transaction processing task (TPT) belonging to TCM0, and the SIGNON application will be started. This must be a special non-NSHS version.

If the screen is not blank when SEND is pressed, it will be blanked and SEND must be pressed again.

If another TCM than TCM0 is wanted, a special version of IBM—3270—HOST must be used.

An application program can also set up a session. This is done with TSOPN, specifying the IBM—3270—HOST module number (16) and the logical unit number of the terminal.

After the session has been set up, data can be sent and received with TSMSG and TRMSG. Note that every time the terminal user presses SEND, a message will be sent to the application program and the keyboard will be locked until the application program sends a message back to the terminal.

Output to the printer should be sent with the TTEXT routine. It is not possible to have a session with a printer.

The NSHS screen handling system cannot be used.

3.3.10.4 ISO—1745—HOST

The ISO—1745—HOST module acts as a host machine to ISO—1745 (STANSAAB Alfascop 3500) terminals. A NORD machine with TPS and this I/O module can be connected, through synchronous modem lines, with terminals using the ISO—1745 line procedure. The NORD machine will be a host to these terminals.

Communication between an application program and a terminal is done through the session TSR routines TSMMSG/TRMSG or with the NSHS screen handling system.

A session will usually be set up by the terminal operator. This is done by pressing SEND with a blank screen. A session will be set up between the terminal and a transaction processing task (TPT) belonging to TCM0, and the SIGNON application will be started. If NSHS is not used, a special non—NSHS version of SIGNON must be used.

If the screen is not blank when SEND is pressed, it will be blanked and SEND must be pressed again.

The standard TPS version of NSHS simulates INBT/OUTBT, but uses the buffer pool system if the terminal type is equal to 6. It will be set automatically to 6 if the session request comes from the I/O module (i.e. the terminal).

If another TCM than TCM0 is wanted, a special version of ISO—1745—HOST must be used.

An application program can also set up a session. This is done with TSOPN, specifying the ISO—1745—HOST module number (16) and the logical unit number of the terminal.

After the session has been set up, data can be sent and received with TSMMSG and TRMSG. Note that every time the terminal user presses SEND, a message will be sent to the application program and the keyboard will be locked until the application program sends a message back to the terminal.

Output to the printer should be sent with the TTEXT routine. It is not possible to have a session with a printer.

It is possible to send broadcasts to terminals using TBRDC/TTEXT. A session is not necessary then.

3.4 STANDARD DEVICES AND FILES

Standard input/output devices, such as card readers, magnetic tapes, paper tape readers and punches, spooling files and other files using the SINTRAN file system (not SIBAS files), are controlled using the standard I/O statements available in the programming languages used.

In addition, FORTRAN, PLANC, MAC and NPL programs may call many (not all) of the SINTRAN I/O monitor-call subroutines available to RT (real-time) programs. COBOL programs may also use these monitor call routines, but some of them (most notably the OPEN monitor call) cannot be called directly because of incompatible character string parameters. Monitor call I/O is therefore not recommended in COBOL.

This section discusses the most common I/O facilities available to TPS application programs. Appendix E contains a complete list of all monitor call routines available. For a detailed description of these routines, see the SINTRAN III Reference Manual.

3.4.1 Allocating Standard Devices and Files

Since TPS application programs are run in a real-time environment, devices and files cannot be allocated ahead of time as for batch jobs. When a program needs a device or file, it acquires it through either the OPEN statement or the RESRV (reserve) monitor call.

OPEN must be used for files, since this call contains file oriented parameters. In FORTRAN, PLANC, NPL and MAC programs, RESRV will normally be used for devices since this is faster and device oriented. However, RESRV requires the SINTRAN device number as a parameter, and this may be unknown. If the device has been defined as a peripheral file (*See SINTRAN Users's Guide, Periferal Devices*), OPEN can be used with the name of the peripheral file as file name.

All devices and files in COBOL programs are allocated in the standard COBOL manner, with file definition (FD) entries, SELECT entries, and OPEN statements.

Note that a feature called "direct file transfer" is available for applications running in the ND-500: access mode D or DC (8 or 9). This feature allows very high disk transfer speeds between disk and memory with a minimum of system overhead. Further information may be found in the ND-500 - Loader/Monitor manual, ND-60.136.

3.4.2 Unavailable Devices and Files

Provision should be made in the application program for unavailable devices and files. An unavailable device or file will cause an error return from OPEN or RESRV. This can be tested in FORTRAN or PLANC by examining the system integer variable ERRCODE or a function value like ISTAT. Any non-zero value indicates an error, while -1 in ISTAT indicates that a device cannot be reserved (RESRV does not set ERRCODE) and the decimal values 69, 77, 78, 98, 107 and 110 indicate that a file is unavailable for OPEN (*See FORTRAN Ref. Man, Run-time Errors*).

COBOL programs can test for errors through the FILE STATUS entry of the SELECT statement and the USE sentence. The file status word will be set to '30' (permanent error) if the file is unavailable. If the USE sentence is not included in the program, the program will be abnormally terminated when a file or device is unavailable.

RESRV has a wait/no-wait option. If the no-wait option is used and the device is not available, the RESRV routine will return to the program immediately with -1 in ISTAT. However, the program can choose instead to wait until it is available. OPEN has no such option; return will always be immediate. A wait loop using the HOLD routine may be programmed to wait until the device is available.

Example of OPEN - FORTRAN

```

                                open the file
10  OPEN(FILE='MYFILE:DATA ',UNIT=IFILNR,ACCESS='WX',ERR=20)
    .                               file opened - continue
    .
                                file not opened - test ERRCODE
20  IF (ERRCODE.EQ.69 OR ERRCODE.EQ.77 OR ERRCODE.EQ.78
    *OR ERRCODE.EQ.98 OR ERRCODE.EQ.109 OR ERRCODE.EQ.110)THEN
    CALL HOLD(10,2)               file unavailable, wait 10 seconds
    GO TO 10                      and try again
ELSE                             else something else wrong
    CALL QERMS(ERRCODE)          stop with QERMS error message
ENDIF
```

Example of RESRV - FORTRAN

```

      INTEGER RESRV,INCH
      CHARACTER TEXT1*100,TEXT2*100

10  ISTAT=RESRV(2,0,1)           reserve device nr 2 (tape-reader)
    IF (ISTAT.NE.0) GO TO 30      check the status
20  CALL TWMSG(TEXT1)            OK, tell operator to put tape in reader
    ICHAR=INCH(2)               read a byte
    IF (ERRCODE.NE.0) GO TO error routine  check ERRCODE
      .
      .
30  CALL TWMSG(TEXT2)            device not reserved, tell operator
    ISTAT=RESRV(2,0,0)           reserve again with wait flag
    IF (ISTAT.EQ.0) GO TO 20     check again
                                  still not reserved, something else wrong
    CALL TSTOP(ICODE)           stop with TPS error message

```

Example of OPEN - COBOL

```

      INPUT-OUTPUT SECTION.
      FILE-CONTROL.
                                select the tape-reader
                                SELECT IN-FILE ASSIGN "TAPE-READER" STATUS IS FSTAT.

      FILE SECTION.
      FD      IN-FILE.
            01 IN-DATA PIC X(100).

      PROCEDURE DIVISION.
      DECLARATIVES.
      I-O-ERR SECTION.
            USE AFTER ERROR PROCEDURE ON IN-FILE.
            T-R-ERROR.
                                tape-reader error
                                IF FSTAT NOT = '30'      file unavailable?
                                                                no, something else wrong, stop
                                CALL 'TSTOP' USING ABEND-CODE-1.
                                CALL 'HOLD' USING TEN TWO.  yes, wait 10 seconds
                                GO TO T-R-OPEN.             and try again
      END DECLARATIVES.
      .
      .
      OPEN-T-R.
            OPEN INPUT IN-FILE.                                open the tape-reader file
      .

```

3.4.3 Accessing Standard Devices and Files

There are various ways of accessing devices and files, (See figure 3.11), but they can be divided into 2 main types:

- standard I/O statements
- special SINTRAN monitor calls

FORTRAN standard I/O statements are READ and WRITE for formatted and binary I/O, and INPUT and OUTPUT for free format I/O. These statements can be used after both OPEN and RESRV.

COBOL standard I/O statements are READ, WRITE and REWRITE. ACCEPT, DISPLAY and EXHIBIT are only allowed in background mode during program testing on the ND-100.

SINTRAN monitor call I/O can be used in FORTRAN, NPL and MAC programs. The most common monitor calls are INCH and OUTCH for character I/O and RFILE and WFILE for block I/O. INSTR is not allowed under TPS (but OUTST is). Monitor call I/O can be used after both OPEN and RESRV. Note that calling the ordinary SINTRAN III I/O monitor calls from the ND-500, gives a substantial amount of system overhead (ND-100 CPU-load). Some special I/O monitor calls, DV INST (MON 503) and DV OUTS (MON 504), from ND-500 applications should be used.

Application programs must remember to close files before terminating since this is not done automatically as for background programs.

A complete list of monitor calls available to application programs is found in appendix E. For a detailed description of the functions and parameters of these monitor calls, see the SINTRAN III User's Guide.

The syntax of standard I/O statements for FORTRAN and COBOL is found in the respective programmer's reference manuals.

LANGUAGE	TYPE	ALLOCATE	RELEASE	ACCESS
FORTRAN	FILES, PERIF. FILES	OPEN	CLOSE	STAND. I/O, MON. CALLS
	DEVICES	CALL RESRV	CALL RELES	STAND. I/O, MON. CALLS
COBOL	FILES, DEVICES	OPEN	CLOSE	STANDARD I/O

Figure 3.11: Local Input/Output

3.4.4 Restart

It must be pointed out that rollback and recovery facilities are not available for files controlled directly by the application program. If a system restart occurs, the contents of such files will be unpredictable. In addition, files and devices acquired by an application program may or may not have been lost, depending on such factors as whether SINTRAN has been restarted, whether the application program has been reloaded, and whether files have been closed by the operator.

The RESTART application can attempt to restore resources, but even if this is possible, problems can arise. If a sequential file is lost and allocated again, processing will start over from the beginning, and the application program should therefore probably be restarted at the beginning. If printer spooling is used, the application can continue from the middle, but output before and after the break may go to separate spooling files. These things should be taken into consideration when writing application programs, modifying the RESTART application and deciding which restart strategy to use (*See Chapter 5*).

4

OTHER TPS AND SINTRAN FACILITIES

Several other TPS and SINTRAN facilities are available to TPS application programs in addition to those described in chapters 2, 3 and 5. These may be grouped as follows:

- message handling
- clock routines
- hold monitor call
- semaphores
- internal devices

These facilities are invoked either through TSR calls or SINTRAN monitor calls. The TSR routines are described in detail in this chapter and appendix H. For a detailed discussion of the monitor calls, see the SINTRAN User's Guide.

With the exception of some TSRs most of the material in this chapter describes general SINTRAN facilities. This material is included in order to make it easier for the TPS programmer to design application programs, knowing exactly which SINTRAN facilities are available.

4.1

MESSAGE HANDLING

The message handling facilities of TPS include routines for writing messages on the operator console and broadcasting messages to a group of devices or application programs.

In addition, the SINTRAN routines for writing error messages on the SINTRAN error device can be used.

The TSRs and monitor calls for message handling can be grouped as follows(See Figure 4.1):

- TWMSG and CWMSG (write message to TPS operator)
- TBRDC and TTEXT(broadcast message to one or several units)
- TGBRD and CGBRD (get broadcasted message)
- ERMSG and QERMS (write standard SINTRAN error message and continue or terminate)
- ERMON (write special ERMON message and continue)

TWMSG (FTN) CWMSG (COB/PLANC)	Write a message to TPS operator
TBRDC TTEXT	Broadcast a message to one or several units
TGBRD (FTN) CGBRD (COB/PLANC)	Get a broadcasted message
ERMSG (FTN/PLANC)	Write a standard SINTRAN message, return to application program
QERMS (FTN/PLANC)	Write a standard SINTRAN message, terminate the program
ERMON (FTN/PLANC and COB)	Write the special ERMON message, return to application program

Figure 4.1: System Messages

4.1.1 TWMSG and CWMSG - The Write Message to Operator TSR

The write-message-to-operator TSR will write a text string on the operator's console. The message will be supplied with time, date and source identity by TPS. This TSR has a special COBOL and PLANC version, CWMSG, due to incompatible character string formats.

```
CALL TWMSG ( <text-string> )  
CALL CWMSG USING <text-string>.
```

Examples

```
CHARACTER MTEXT*256
```

```
MTEXT='MESSAGE TO OPERATOR'''  
CALL TWMSG(MTEXT)
```

```
MOVE "MESSAGE TO OPERATOR'" TO MESSAGE-TEXT.  
CALL 'CWMSG' USING MESSAGE-TEXT.
```

The text must be terminated by an ' (apostrophe) and may not exceed 255 characters.

4.1.2 TBRDC - The Broadcast Message TSR

A message can be broadcast from an application program to all terminals, all active terminals, or a single terminal connected to an IOM. It can also be sent to all active TPTs controlled by a TCM.

```
CALL TBRDC ( <module>, <sub-address>, <text>, <units>, <status> )
CALL 'TBRDC' USING <module> <sub-address> <text> <units> <status>
```

Example - FORTRAN

```
DIMENSION MOD(5),ITEXT(36)      array definitions
CHARACTER MNAME*6,CTEXT*72      character string definitions
EQUIVALENCE (MOD(3),MNAME),(ITEXT(1),CTEXT)

MOD(1)=2
MOD(2)=5
MNAME=32                        module is TCM0
CTEXT='MESSAGE TO ALL ACTIVE TPTS'
CALL TBRDC(MOD,0,ITEXT,1,ISTAT) broadcast the message to all TPTs
IF (ISTAT.NE.0) GO TO error routine  check return status
```

Example - COBOL

```
MOVE 1 TO MOD-ADD-TYPE.
MOVE 2 TO MOD-ADD-SIZE.
MOVE 16 TO MOD-NAME.           module is the STANSAAB IOM
MOVE 1 TO UNIT-ADD-TYPE.
MOVE 2 TO UNIT-ADD-SIZE.
MOVE 1 TO UNIT-NUMB.          unit is VDU number 1
MOVE "MESSAGE TO STANSAAB-TERMINAL-01" TO MESSAGE-TEXT.
CALL 'TBRDC' USING             broadcast to 1 terminal only
    MODULE UNIT MESSAGE-TEXT TWO STATUS-CODE.
IF STATUS-CODE NOT = 0 GO TO PARM-ERROR.  check return status
```

The text must be terminated by an ' (apostrophe) and may not exceed 72 characters.

Messages broadcast to IOM terminals will be written on the *broadcast line* (usually the bottom line).

Messages sent to TPTs can be read by the application program with the TGBRD TSR (get broadcasted message).

4.1.3 TTEXT — The Send Text Message TSR

The send text message TSR is very similar to the broadcast message TSR except that the message can only be sent to a single unit and the message length in bytes is given as a parameter instead of being indicated by an apostrophe in the text itself.

```
CALL TTEXT (<module>, <sub-address>, <text>, <length>, <status>)  
CALL 'TTEXT' USING <module> <sub-address> <text> <length>  
<status>.
```

No screen positioning is performed by the IOM so the message will be written where the cursor happens to be positioned.

Messages sent to TPTs can be read by the application program with the TGBRD TSR.

4.1.4 TGBRD and CGBRD - The Get Broadcasted Message TSR

This TSR can be used to see if a message has arrived. If it has, it will be put in the text area indicated. The message can have come from another application program using broadcast-message (TBRDC) or it can have come from the TPS operator using the BROADCAST command or the MESSAGE-TO-UNIT command.

If the application program wants to write the message on a display terminal using NSHS, the WMSGGE or CWMSGGE routine may be called. If using FOCUS, the FCZMSGGE routine may be called.

This TSR has a special COBOL and PLANC version, CGBRD, due to incompatible character string formats.

```
CALL TGBRD ( <text-string>, <status> )
CALL 'CGBRD' USING <text-string> <status>.
```

Examples

```
CHARACTER BCTEXT*72          message area defined as
                              character string
CALL TGBRD(BCTEXT,ISTAT)      get message
IF (ISTAT.EQ.0)CALL WMSGGE(BCTEXT) display it on screen

CALL 'CGBRD' USING MESSAGE-TEXT STATUS-CODE.  get message
IF STATUS-CODE = 0
    CALL 'CWMSGGE' USING MESSAGE-TEXT.  display it on screen
```

The text may not exceed 72 characters. It will be terminated by an apostrophe.

This TSR should be called fairly often in systems that make use of the broadcast facility, since this is the only way the message will be detected. There is no automatic presentation of messages broadcast to TPT-controlled terminals (in contrast to IOM-controlled terminals, where the IOM will see to it that the message is sent out to the terminals).

Another reason for checking often is that if a new broadcast message arrives, it will overwrite the old message. There is just one message buffer area and no queuing system. Therefore it is important that the message be sent to the terminal before it is overwritten.

4.1.5 Monitor Calls (ERMSG, QERMS, ERMON)

SINTRAN error message monitor calls can also be used by TPS application programs. If a SINTRAN routine detects an error, it will put the error number in the ERRCODE variable (or a function value like ISTAT); if no error, the variable will contain 0. This can be tested and if it is non-zero, the appropriate error message can be written on the SINTRAN error device by the ERMSG or the QERMS monitor call. ERMSG will write the error message and return to the application program, QERMS will write the message and terminate the transaction.

The ERMSG and QERMS monitor calls can also be used in connection with standard FORTRAN I/O, since these I/O routines also set the ERRCODE variable. Standard COBOL I/O, however, does not have this facility. Instead, the standard error facilities, the FILE STATUS entry of the SELECT statement and the USE sentence, can be used to process error conditions (*See Example 3.8*). ERMSG and QERMS cannot be used here because the SINTRAN error number is not available to COBOL programs.

Examples

WRITE(5,10,ERR=100)	standard I/O statement
.	no error, continue main routine
.	
100 CALL ERMSG(ERRCODE)	error, call ERMSG
.	continue error routine
ICHAR=INCH(IFILNR)	monitor call I/O
IF (ERRCODE.NE.0) CALL QERMS(ERRCODE)	if error, call QERMS, stop
.	else continue
.	

A special user error message can be written on the SINTRAN error device using the ERMON monitor call. An error number in the range 50-69 must be given (in ASCII code), together with a suberror number of any value (integer). The error message will be printed as follows:

hh.mm.ss ERROR nn IN rr AT ll USER ERROR SUBERROR:ss

where

hh.mm.ss	time when the message is printed
nn	<error number>
rr	TPT identification
ll	address of error in application program
ss	<suberror number>

ERMON can be called from both FORTRAN and COBOL programs.

Example - FORTRAN

IF (DATE.EQ.0) GO TO 100	check for bad date
.	
100 CALL ERMON(2H62,3)	bad date, call ERMON
.	continue

Example - COBOL

01 ERROR-NR PIC XX.	
01 SUB-ERROR-NR COMP PIC 999.	
.	
IF DATE = 0 GO TO BAD-DATE.	check for bad date
.	
BAD-DATE.	
MOVE '62' TO ERROR-NR.	
MOVE 3 TO SUB-ERROR-NR.	
CALL 'ERMON' USING ERROR-NR SUB-ERROR-NR.	call ERMON
.	continue

In both cases a message of the following type will be written on the SINTRAN error device:

11.50.03 ERROR 62 IN TPT5 AT 5320 3:USER ERROR

4.2 CLOCK ROUTINES

COBOL programs can use the ACCEPT DATE/DAY/TIME statement to examine the calendar and clock.

In addition, the SINTRAN monitor calls for examining and changing the internal clock are available to all application programs. The CLOCK routine will return the current time/date to a 7 word integer array containing basic time units, seconds, minutes, hours, day, month and year. The clock can be changed with the UPDAT call, specifying the new minute, hour, day, month and year. The clock can also be adjusted relative to its current value using the CLADJ call. The last two monitor calls (especially UPDAT) should be used with care!

SINTRAN also has an interval clock containing, in a double word, the number of basic time units since SINTRAN was last started. The TIME routine will return this current interval time. It is set to zero each time the MASTER CLEAR and LOAD buttons are pressed and is incremented by 1 each basic time unit.

4.3 HOLD MONITOR CALL

The HOLD monitor call can be used to put an application program in a wait state for a given time interval. However, the wait state will be terminated if the TPT is started for any reason, i.e. the arrival of a checkpoint message. After the message is processed, the application program will receive control as if the time interval had expired, since the TPT does not know how much time was left. If it is important that the wait state is not terminated before the interval has expired, the application program should control the length of the expired interval by examining the clock before and after the hold routine is entered. A new hold can then be given if the interval has not expired.

4.4 SEMAPHORES AND INTERNAL DEVICES

TPS application programs may use semaphores and internal devices in the same way as other RT programs. A semaphore is a binary variable which can have one of two values, reserved or unreserved. It is reserved and released by the RESRV and RELES monitor calls. Semaphores are discussed in chapter 4 of the SINTRAN User's Guide.

Internal devices are used for the exchange of data between independent programs. One of them writes on the device as if it were an external device and the other can then read from the device. Devices are reserved and released (program A reserves the output part of the device, program B the input part) and accessed as normal I/O devices. As described in section 3.4, these devices can be accessed by FORTRAN programs by either standard I/O statements or monitor calls, whereas COBOL programs must always use monitor calls. Internal devices are described in chapter 4 of the SINTRAN User's Guide.

5

CHECKPOINT—RESTART

5.1

PROTECTING THE DATABASE

An online transaction processing system should have adequate facilities for protection of the data base. If a system failure occurs in a *batch* system, a backup copy of the data base can be mounted and the whole job run once more without too much inconvenience or waste of time. If a failure occurs in an *on-line* transaction system, reentering and reprocessing the transactions may be very inconvenient, time-consuming or even impossible.

Facilities should be available to assure the integrity of the data base, to minimise the amount of data lost and to restart the system automatically at a well-defined point.

TPS makes use of the extensive checkpoint/restart facilities of SIBAS. These are largely transparent to both the user and the application program, TPS itself controlling them. *Synchronised* checkpoints of the whole system are taken automatically according to the load on the system. In addition, the application program can take individual *transaction* checkpoints.

If a system failure occurs, the latest transaction checkpoint can be used to restore the system to its status at or near the point of failure (recovery). If the data base is incorrect at this point, synchronised checkpoints can be used to restore the system to a previous state (rollback). (*See Figure 1.7.*) In both cases, those transactions which were active can be restarted automatically at the correct point. Note that applications running on the ND-500 cannot be restarted at a point inside an application, but may for example be restarted at the beginning of the current application.

How often checkpoints are taken and what types of backup/restart facilities are used are system parameters controlled by the user. He must weight the advantages of assuring the protection of data in the data base and fast recovery against the overhead needed to accomplish this.

The facilities in TPS for protecting the data base from system failure can be divided into two types, preventive facilities and restart actions (*See Figure 5.1*).

PREVENTIVE	BACKUP	copy whole data base
	LOGGING DELAYED UPDATING	write updated records on a special update file
	BEFORE—IMAGE LOG	log records to be updated (before they are changed)
	ROUTINE LOG	log calls to SIBAS routines
RESTART	CHECKPOINTS SYNCHRONIZED	save checkpoint information for all transaction tasks and for data base
	TRANSACTION	save checkpoint information for one transaction task
	ROLLBACK	restore data base and transaction tasks to synchro- nised checkpoint
RESTART	RECOVERY	restore data base and transaction tasks to transaction checkpoint
	RESTART STRATEGY	restart transactions according to transaction restart strategy

Figure 5.1: Checkpoint/Restart Facilities

5.2 PREVENTIVE FACILITIES

The preventive facilities of TPS consist of:

- utility routines to take a backup copy of the data base
- facilities in SIBAS for updating on a special update file (delayed updating), logging old versions of data (before-image log), and logging SIBAS routine calls (routine log)
- routines for taking checkpoints either automatically, by operator command or by an application program

Most of these facilities are controlled by the TPS system itself or by the system operator. The only facilities controlled to some extent by the application program are checkpoints.

5.2.1 Backup

A backup copy of the data base is a complete copy of the files in the data base. This can be done with a COPY—FILE command while SINTRAN is running, and even while TPS is running if the data base itself is not being updated (for example if delayed updating is used).

It can also be done using a stand-alone disk utility program. If a stand-alone program is used, TPS should be closed (CLOSE-TPS) before SINTRAN is stopped and the copy program loaded. After this type of copy is made, TPS will normally be started again with INITIATE-TPS (See Figure 5.2).

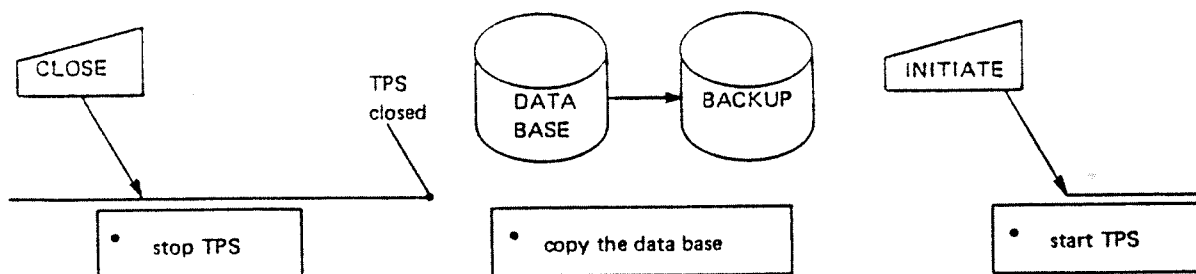


Figure 5.2: Taking Backup with Utility Program

5.2.2 Data Base Logging

SIBAS has three logging facilities available (*See Figure 5.3*):

- Delayed updating consists of the writing of all updated records on a special update file instead of on the data base itself. The data base is only read, not written on. This decreases considerably the chance that it will be physically destroyed, it prevents bad data from being written before the actual updating takes place, and it gives a simple checkpoint/rollback mechanism.
- Before-image (BIM) logging can be used instead of delayed updating. With this method, a page is logged on the before-image area in the SIBAS system realm before it is updated. The data base can then be rolled back by copying all before-images out to the data base.
- The routine log is a sequential disk file containing a record of all calls to SIBAS, in the order they were originally received. This file may be used to update the data base from a backup copy or from a checkpoint without having to rerun the application programs.

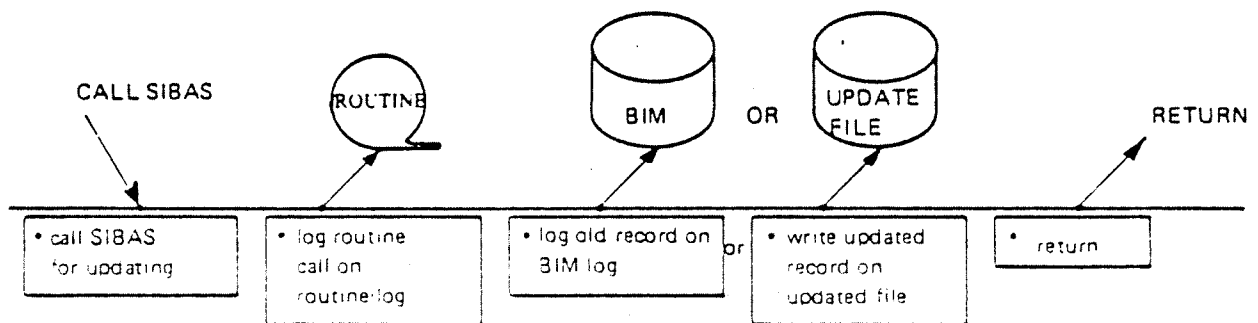


Figure 5.3: SIBAS Logging Facilities

5.2.3 Synchronised Checkpoints

Synchronised checkpoints are checkpoints taken by all TPS and application programs at the same time. When a synchronised checkpoint is taken, the individual transactions will be halted when they reach a suitable point (normally in a TSR). When all transactions have stopped, data needed to restore both application programs and system modules to the present state are recorded on the checkpoint files* (See Figure 5.4). The special application CHECKPOINT is then activated to send a checkpoint call to SIBAS. SIBAS will empty buffer areas, save necessary data, and, if version updating is used, save the *version table* containing pointers to the latest version of all updated records. When SIBAS has completed taking its checkpoint, processing will be restarted automatically. The whole sequence should not take more than 10-20 seconds.

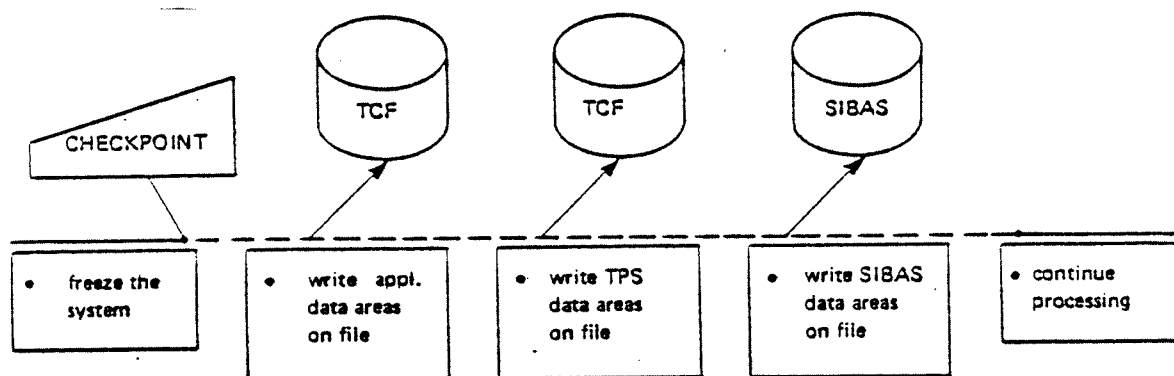


Figure 5.4: Taking a Synchronised Checkpoint

* Note: It is not possible to restore an application that runs on the ND-500.

5.2.3.1 TTSYN - The Allow-Synchronised Checkpoint TSR

It is possible that certain types of transactions may cause a checkpoint to take much longer than 10-20 seconds if the programmer does not prevent it. As mentioned, the system is synchronised by halting transactions in TSRs. This means that a transaction that does a lot of processing without calling a TSR could go a long time before being stopped, especially since SIBAS calls do not go through TSRs that allow synchronised checkpoints (all other I/O calls do). To prevent this situation, an application program with long processing and data base sequences without other TSR calls than SIBAS calls can use a special TSR, TTSYN, that does nothing but allow a checkpoint to be taken if a checkpoint message has come.

```
CALL TTSYN
CALL 'TTSYN'.
```

In TPS systems with automatic synchronised checkpoints, application programs with long processing and data base sequences without other I/O or TSRs should call TTSYN fairly often.

5.2.3.2 THSYN - The Hold Synchronised Checkpoint TSR

In some situations it may be desirable to carry out a processing sequence without allowing a checkpoint to be taken in the middle of the sequence. An example would be the updating of several related records in the data base. If this sequence includes the use of calls to other parts of the TPS system than SIBAS alone, the sequence may be interrupted by a checkpoint. This can be prevented by using the TTSYN TSR.

```
CALL THSYN
CALL 'THSYN'.
```

This will prevent the application from being trapped for a checkpoint until the application makes a call to either the TTSYN TSR or the TTRAN TSR.

All other applications, however, *will* be trapped, so the whole TPS system will eventually stop and wait for this application to complete its sequence. This TSR should therefore be used with care.

5.2.3.3 TCHCK - The Take Synchronised Checkpoint TSR

Synchronised checkpoints can be taken automatically by TPS when the load on the system has reached a certain value. Each application has a *checkpoint weight*, given when the application is loaded into the TPS system and stored in the application table. Every time an application terminates or switches to a new application, its checkpoint weight is accumulated. When the sum has reached a certain value, a checkpoint will be taken automatically.

The operator can also take a checkpoint with the CHECKPOINT TPS command.

An application program is also able to initiate a synchronised checkpoint sequence. This facility should be used with care since taking a synchronised checkpoint involves a good deal of overhead and may halt the system for many seconds. Normally it is best to let TPS take automatic checkpoints based on the activity in the entire system, but some application systems may be such that it is more suitable, for instance, to take a synchronised checkpoint every time a certain application program is run. This can be done from an application program by using the TCHCK TSR.

```
CALL TCHCK ( <scope> )
CALL 'TCHCK' USING <scope>.
```

Examples

```
CALL TCHCK(0)

CALL 'TCHCK' USING ZERO.
```

5.2.4 Transaction Checkpoints

In addition to synchronised checkpoints taken by all programs in the TPS system, application programs take individual *transaction checkpoints* at certain stages in transaction processing. As for synchronised checkpoints, a transaction checkpoint will save all data areas belonging to the transaction, enabling the transaction to be restarted at this point if a recovery is made.* The information is written on a special area on the checkpoint file belonging to the TPT and the transaction checkpoint data will overwrite the previous transaction checkpoint data for that TPT. (See Figure 5.5). The *synchronised* checkpoint data will not be affected however.

Some transaction checkpoints are taken automatically by TPS and the application programmer does not have to take any action in connection with them.

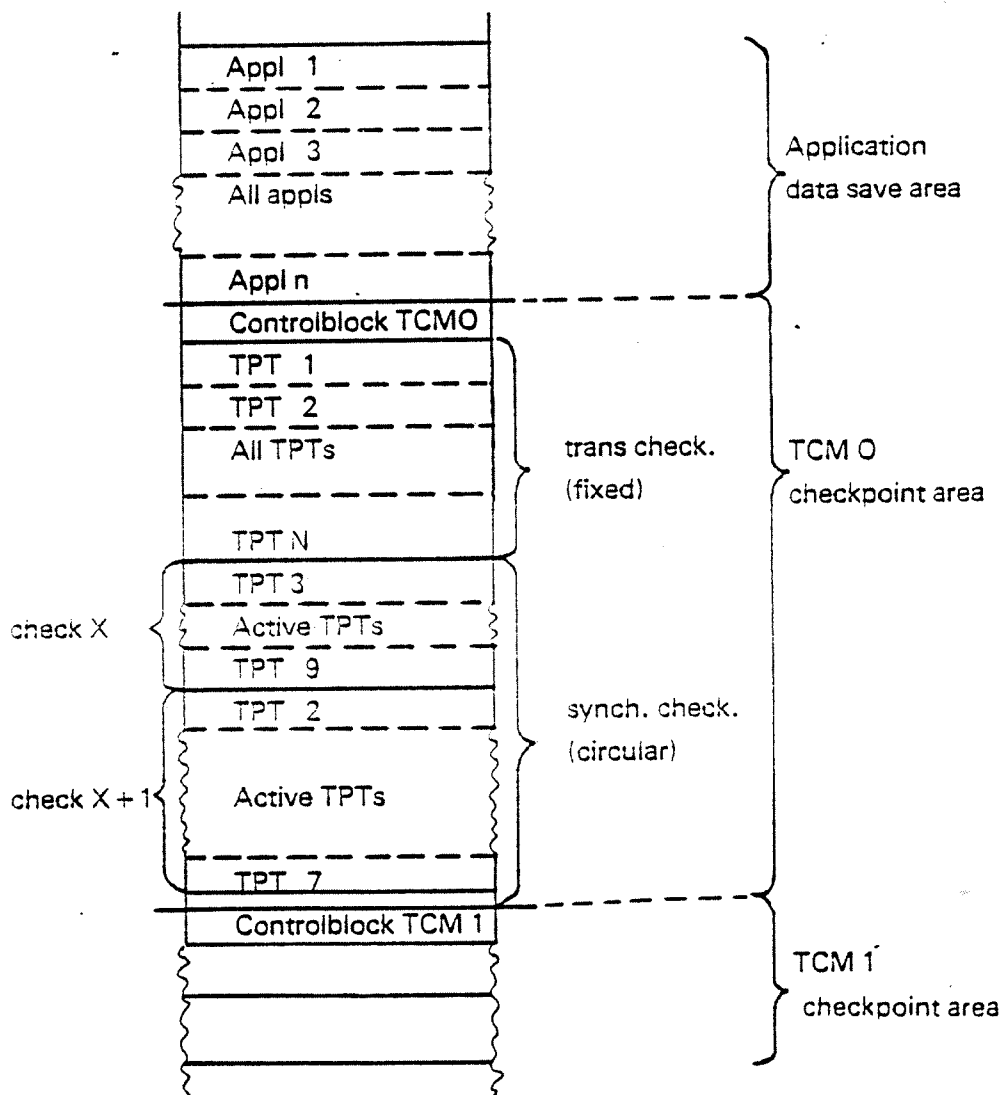


Figure 5.5: The TPS Checkpoint File

* Note: It is not possible to restore an application that runs on ND-500. However, such an application may be restarted from the beginning.

5.2.4.1 TTRAN - The Take Transaction Checkpoint TSR

The application program itself also can take transaction checkpoints, using the TTRAN TSR.

```
CALL TTRAN
CALL 'TTRAN'.
```

In order to decide when and if to take transaction checkpoints, the application programmer must know how they are used. This is explained in connection with the recovery procedure in section 5.3.1, but in short it may be said that the latest *transaction* checkpoint will be the point of restart for an application after *recovery* has been carried out. In contrast, the point of restart after a *rollback* operation will be a *synchronised* checkpoint.

Transaction checkpoints are taken at the following processing stages (*See Figure 5.6*).

- at the beginning of SIGNON (standard version)
- at the beginning of SELECT (standard version)
- when the data base is opened (SOPDB call)
- when the data base is closed (SCLDB call)
- when a task is terminated and the TPT is freed (unless TTERM (1) is used)
- when an application program calls TTRAN

When deciding how often to take checkpoints, it must be remembered that a transaction is restarted at the latest checkpoint after rollback or recovery and any processing done after this point will have to be repeated. Of course rollback or recovery are usually only done in the case of system failure and this should not happen often. As in the case of other preventive facilities, the advantages of taking checkpoints often must be weighed against the costs.

Applications running in the ND-500 may have up to 134 megabytes of local data, and this data will not be saved when TTRAN is called. Only the data in task-common and the TPT system data is saved. This implies that an application running on the ND-500 cannot be restarted at a point *inside* the application, but may be restarted for example at the beginning of the application.

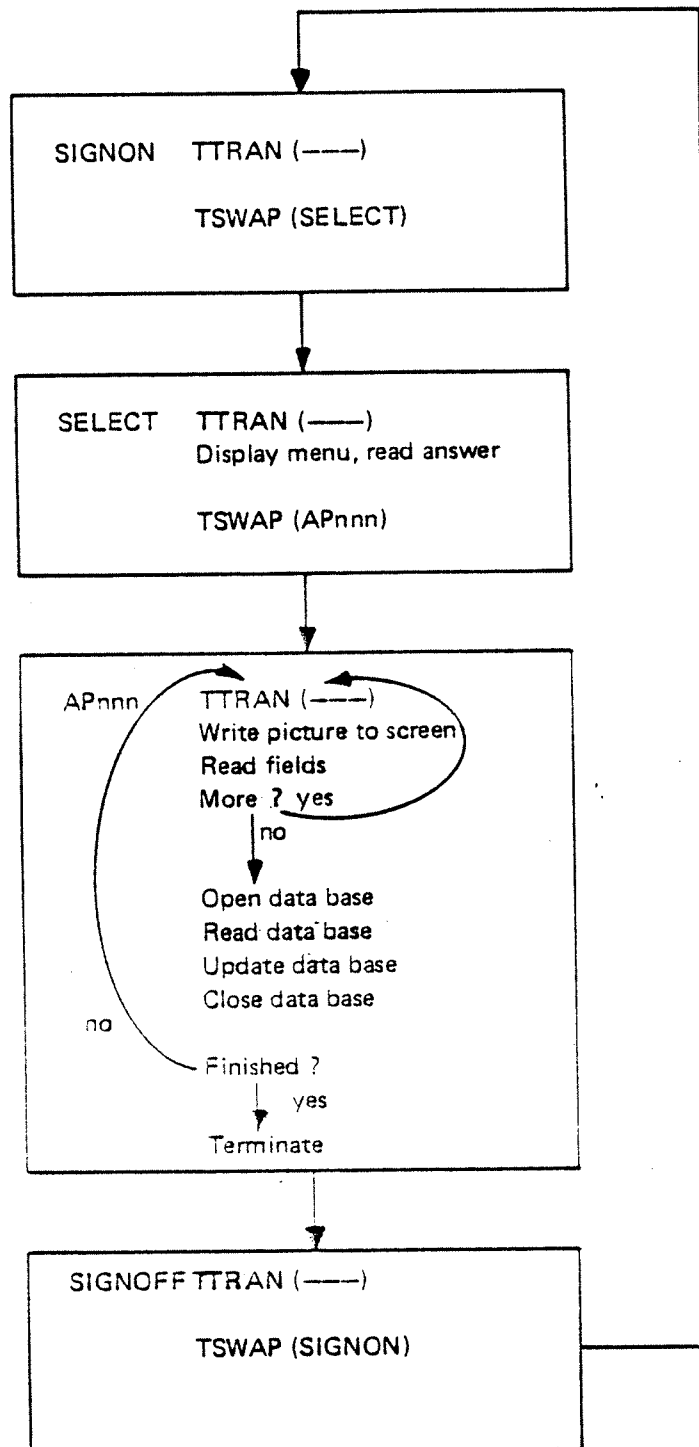


Figure 5.6: Transaction Checkpoints

5.3 RESTART FACILITIES

When a serious error is detected, three steps must be taken:

- the cause of the error should be determined and the error corrected
- any damage to the data base should be repaired
- the system should be started again with as little inconvenience to the users as possible

5.3.1 Rollback and Recovery

Two procedures are available for repairing the data base after a serious error, rollback and recovery.

Rollback will restore the whole system to its state at the last synchronised checkpoint. The ROLLBACK special application will supervise the SIBAS roll-back procedure, causing it to execute the rollback in one of two ways (*See Figure 5.7A*):

- using the update file or before-image (BIM) log to roll the data base back to the checkpoint from the point-of-failure
- using the routine log to update the data base to the checkpoint from the backup copy

In addition, all programs will restore their data areas at the checkpoint so that their state corresponds to that of the data base. The point-of-restart after a rollback will be at the latest valid synchronised checkpoint (unless a recovery is given immediately afterwards).

Recovery, under the supervision of the RECOVER special application, will restore the data base to its state at the latest transaction checkpoints for the various transactions (*See Figure 1.7B*). A rollback to the last synchronised checkpoint will first be performed, using one of the methods described above. The routine log will then be used to update the data base to the individual transaction checkpoint (*See Figure 5.7B*). Programs will then restore themselves to their transaction checkpoints.

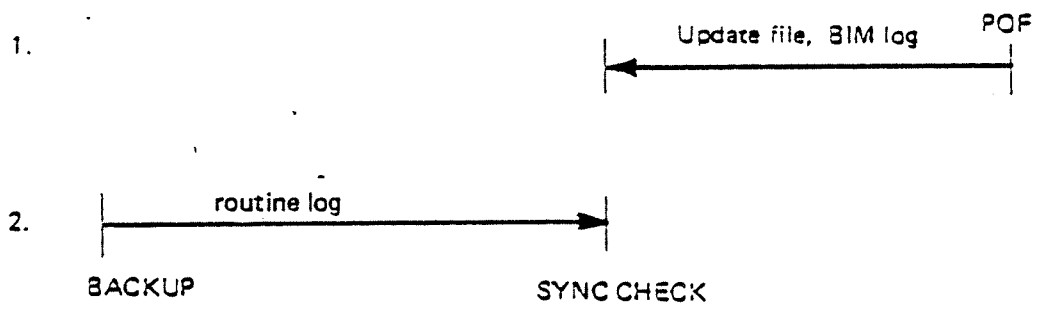


Figure 5.7A: Rollback

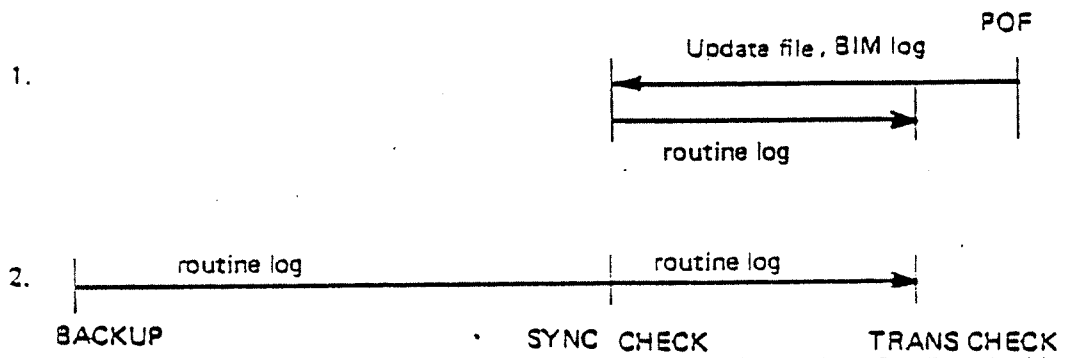


Figure 5.7B: Recovery

The point-of-restart after recovery will thus be the latest transaction checkpoint for each active transaction (See Figure 5.8). There is one exception to this rule. If no transaction checkpoint has been taken since the last synchronised check-point, the transaction will be restored to the synchronised checkpoint. Point-of-restart is summarised in figure 5.9.

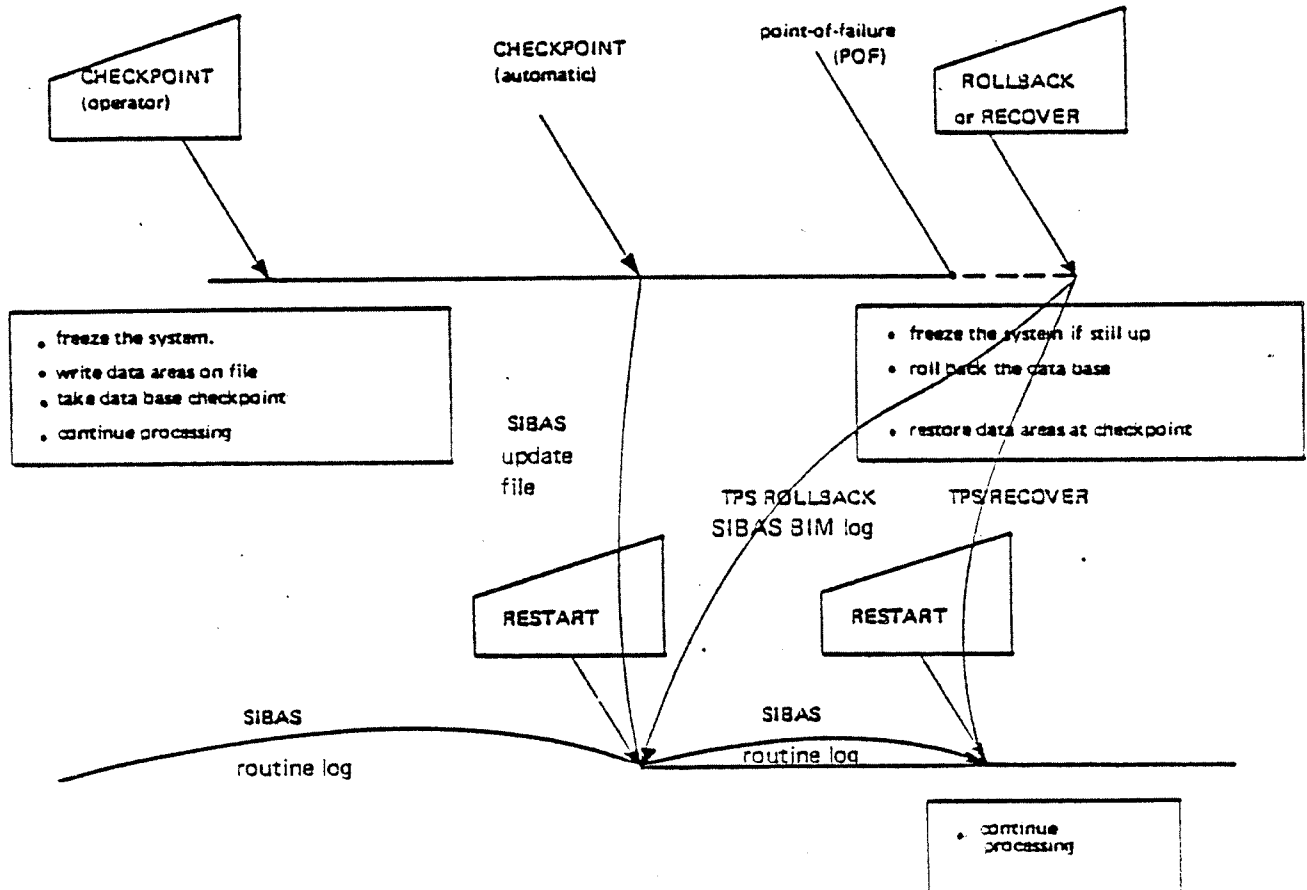


Figure 5.8: Rollback and Recovery

5.3.2 Restarting TPS

After rollback or recovery, the TPS system may be restarted either automatically or using the CONTINUE command. Transaction processing will continue as far as possible as though there had been no break.

To what extent this can be done depends mainly on the state of the connections between the application programs and the external environment. Both the data base and the application programs* have been restored to their state at the appropriate checkpoint and are ready to continue processing from there. However, terminals and local devices may have been lost and sessions may have been broken, depending on such things as whether SINTRAN, NSHS or I/O modules have been reloaded, files have been closed, terminal operators have broken connections, etc. After rollback, there is the additional problem that external connections at the synchronised checkpoint were probably different from those at the point of failure.

Another important consideration at restart is that some transactions may not want to be restarted at the checkpoint. There are several ways of restarting transactions and each must be restarted according to the appropriate restart strategy. This is done by the RESTART special application.

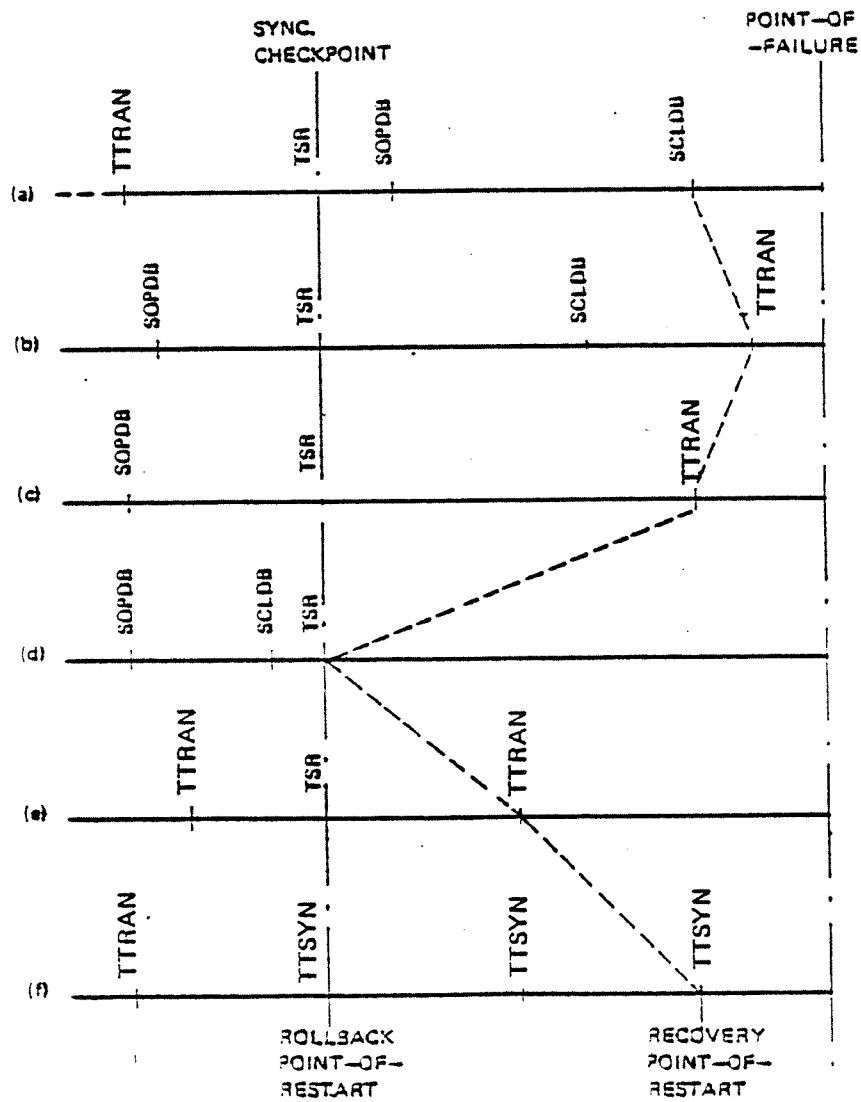


Figure 5.9: Point of Restart

* Note: It is not possible to restart an application in the middle that runs on the ND-500, but the application may be restarted from the *beginning*. You have to remember this when setting the restart strategy.

5.3.2.1 The RESTART Special Application

The RESTART application is the first application to be called when the system is restarted after rollback or recovery. It is called for each active transaction by the TPT for that transaction.

When RESTART is activated, the data base will have been reopened by SIBAS for that transaction if it was open at checkpoint. Terminals controlled by NSHS will be automatically reacquired by NSHS the first time they are used. On the other hand, the standard version of the RESTART program will not open files or reserve local devices. Likewise, broken sessions may not be reestablished. However, users can modify RESTART to do these things. The (TSEST) and restore-session (TRSES) TSRs are available to assist users in this. Modifying the RESTART and other special application programs is discussed in the TPS System Supervisor's Guide.

When modifying this application, remember that applications running in the ND-500 cannot be restarted at a point inside the application (the TSR-call TRSTO is not allowed).

The task of the RESTART special application is to determine how the transaction is to be restarted, to restore or break external connections, and restart (or terminate) the appropriate application program. RESTART will have available the necessary information to do one or more of the following:

- restart the transaction or terminate it
- reestablish the connection with the terminal operator. If no answer is received (timeout), the ABEND application is activated
- ask the operator to choose which restart strategy is to be used
- give the operator information to enable him to resume operation at the correct point.

5.3.2.2 TSRST - The Set Restart Strategy TSR

RESTART uses the restart strategy for the task to determine if and how to restart a transaction. The restart strategy is given the default value 2 when the task is started and the TSRST TSR can be used to change it to another value, the meaning of each value depending on the way it is interpreted by RESTART.

```
CALL TSRST ( <restart strategy>, <restart application> )
CALL 'TSRST' USING <restart strategy> <restart application>.
```

In addition to setting the restart strategy, the TSRST TSR can be used to change the restart application. The default value for this is the first application activated. The standard restart strategies are described under the RESTART application in chapter 6.

Examples

```
CALL TSRST(1)

CALL 'TSRST' USING TWO RE-APPL.
```

In summary, TPS is designed so that users can exert full control over both checkpoints and restart with the TTRAN, TTSYN and TSRST calls. They can write their own restart routines and modify the RESTART special application. They can decide how often to take synchronised checkpoints and what type of logging, rollback and recovery to use.

However, they can usually ignore all these things, using only the standard defaults supplied with TPS and still have a system that functions well.

SPECIAL APPLICATIONS

NORD TPS is delivered with a set of standard "special applications". These programs are used to carry out such user dependent functions as the administration of TPS users, authorising and limiting user access to application systems and individual programs, automatic administration of menu pictures and security control. These functions are carried out by the SIGNON and SELECT applications. The SIGNOFF, ABEND and RESTART applications control transaction administration at transaction termination (with the possibility of gathering transaction statistics), abnormal end and restart after system failure. In addition, the system special applications, TPOPEN, TPCLOSE, CHECKPOINT, ROLLBACK and RECOVERY, are used to administer system functions, especially SIBAS control. With the exception of TPOPEN, they are of less interest to the application programmer and are therefore not discussed in detail in this manual. See TPS System Supervisor's Guide for more information about them.

The special applications are written in FORTRAN and/or COBOL (SIGNON and SELECT can be obtained in both languages). Some or all can run in either the ND100 or the ND500 CPU. (RESTART cannot run in ND500 if it calls the TSR-routine TRSTO.) Emphasis has been put on comments and an easy to follow structure so that users can easily change the programs to suit their own needs. In addition, the configuration and security routines are controlled by user-defined tables and make extensive use of default values to simplify table definition in those cases where it is not necessary to limit access to the system.

The relation between the special applications and user applications is illustrated in figure 6.1. This figure shows the *standard* use of these applications. They may be changed by the user so that the illustration no longer applies. Also, the standard version of SIGNOFF, ABEND and RESTART have different processing sequences for the different strategies they use (termination strategy, abend strategy, restart strategy) and do not necessarily follow the illustrated sequence (See Section 6.2.4).

The special applications are described below in functionally related groups as follows:

- SIGNON and SELECT
- SIGNOFF, ABEND and RESTART
- TPOPEN and TPCLOSE
- CHECKPOINT, ROLLBACK and RECOVER

Again it must be emphasized that the following descriptions are only valid for the *standard* versions of the special applications.

In addition to the above mentioned user modifiable special applications, there is a special non-modifiable application, TPMON, which monitors applications running on the ND-500. Further description can be found in Appendix J.

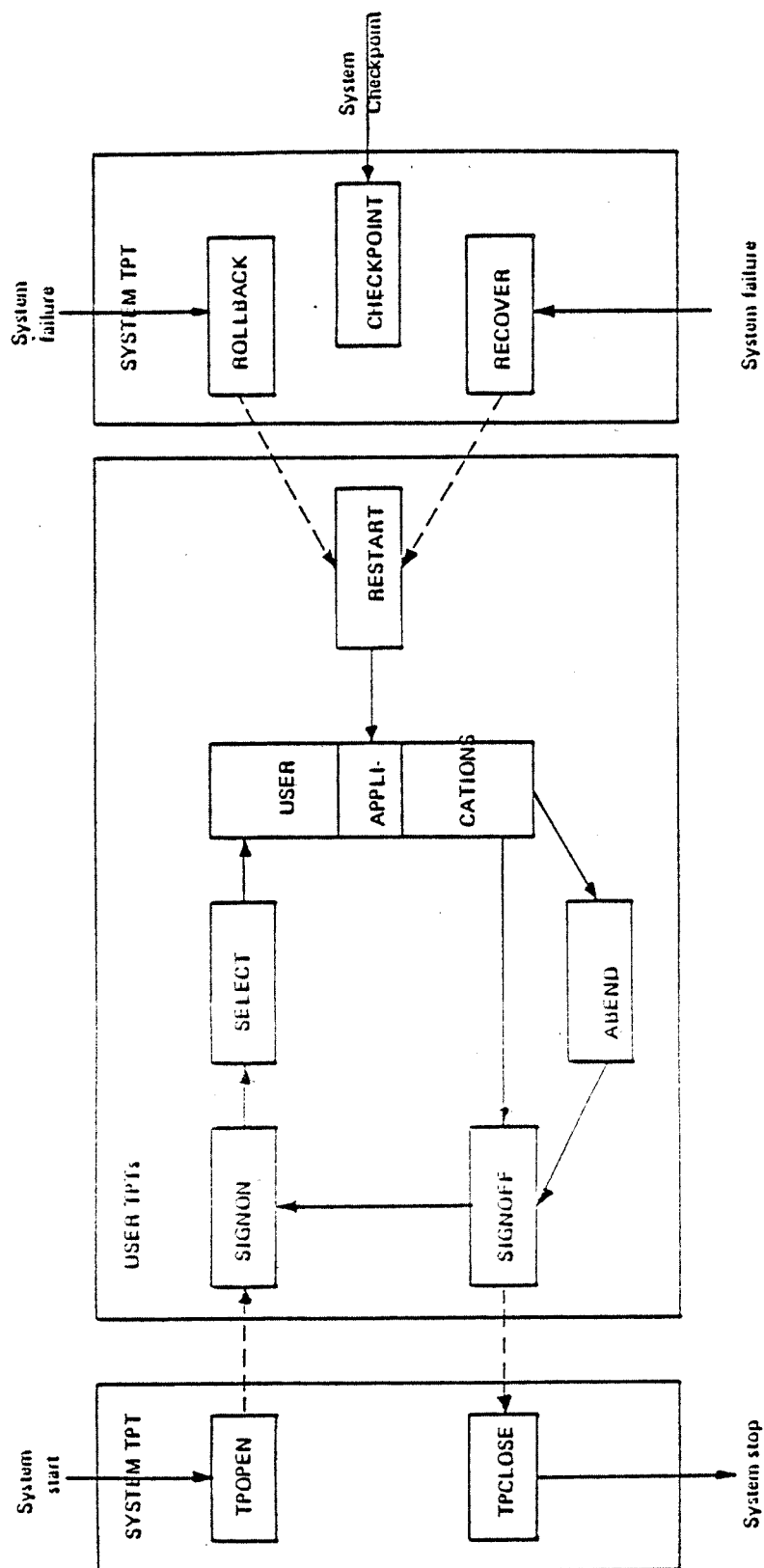


Figure 6.1: Special Applications

6.1 SIGNON AND SELECT

6.1.1 SIGNON

The main function of SIGNON is to check the terminal user's name and password. It is normally the first application to be called when a task is started. The program will reserve the terminal the first time it is called and will then write the SIGNON picture on the terminal. The cursor will be positioned at the input field for the user name.

When a name is entered, SIGNON will look for the name in the user table, TPS-USERTAB. The name may be abbreviated in the usual SINTRAN manner. If the name is found, the next question is for the password.

The password is then checked, and it is either accepted or the cursor moved back to the name field. Figure 6.2 shows a sample SIGNON picture.

If name and password are accepted, SIGNON will use the user's EXIT TYPE in TPS-USERTAB to carry out one of the following actions:

- Switch to SELECT to present the user's master menu
- Switch directly to a user application
- Exit from TPS (release the terminal for SINTRAN background use). This action is usually defined for a special user, for example the user SINTRAN. This action is carried out by releasing the terminal and going into a wait loop. Every 2 seconds SIGNON will try to reserve the terminal again, so that when the SINTRAN background user is done (logs off), the terminal will again be brought into the TPS system.

ND TPS ON LINE AT 15.45 ON MARCH 1, 1982

TTTTTTT TTTTTTT TT TT TT TT TT TT TT	PPPPPPP PPPPPPP PP PP PPPPPPP PPPPPPP PP PP PP PP	SSSSSSS SSSSSSS SS SSSSSSS SSSSSSS SS SS SSSSSSS SSSSSSS
--	---	--

PLEASE ENTER YOUR NAME:

PASSWORD:.....

Figure 6.2: A SIGNON Picture

6.1.2 SELECT

The main function of SELECT is to control the menu choices for starting user transactions. When SELECT is started, it will usually send the user's master menu given in the user's entry in TPS-USERTAB. However, if SELECT was activated directly from a user application instead of from SIGNON, the menu presented will be the last master-type menu used before the application was started.

The menu picture will normally show several numbered entries and the user is asked to choose one of them (*See Figure 6.3*).

An entry, when chosen, is handled in one of four ways:

- Switch to a user application
- Present a new menu
- Present the user's master menu
- Log off as present user and switch to SIGNON

The way in which each entry is handled and additional information such as application number (if the first way) and menu number (if the second way) is given in the menu table, TPS-MENUTAB. Each menu is described here, giving the different entries and the type of handling for each entry.

These possibilities provide a large amount of freedom in defining menus. Figure 6.3A shows a sample master menu with several sub-menu entries plus log off ('STOP'). Figure 6.3B shows one of the sub-menus with entries for application programs, new sub-menus, the master menu or logging off.

ND TPS	MASTER MENU
1	ACCOUNTING
2	PAYROLL
3	INVOICE
4	INVENTORY
5	TEXT PRO CESSING
6	STOP
ENTRY CHOICE:	

Figure 6.3A: A Master Menu

ND TPS	ACCOUNTING
1	BOOKKEEPING
2	ACCOUNTS RECEIVABLE
3	GENERAL LEDGER
4	REGISTER UPDATE
5	REPORTS
6	MASTER MENU
7	STOP
ENTRY CHOICE:	

Figure 6.3B: A Sub-Menu

6.1.3 The Access Control System

Access control in the standard versions of SIGNON and SELECT is done in three ways:

- A user may have a password which must be given when the user enters his name - this controls total access to TPS
- A user has only access to his master menu and the sub-menus which can be chosen through the master menu - this controls menu access
- Every menu entry can have a security code and only those users with a code greater than or equal to the menu entry code may choose that entry - this controls menu entry access

SIGNON and SELECT use three tables to control user access to TPS:

- the user table, TPS-USERTAB
- the menu table, TPS-MENUTAB
- the default table, TPS-DEFAULT

Defining these tables is usually a task for the TPS system supervisor and is described in the TPS System Supervisor's Guide. In addition the system supervisor should use NSHS or FOCUS to define:

- the SIGNON picture for control of user name and password
- pictures for the various menus defined in the menu table

The access control system in the standard versions of SIGNON and SELECT is designed to provide a large amount of freedom in defining the control for a particular TPS system. The amount of information contained in the tables will depend on the degree of control needed, from the simple use of defaults to the detailed use of passwords, restricted menu choices and security codes.

Access to different subsystems may also vary greatly. Access to an invoicing system, for example, may be quite general, while the payroll system may be more restricted.

6.2 SIGNOFF, ABEND AND RESTART

6.2.1 SIGNOFF

The SIGNOFF application is given control when a transaction terminates normally, i.e. when one of the following occurs:

- reaching the logical end of the program (the END or STOP RUN statement)
- using the TSTOP TSR with a stop code of 0
- the LEAVE monitor call

The function of SIGNOFF is to terminate the transaction in the way indicated by the termination strategy for that task. The termination strategies and the actions taken by the standard version of SIGNOFF are:

- 1 — Terminate the task completely (release the terminal if it has one, break a session if there is one, take a transaction checkpoint and free the TPT)
- 2 — Switch to the SIGNON application, using TSWAP (no devices or other resources freed)
- 3 — Switch to the SELECT application
- 4 — Switch to the user-defined termination application

The termination strategy and the user termination application are obtained by SIGNOFF with a special TSR, TSTAT (read the status of the current task).

When a task is originally started, the termination strategy is set to 1, complete termination. The TSTST TSR (set termination strategy) can be used to change it and to define a user termination application. The standard version of SIGNON changes the strategy to 2, switch to SIGNON; the standard version of SELECT does not change it.

If a user termination application is used, it must not itself terminate "normally" unless it has changed the termination strategy, since this would result in an endless loop. It may, for example, terminate by switching to SIGNON.

6.2.2 ABEND

The ABEND application is given control when a transaction terminates abnormally, i.e. when one of the following occurs:

- a serious error is detected by the FORTRAN, PLANC or COBOL runtime system
- a serious error is detected by the TPS system
- the TPS operator terminates the transaction
- the application program uses the TSTOP TSR with a non-zero stop code

A serious error is any error which prevents the program from continuing, such as timeout, an I/O error without an error handling routine, switching to an application program that has not been loaded, a 'fatal formatting system error', etc.

When ABEND is activated, it will start by sending the "abend error message" to the TPS operator console (see below for the format of this message). It will then carry out the action indicated by the abend strategy for the task and it will finish by switching to SIGNOFF to terminate the transaction. The abend strategies and the corresponding actions taken by the standard version of ABEND are:

- 1 — No more action - just switch to SIGNOFF
- 2 — Send the abend error message to the terminal operator (if the transaction has a terminal), switch to SIGNOFF
- 3 — Dump the data areas for the TPT on the line printer, switch to SIGNOFF
- 4 — Switch to the user abend application
- 5 — Halt TPS

The abend strategy and the user abend application are obtained by ABEND using a special TSR, TABST (read the abend status of the current task).

When a task is originally started, the abend strategy is set to 1, send the abend error message to the TPS operator console and switch to SIGNOFF. The TSAST TSR (set abend strategy) can be used to change it and to define a user abend application.

It is important that the user abend application is thoroughly tested before being used, since an abend in that application would probably result in an endless loop. It should terminate in the normal way (END, STOP RUN, TSTOP(0)) so that SIGNOFF will be activated when it is done.

6.2.2.1 The Abend Error Message

The error message sent by the standard ABEND application is as follows:

```

APPL. NO. aaa  ABENDED BY {TPS
                           {RUNTIME SYSTEM
                           {APPLICATION
                           }
                           }

IN ADDR yyy    TPT NO. tt

DUE TO {reason (text) if abended by TPS
        {reason (code) if abended by appl. or RUNTIME SYSTEM
        }

DATA BASE ACTIVITY: {CLOSED
                     {READ
                     {UPDATE
                     }
  
```

Codes:

aaa TPS application no. (0-255)

tt TPT no. 1-63

yyy Latest link register

reason (text)
or reason (code)

If abended by TPS, one of the following texts:

- 0 = Abended by operator
- 1 = Application cannot be activated
- 2 = Illegal use of TSRs
- 3 = Subroutine not loaded
- 4 = Application Timeout
- 5 = Internal TPS error
- 6 = Operator Timeout
- 7 = Attempt to restore ND-500 application
- 8 = Error from ND-500 monitor

If abended by application:

Stop code given in TSTOP or error message from NSHS or SIBAS

If abended by runtime system

SINTRAN error code

6.2.3 RESTART

The RESTART application is given control when a transaction is to be restarted after a system rollback or recovery operation. It is started for each TPT and has the function of examining the restart strategy for the task and carrying out the appropriate restart action.

The restart strategies and the corresponding actions taken by the standard version of RESTART can be divided into 2 types, terminal operator controlled and automatic. Transactions involving interaction with a terminal should normally use the terminal controlled restart strategy, since the operator will be better informed of the situation and have control of it to some extent.

The standard restart strategies (*See Figure 6.4*) are:

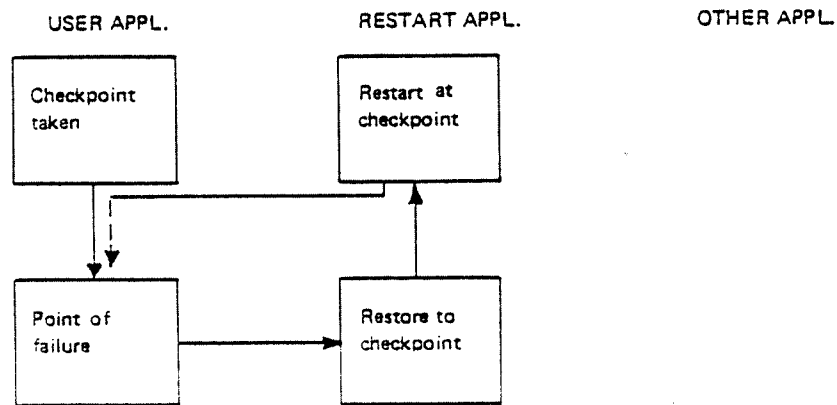
- 1 — *Automatic Restart at Checkpoint.* Go directly back to the application program active at checkpoint and continue processing at the next statement after the checkpoint was taken. This strategy cannot be used for transactions running on the ND-500. A special TSR, TRSTO (restore application status and restart), is used by the RESTART application to do this. It is described in the TPS System Supervisor's Guide.
- 2 — *Start the Application in a User-Specified Restart Application.* The RESTART program will not return to the active application program but switch to the user restart application program set by the TSRST TSR. This program may be SIGNON, SELECT, the active application program (which will then be restarted from the beginning) or any other user application program. For example, if several application programs are run sequentially (using TSWAP) they can be restarted from the beginning of the first one or any of the others, or a special user-restart application may be started. User restart application will have access to the common data area of the transaction and the data will have the values they had at checkpoint.
- 3 — *Terminate the Transaction.* RESTART will switch to SIGNOFF.
- 4 — *Terminal Operator Controlled Restart.* This strategy can only be used by application programs with terminals controlled by NSHS/FOCUS. RESTART acquires the terminal and sends a message informing the operator of the restart condition and asking him to choose the restart action which suits him best (*See Figure 6.5*). As the figure shows, the operator can choose between:
 - a — Restarting at checkpoint (not allowed for ND-500)
 - b — Terminating the transaction
 - c — Switching to SELECT to choose from the user's master menu

The operator controlled strategy includes a timeout, and if no answer is received before the timeout expires, the ABEND application is started.

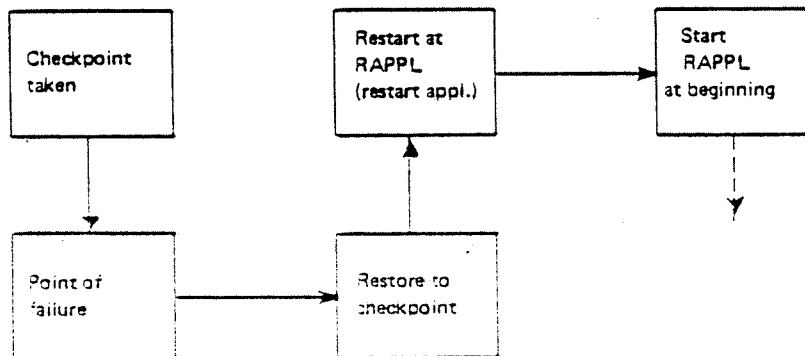
The default value of the restart strategy when a task is started is 2, start the application in user restart application, and user restart application is the first application that has been started (normally SIGNON). The restart strategy and the restart application can be changed with the TSRST TSR.

The value of the restart strategy and the user restart application are obtained by RESTART using a special TSR, TRRST (read the restart status of the current task).

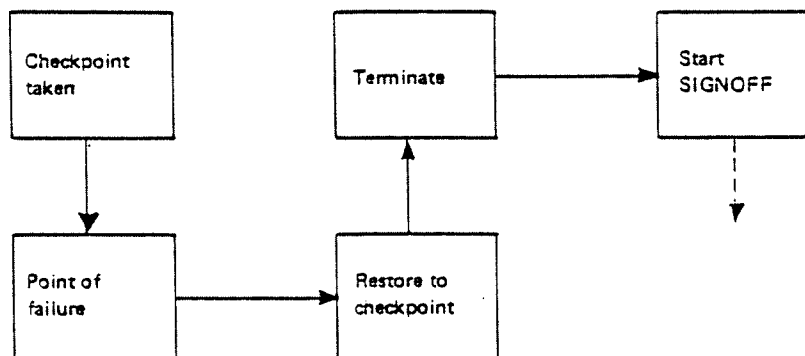
When a task is originally started the restart strategy is set to 2, switch to the user restart application, and the user restart application is the first application that is activated. This will normally be SIGNON. The TSRST TSR (set restart strategy) can be used to change the strategy and the restart application.



RESTART AT CHECKPOINT (NOT IN ND500)



RESTART AT RAPPL



TERMINATE

Figure 6.4: Standard Restart Strategies

WE'RE SORRY

THAT THE SYSTEM TOOK A BREAK WHEN THIS TERMINAL WAS OPERATED
BY user name

ANYHOW, IT IS NOW RESTARTED AT THE STATE FOUND AT

PLEASE SELECT ONE OF THE FOLLOWING ACTIONS:

1. CONTINUE AT CHECKPOINT (PRESS CONTROL QQQ TO RESTORE PICTURE)
2. TERMINATE THE TRANSACTION
3. RETURN TO THE MASTER MENU

Figure 6.5: Terminal Operator Controlled Restart

6.2.4 Summary of Termination, Abend and Restart Strategies

Figure 6.6 shows the relationship between the various special applications, user applications and the strategies employed by SIGNOFF, ABEND and RESTART. The numbered paths on the figure show the flow of control for the corresponding strategy numbers. The dark paths show the control flow if the standard default strategies are used. The dashed lines show the normal flow of control in the TPS system apart from the strategies described.

Note that the user restart application is placed as a separate application from all of the others. This is not necessarily the case, as it may be any application, either special or user. The default value for it is, in fact, SIGNON (however, drawing a dark path from RESTART to SIGNON to show this default would have made the figure too messy!). It may also be the application that was active at checkpoint, the difference between strategies 1 (restart at checkpoint) and 2 (switch to restart application) being that strategy 1 would start the application in the middle, after checkpoint, while strategy 3 would start it at the beginning.

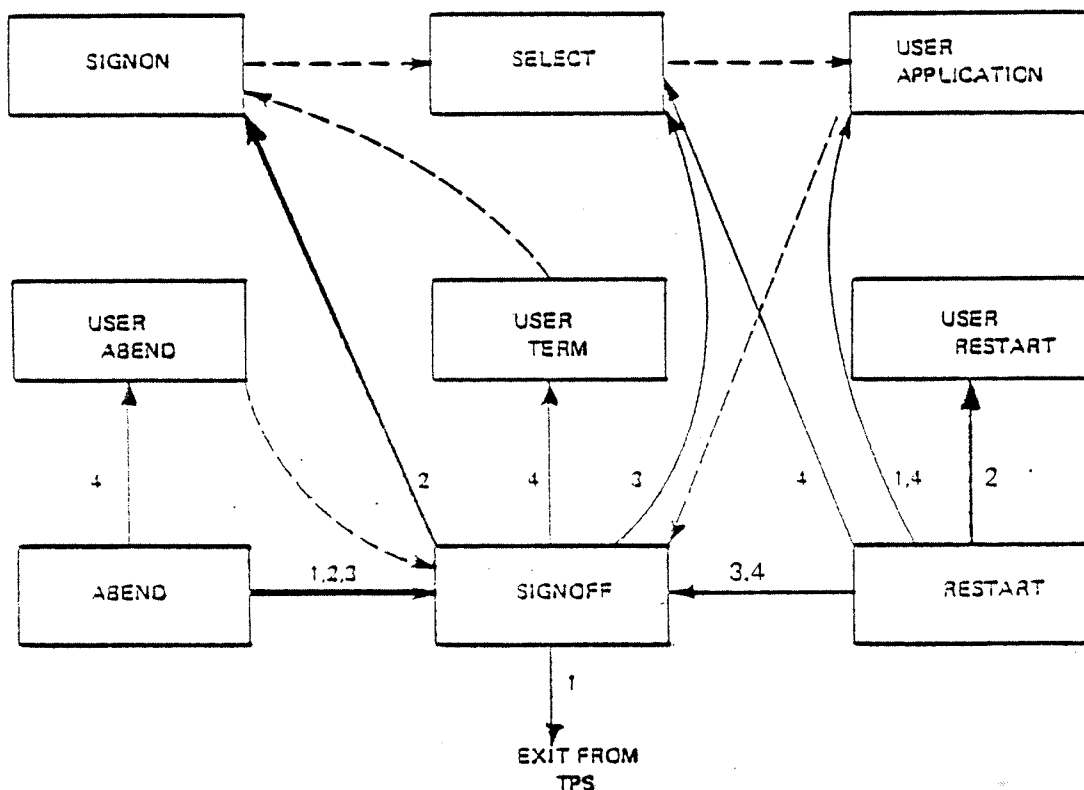


Figure 6.6: The Terminate, Abend and Restart Strategies

6.3 TPOpen AND TPClose

The TPOpen and TPClose special applications are system applications called when TPS is initially started up and when TPS is closed down. A system special application is one that is only called by one TPT for each TCM, the system TPT, and that performs processing that affects the whole TPS system, not only a single task.

TPOpen has several functions. It sends a 'good morning' message to the TPS operator. It opens the data base for a special system user in order to take checkpoints and control rollback and recovery. The efficiency of the TPS system is also increased if the data base is always open for at least one user, since opening it for other users will then go faster (*See Section 3.2.3*). TPOpen will also go through the terminal configuration table, TPS-TERMTAB, and start up a task for each terminal in the table, using the TACTV TSR to acquire a TPT and activate the SIGNON application. TPS-TERMTAB is defined in the same way as TPS-USERTAB, TPS-MENUTAB and TPS-DEFAULT (*See Section 6.1.3*). Defining these tables is described in the TPS System Supervisor's Guide.

TPClose is activated when a CLOSE-TPS command has been given and all transactions have been completed. After CLOSE-TPS has been given, no new transactions may be started. SIGNOFF controls this by checking for a close situation when a transaction terminates and causing complete transaction termination with release of the terminal and the TPT, regardless of the normal termination strategy. When all TPTs have been released, the TPClose application will be activated. It will also be activated if an ABEND-TPS command is given, but in this case the activation is immediate without waiting for transactions to terminate. TPClose will close the data base and send a 'good night' message to the TPS operator.

6.4 CHECKPOINT, ROLLBACK AND RECOVER

The CHECKPOINT, ROLLBACK and RECOVER special applications are also system applications called by the system TPT to perform functions affecting the whole TPS system.

They are activated when the corresponding commands are given either by the operator, a system module (a TCM for example) or an application using an operator-command TSR (these are restricted to special applications). Their task is to supervise the SIBAS actions needed to carry out the required functions, either by calling SIBAS directly (GCHPO, SROLL, SREPR, etc.) or by instructing the TPS operator in carrying out the functions manually. Checkpoint, rollback and recovery are discussed in detail in chapter 5.

7

SPECIAL CONSIDERATIONS

This chapter describes various special considerations which should be taken when writing programs to be run under TPS. It includes data area definition, language dependent limitations and requirements, program structure and efficiency.

7.1

DATA AREAS IN THE ND—100

Application programs must be reentrant. This means that they cannot be written into and thus may not contain any variable data. The data area for an application program is placed instead on the *non-reentrant part of the TPT* for that transaction. This is done automatically by TPS and demands no special action on the part of the application programmer. There is, however, a restriction on the *size* of the data area, discussed in section 7.1.4, and some rules must be followed when defining the data.

Since data areas belong to the TPT and not to the application program itself, no *variable* data may be initialised before execution. This is a general rule for reentrant programs. Note that the data area will not be cleared either but contain arbitrary values.

Constant data, on the other hand, may be initialised in application programs written in FORTRAN/PLANC, NPL and MAC (but not COBOL). FORTRAN programs must define the data as belonging to a COMMON area (not the COMMON/Private/area), initialise the data in a BLOCK DATA subprogram and load the block data subprogram together with the application program. PLANC programs may define the data as *global* read only data in modules. It will then be part of the read-only segment containing the application program and can be read but not changed. This method of initialising the data must be used since the DATA statement is not allowed in reentrant programs. NPL and MAC programs can initialise the data directly in the programs.

Example - Constant data in FORTRAN

MAIN PROGRAM

```
PROGRAM AP050
COMMON/Private/ITERM....      task common data area in TPT
.
.
COMMON/CONST/K1,K2,C3,TABLE(10)....  constant data area in program
.
.
```

CONSTANT DATA

```
BLOCK DATA
COMMON/CONST/K1,K2,C3,TABLE(10)....
DATA K1,K2,C3/1,2,5.642/TABLE/1.2,2.0,10.3,5*50.0,0.0,1.0/....
.
.
```

END

ND-60.111.03

LOADING THE APPLICATION (SEE SECTION 8.2.1)

^ADD-APPL,AP050-BRF,AP050;
^ADD-UNIT,CONST-DATA-BRF;

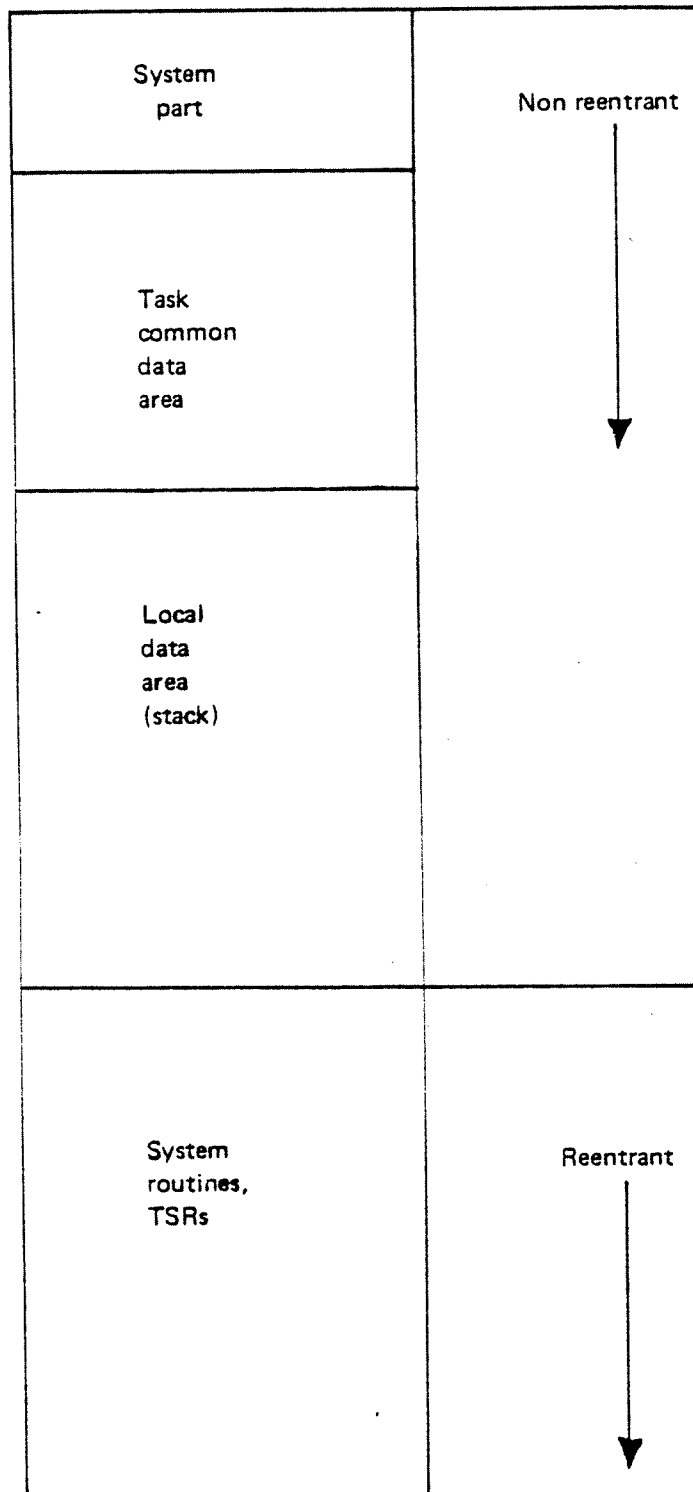


Figure 7.1: The TPT

7.1.1 The Variable Data Area In The ND—100

The variable data area is located in the non-reentrant part of the TPT. It consists of two parts, the task common data area for the transaction and the local areas or stack (*See Figure 7.1*).

7.1.2 The Task Common Data Area In The ND—100

The task common data area contains data that is available to all the application programs and subroutines for a transaction. If one application program switches to another using TSWAP, the new application program have the same task common area, containing the old data.

Note: The first word of task-common should always contain the terminal device number if the application uses a terminal. If not, it ought to equal zero.

If the transaction uses NSHS, the first variables in the task common data area must be the terminal buffer ITERM and the private picture area IPRIV (*see the NORD Screen Handling System*). After that follow common transaction data.

If the transaction uses FOCUS, then the task common area from word 100 upwards will be used by FOCUS. The first 100 16-bit words only are available for common transaction data.

The task common area is defined in a FORTRAN program with the COMMON statement and must have the name PRIVATE.

```
COMMON/PRIVATE/ITERM(128),IPRIV(1024),REST
```

Or in a PLANC-program:

```
IMPORT (COMMON)(type:PRIVATE)
```

This must be the only COMMON statement for *variable* data in the program (constant data may be defined in other COMMON areas as described above).

The total length of the common area PRIVATE is fixed for FORTRAN/PLANC programs at system generation time. The individual transactions may define a common area of any length up to the fixed maximum.

In order to assure that the COMMON statement is correct in every application program for a transaction, the statements can be defined separately and copied to the individual applicaton programs with the INCLUDE statement.

In COBOL main programs, the task common data area is just defined as the first part of working storage. As long as this data is defined in the same way in all application programs for a transaction, they will all have access to it. As in FORTRAN/PLANC, part of the common area must contain screen handling system variables. In contrast to FORTRAN, the total length of the common area is variable (up to a maximum value) for individual transactions, since it is just the first part of working storage defined in the same way for several programs.

WORKING-STORAGE SECTION

```
01  ITEM  COMP  OCCURS  126.  
01  IPRIV COMP  OCCURS  1024.  
01  REST.
```

The COPY statement can be used in COBOL to assure the correct definition of the common area.

The task common data area cannot be accessed from COBOL subroutines via the working storage section. For subroutines the working storage area is allocated at a fixed address of the local data area at load time. However, the task common data area may be accessed via parameters defined in the linkage section.

If application programs are written in several languages and an application program uses the TSWAP TSR to switch to a new application program written in a different language, special care must be taken that the task common data areas are defined in the same way in both programs. Then the new program will have access to the common data of the old program even though they are written in different languages. It is the responsibility of the programmer to see to it that the data definitions match.

7.1.3 The Local Data Area In The ND—100

In addition to data in the task common data area, the individual application programs can define local data. This data will be lost when switching to a new application program and is not available to subroutines.

Local data for all FORTRAN/PLANC programs, both main programs and subroutines, are placed in the stack (*See Figure 7.2.A*). The stack is the data area in the TPT immediately following the common area. When a FORTRAN/PLANC main program or subroutine is started, it is given an area in the stack large enough to contain all local data defined in the routine. It will have access to this area until it is done, when the area will be freed. If the program calls a subroutine, the subroutine will be given an area in the stack following the area for the calling program. The stack can thus be considered a pool of storage space allocated to individual routines dynamically during execution. Since space is allocated dynamically, data in the stack may not be initialised before execution time.

MAC and NPL programs must simulate reentrant FORTRAN/PLANC programs when interacting with the system.

The MAC and NPL programmer must follow rather strict rules in order to use the stack and define data correctly. The FORTRAN programmer, on the other hand, defines data in the usual FORTRAN manner with the exception of the restriction on the use of COMMON described above. The compiler must be set in reentrant mode when compiling the program and the program must be loaded as described in section 8.3.1. If these things are done, addressing and stack administration will be performed correctly.

The data area for a COBOL program is described in figure 7.2B.

In PLANC-programs, you must refer to a stack in the INISTACK statement in order to satisfy the PLANC-compiler. You have to declare a global *dummy-stack* INTEGER ARRAY STCK (0:1) in your main program module and include the statement INISTACK STCK in the main program. The symbol 5STLEN should be left undefined by load-time. The actual stack length will be set up when the TPT is initiated.

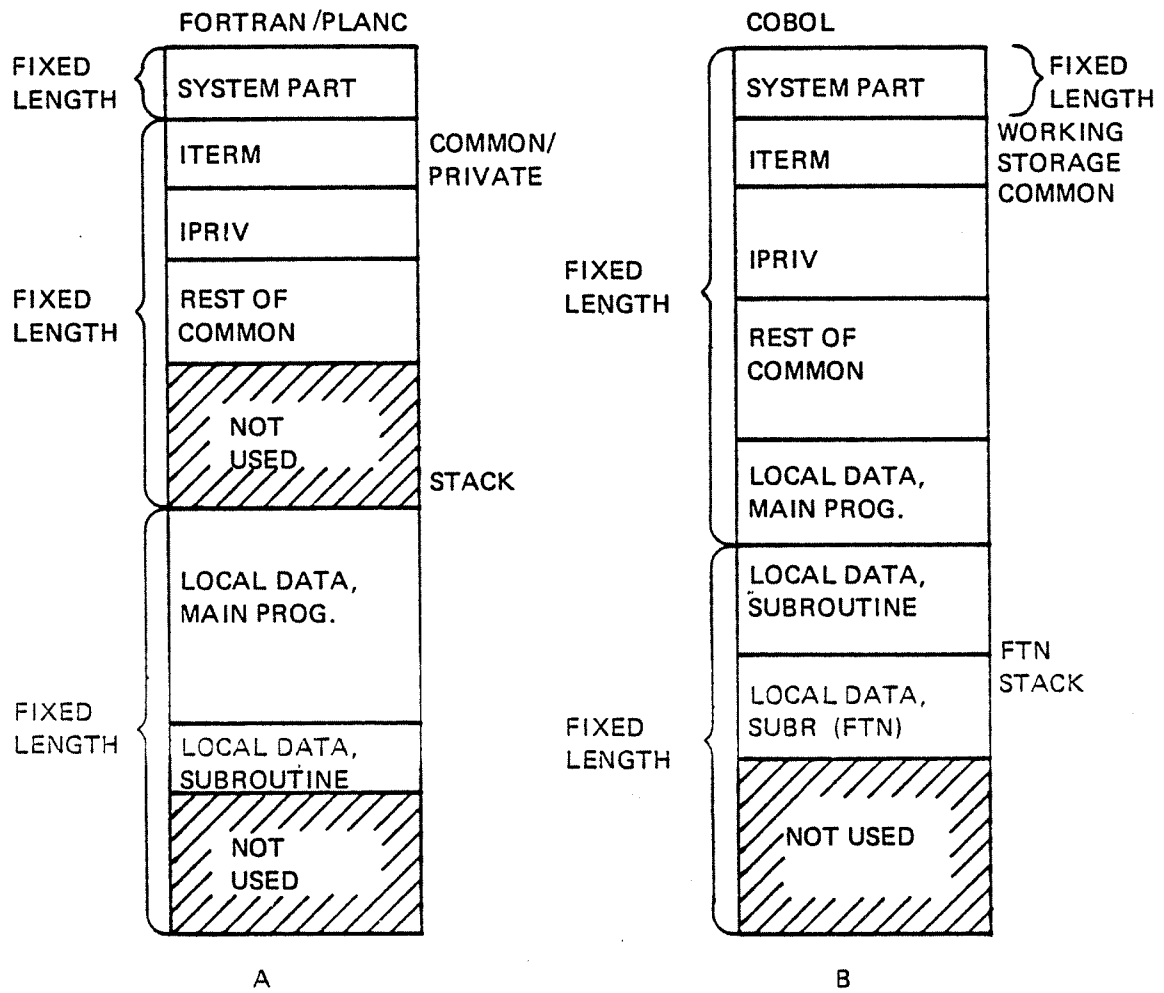


Figure 7.2: Data Areas for FORTRAN AND COBOL Programs

7.1.4 The Size of the Data Area In The ND—100

The total size of the data area for an application program, regardless of the language it is written in, is a fixed number, determined at system generation time. (See Figure 7.2 and Appendix D). This size includes the non-reentrant part of the TPT, about 1400₈ locations of which 1000₈ is used by SIBAS (2000₈ locations if FORTRAN-100 is used).

This size will for FORTRAN programs be the sum of the following:

- non-reentrant TPT
- task common data area (screen handling part and transaction part and unused part)
- stack area

For COBOL programs the size will be the sum of:

- non-reentrant TPT
- WS - task common data area (screen handling part and transaction part and local area)
- COBLIB area
- stack area

The maximum size of working storage is also a system generation parameter. If a COBOL program demands a work area larger than this maximum, an error message will be written at load time (See Appendix C) and the program will not be loaded.

The size of the stack will be whatever is left over after the rest of the areas are allocated. If NSHS is used, it needs approximately 1.5 Kwords in the stack in addition to the data in the common area. SIBAS also uses 0.5 Kwords. However, since this area is freed upon return to the calling routine, NSHS and SIBAS routines can use the same stack area. FOCUS does not use this stack area.

7.2 DATA AREAS IN THE ND-500

All programs running in the ND-500-CPU are reentrant, and the code and data are always separated.

The complete data area, i.e. the local data area, the common data area and the stack area, containing both constant and variable data, is placed in the data-part of the segment where the application is loaded.

Note that, in contrast to the ND-100, *all* data will be given their initial values (initiated by load-time) each time an application is started.

The total size of the data area for an application program is up to 134 megabytes. Note that this data area will not be saved at checkpoint.

In addition to the data area described above, application programs may access the task common data area (see fig. 7.1). This area is common to all application programs and subroutines in both the ND-100 and ND-500 for a transaction. If one application program switches to another, e.g. from an ND-100-application to an ND-500-application, the new application program has the same task common data area, containing the old data.

Note: The first 16-bit word of task common should always contain the terminal device number if the application uses a terminal. If not, it ought to equal zero.

In the ND-500, the task common is labelled by the common label PRIVATE.

FORTTRAN programs may access task common by the statement:

```
COMMON/PRIVATE/<array>
```

PLANC programs may access task common by the statement:

```
IMPORT (COMMON)(type:PRIVATE)
```

COBOL programs may access task common by these statements (also applies to COBOL subroutines):

```
LINKAGE SECTION.
  01 PRIVATE IMPORT.
  02 <array>.
```

Note: If the transaction uses the FOCUS screen handling system, only the first 200 bytes of task common will be available for the user application program. The rest of task common is used by FOCUS, because the FOCUS data must be passed on (by TSWAP) to the next application of the transaction.

The total length of task common (PRIVATE) is the same as in the ND-100 (counted in bytes), and it is fixed at system generation time.

If an application in the ND-100 uses the TSWAP TSR to switch to a new application program in the other machine, the ND-500, or the other way around, special care must be taken that the task common data areas are defined in the same way in both programs. Be aware that the word-length (used for example in data type INTEGER) is 16 bits in the ND-100 and 32 bits in the ND-500.

7.3

LANGUAGE DEPENDENT CONSIDERATIONS

There are few limitations to the full set of FORTRAN and COBOL facilities available on NORD computers when writing programs to be run under TPS. TPS application programs are very similar to general real-time programs using SINTRAN, SIBAS and NSHS/FOCUS. For a discussion of real time programming, see SINTRAN III User's Guide, Chapters 4 and 7.

ND-500 COBOL, FORTRAN and PLANC programs may be tested by using the ND Symbolic Debugger «live» in an ordinary TPS-run. (Chapter 8.1.)

In addition, all languages are extended by the TSR facilities of TPS, including session communication and checkpoint/restart.

In the ND-500, there are no restrictions on language usage, such as initialized data, local data and common areas, and most language features may be used freely. Data in task-common may be accessed in the common area PRIVATE as described in section 7.2. Note that FOCUS uses PRIVATE from ND-500 address 200 upwards. Note also that the COBOL statements ACCEPT, DISPLAY and EXHIBIT cannot be used in the ND-500.

The following sections discuss the languages in ND-100 individually.

7.3.1 **FORTRAN/PLANC in ND—100**

All application programs run under TPS must be reentrant. In FORTRAN this is done by setting the compiler in reentrant mode before compiling. In addition the compiler should be set in a state to generate allocation of 20₈ extra stack locations whenever subroutines, written in MAC or NPL, that use these extra locations are linked to the main program. This makes it possible to keep earlier written subroutines in a lowlevel language unmodified.

The commands to obtain these things are:

```
FTN: REENTRANT ON
FTN: RESERVE—WORK—SPACE ON
```

Constant data in FORTRAN programs must be defined in a BLOCK DATA subroutine as described in section 7.1. Constant data in PLANC-programs may be defined by read-only global data declarations in PLANC-MODULES. Defining variable data areas is also discussed in section 7.1.

When using FORTRAN input-output 2 words following the stack will be used for the administration of a FORTRAN input-output statement. These two words will *not* be checkpointed at runtime, and special considerations should therefore be taken when using FORTRAN input-output from applications.

Application programs may also be run as background programs using the TPS background system (*See Section 8.2*). This is mainly useful for program testing. The debugging facility of FORTRAN/PLANC may be used in background programs by setting the compiler in debug mode and then loading the debugging supervisor. (part of the runtime system). Programs using the debugging facility must *not* be reentrant. When programs are run as background programs they will use background versions of the special TPS facilities available. They will also use the background version of the FORTRAN/PLANC library, FORTRAN—1BANK.

When loading real-time programs to be run under TPS, a special TPS version of FORTRAN-1REENT is loaded automatically due to the TPS load macros (*See section 8.3.1*). This library is usually merged with the TSR file to one BRF file, TPS-LIBRARY:BRF.

When calling TSR routines from PLANC, all routines, must be declared as ROUTINE STANDARD..... in IMPORT statement. PLANC-routines should call CGBRD and CWMSG instead of TGBRD and TWMSG, using type "BYTES" for the parameter <text>.

7.3.2 COBOL in ND—100

As for FORTRAN, COBOL programs running under TPS must be reentrant. This however is done automatically by TPS when the application program is loaded, so the programmer does not have to do anything special.

No data may be initiated in real time COBOL programs, since all data is placed in the data area of the TPT. This includes both constants and variables. Defining data areas in COBOL is discussed in section 7.1.

The SORT function is not yet available under TPS.

ND-100 COBOL programs may also be tested as background programs using the TPS background system. The interactive debugging option can then be used. The ACCEPT, DISPLAY and EXHIBIT commands can only be used in background programs.

Most COBOL programs running under TPS will need routines in the FORTRAN library, since SIBAS and NSHS/FOCUS use them. When testing in background FORTRAN—1BANK must be loaded. When loading real-time programs, the special TPS version of FORTRAN—1REENT is loaded automatically by the TPS load macros (*See Section 8.3.1*).

When compiling COBOL programs, the compiling mode must be two-bank/64KW. This is because code and data must be separated while the logical address space still should be 64KW.

The command to obtain this is:

```
*SEPARATE-CODE-DATA 64K
```

7.3.3 MAC—NPL

Most application programs under TPS will be written in FORTRAN or COBOL, but in a few cases it may be desirable to use a lower level language. Two languages are available for doing this, the MAC assembly language and the NORD—PL system programming language. However, when using these languages, some rules must be followed.

The main rule is that they must simulate reentrant FORTRAN programs when interacting with the system. All interaction with the system is done through calls - SINTRAN monitor calls, TSR calls, SIBAS calls and FOCUS or NSHS calls. These calls must be set up as if they came from a reentrant FORTRAN program.

7.4 PROGRAM STRUCTURE

7.4.1 Application Names and Numbers

Application programs are written as main, real-time, reentrant programs. They may be given any names of 1—6 alphanumeric characters, but in this manual they are given names of the type

APXXX

where XXX is the TPS application number. User applications have positive numbers (1 to 255), while special applications have negative numbers (—10 to 0) and special names. The name is given in the PROGRAM statement in FORTRAN and PLANC, the PROGRAM—ID statement in COBOL, or the)9RT statement in MAC and NPL.

Before the program is loaded, the name must be in TPS—TABLES (*See Section 8.3.1*).

7.4.2 Subroutines

A main program in any language may call reentrant FORTRAN or FORTRAN-compatible subroutines. COBOL subroutines may be called from COBOL only. The subroutines are loaded together with the main program on the same segment*. In the ND-100-they may not exceed a total length of X words of program code where X is an installation dependent maximum (data areas, SIBAS, NSHS/FOCUS and many of the FORTRAN—1REENT routines are not included in this length). In the ND-500, main programs and subroutines may have a length of up to 134 megabytes both for data and code. The subroutines will have access to the task common data area, parameters and their own local data.

Note that FOCUS uses the PRIVATE common area (ND-100 and ND-500) from byte 200 upwards as local data area. This area cannot be used by applications using FOCUS.

If several main programs use common subroutines, each segment must have its own copy of the subroutines. It is, however, possible to have more than one main program on a single segment (in ND-100: if there is room), and programs may then share a common copy of subroutines. (See Figure 7.7). Again in the ND-100 the sum of the lengths of all routines, both main and sub, must not exceed the installation dependent maximum.

*See appendix D for a description of how TPS uses the SINTRAN segment structure in ND-100.

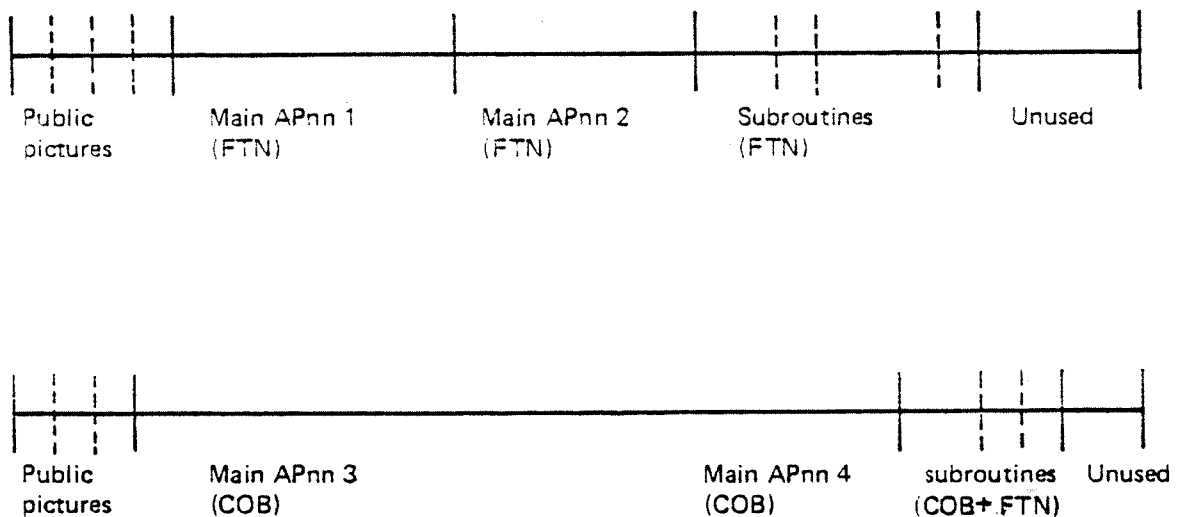


Figure 7.7: Application Segments in ND-100.

A different method of making use of common routines is through the TSWAP TSR (See Figure 7.8). An application program can switch to another one with TSWAP. They will both have access to the task common data, but local variables for the old routine will be lost. Also, there is no return to the old routine when the new one is done. A "manual" return may be programmed by storing the name of the calling program and the return point in the common area, TSWAPing back to the calling program (at the beginning) and jumping to the correct return point. This method can only be used in main programs.

Using main programs as common subroutines in this manner may save both space (only one copy is necessary) and time (less swapping). In addition, in the ND-100 the routines may be larger, since each routine may be up to X words long. However, the method has the disadvantage of losing local variables, and it demands a greater programming effort.

In the ND-500, where there is plenty of room, this kind of structure will not be necessary.

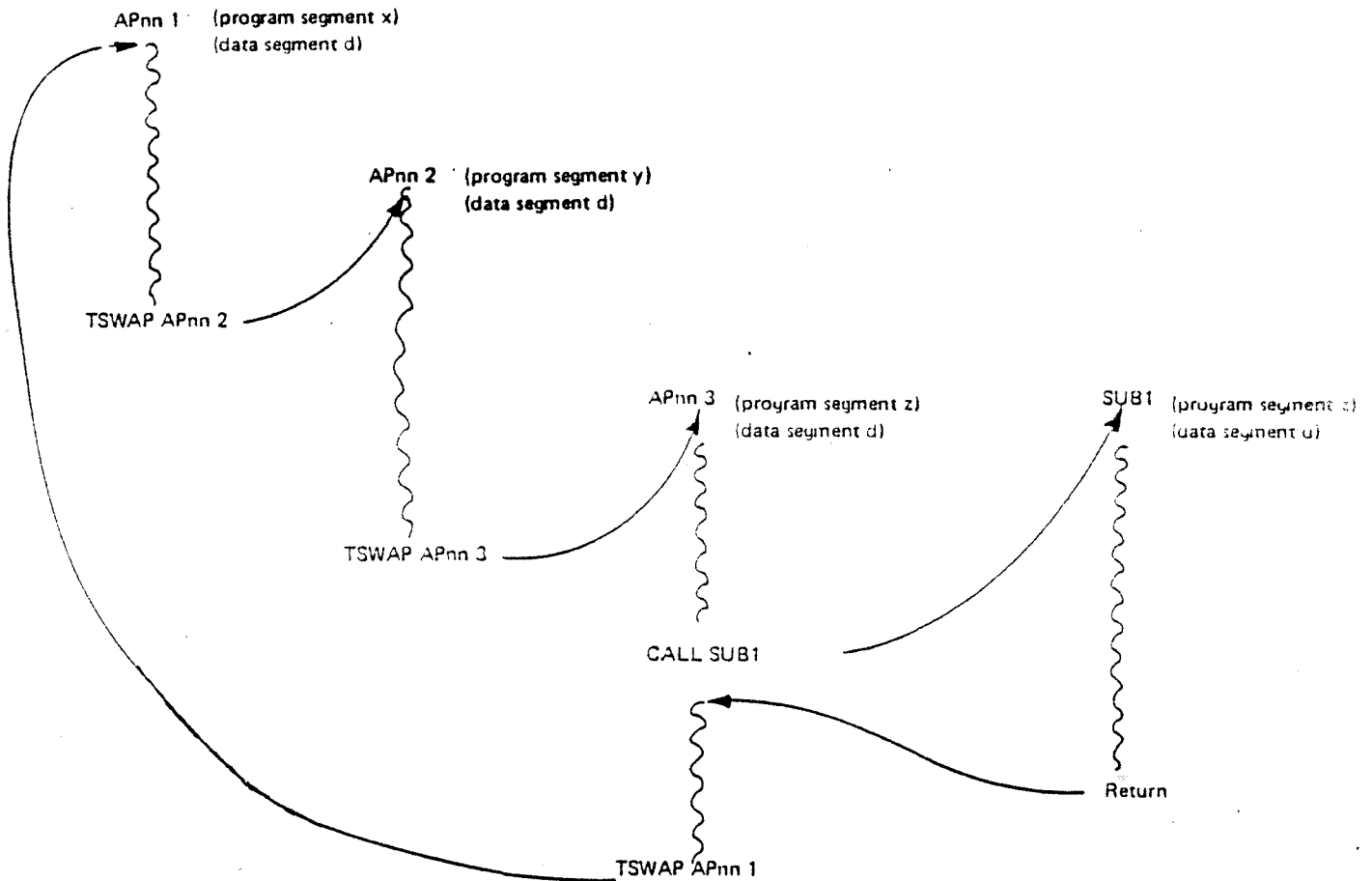


Figure 7.8: Switching Application Programs

7.5 EFFICIENCY

TPS is designed for a combination of fast response times, data protection and ease of use. However, no matter how carefully the system is designed, the manner in which application programs are written does have an influence on both efficiency and data security.

7.5.1 ND-500 Efficiency

A high standard of performance can be obtained by running applications in the ND-500, especially if these applications *only* run in the ND-500 and do not «call» the ND-100 very often.

Calling the ND-100 creates a good deal of system overhead. It is therefore important to keep the number of such calls to a minimum. This can mean the difference between a high-performance system and one with fairly poor performance.

Calls to the ND-100 will be made in connection with most SINTRAN-III monitor calls (e.g. MON CLOCK, MON OUTBT), etc. and many TSR calls (e.g. TRMSG, TSMSG).

NB! When an application on a non-current ND500 domain is started, i.e., when a domain is to be "placed" in the ND-500, the ND-500 monitor will make many disk accesses. This not only takes time, it also loads the system. It is therefore very important that applications that belong together are loaded to the same domain. They can, however, be loaded on several segments in the domain.

Note that when a transaction runs an application in the ND-100, it will not have reserved an ND-500 process. When swapping from an ND-100 application to an ND-500 application, a domain must be "placed" in the ND-500.

7.5.2 Taking Checkpoints

Synchronised and transaction checkpoints are taken automatically, but the programmer must decide if extra transaction checkpoints should be taken with the TTRAN TSR (*See Section 5.2.4*). Taking a transaction checkpoint involves 1 - 6 disk accesses depending on the size of the data area and if the data base is open.

The programmer must also determine if the TTSYN routine should be called to allow synchronised checkpoints to be taken, for example during a long processing and data base sequence without any other calls than SIBAS calls. Calling the TTSYN routine causes very little overhead (a few instructions, - no disk accesses). (*See Section 5.2.3.1*)

7.5.3 Opening and Closing the Data Base

When to open and close the data base is also a decision that may have an influence on efficiency. As mentioned in section 3.1.3, opening and closing the data base itself causes little overhead, since it will probably always be physically open for a dummy user, but transaction checkpoints are taken when it is opened or closed and these involve disk accesses. On the other hand, taking a check-point causes more overhead when the data base is open than when it is closed.

In general, it may be said that if a program has long processing sequences with-out data base accesses, the data base should probably be closed and reopened again. An example would be a long dialogue with the terminal, perhaps involving several transaction checkpoints.

7.5.4 The Working Set

As for all computer systems with paging, the concept of a working set is applicable to TPS programs. Only those pages being used at the moment need to be in main memory, the rest being out on the disk. When a page is needed that is not in main memory, it is read in from the disk, perhaps causing another page to be written out. This page swapping process of course involves overhead and the most efficient programs are those with little swapping.

This means that the program should be organised such that logically connected program sequences are placed close together; this also applies to data. The working set is the number of pages needed to carry out a processing sequence in the program. If this working set is small enough to have all the pages in main memory at one time, no swapping will be done during the processing sequence.

These efficiency considerations may not be necessary in a well-dimensioned system with a fairly low work load and good response time. On smaller systems with high work loads, they may be quite important. In any case, it may be worthwhile to determine which application programs or parts of programs will be used most and devote some extra effort to making these routines efficient.

7.6

REAL TIME VERSUS BACKGROUND

Installations running TPS will usually also have some need of running programs in background mode (timesharing or batch) under SINTRAN (See Figure 7.9). A large part of program testing of ND-100-applications may be carried out in background mode as described in section 8.2, since loading and running application programs are simpler and the debug option is available.

Certain types of interactions with the data base may be better done in back-ground mode than under TPS. Large updating jobs, for example, where a great amount of data has been gathered ahead of time, especially if it is in machine readable form, may best be run as batch jobs. Some users may prefer to use TPS for data entry, gathering data with on-line transactions, storing it in temp-orary files and using it later as input to batch updating jobs. In some cases, it may be an advantage to run large reports as batch jobs, perhaps after stopping TPS to prevent the information in them from being updated as the report is being written. The checkpoint/restart facilities of SIBAS are also available to back-ground jobs, but they must be controlled manually (by the operator), and the programs themselves are not checkpointed, only the data base.

Some tasks *must* be done in background mode before running TPS application programs. The SIBAS data base must be created, private screen pictures must be defined on picture files and public pictures must be defined and dumped to load files. Creating a SIBAS data base is described in the SIBAS User's Manual and defining private pictures is described in the screen handling system manuals. In addition, a special background utility program is available for producing public pictures for NSHS and a program is available for producing public pictures for FOCUS. This is described below.

Updating the data base after data entry transactions
Printing large reports
Testing programs
Creating the data base
Defining pictures

Figure 7.9: Some Background Tasks

7.7 PICTURES FOR NSHS IN ND-100

Pictures used by the Screen Handling System NSHS can be either public or private. Private pictures are stored in a file and read from the file when they are used (*See Figure 7.10*). Public pictures are stored together with the application code on the application segment. Several pictures can be stored together with an arbitrary number of applications on each sequence. When running those application together with NSHS, the picture data need not be read in at run-time. It can be referred to directly through NSHS by specifying "public picture" in ITERM(5). Thus the execution is speeded up by omitting mass storage access of the picture-file.

Further, several applications can share pictures or picture elements. Thus swapping activity is essentially decreased due to the fact that:

- less physical memory is used
- public picture areas are not written into and the pages not written back to disk.

7.7.1 Defining Private Pictures for NSHS

Picture formats are defined through use of an interactive utility program, SCREEN—DEFINITION, at the display terminal. Picture definitions are stored in ordinary SINTRAN files of two different types. The output from the picture definition is called the "source picture" and is stored in the source picture file. Several pictures can be stored in the same file, identified by their individual picture names.

Before the picture definitions are used by application programs, they are compiled using SCREEN—DEFINITION into a certain run-time format called the "object picture" and stored in the object picture file. Here also one file may contain several pictures which can be independently compiled.

Comprehensive editing facilities allow the operator to create, replace, repeat or remove any part of a picture. For a detailed description of how to create and maintain picture definitions, see the NORD Screen Handling System, Chapter 2.

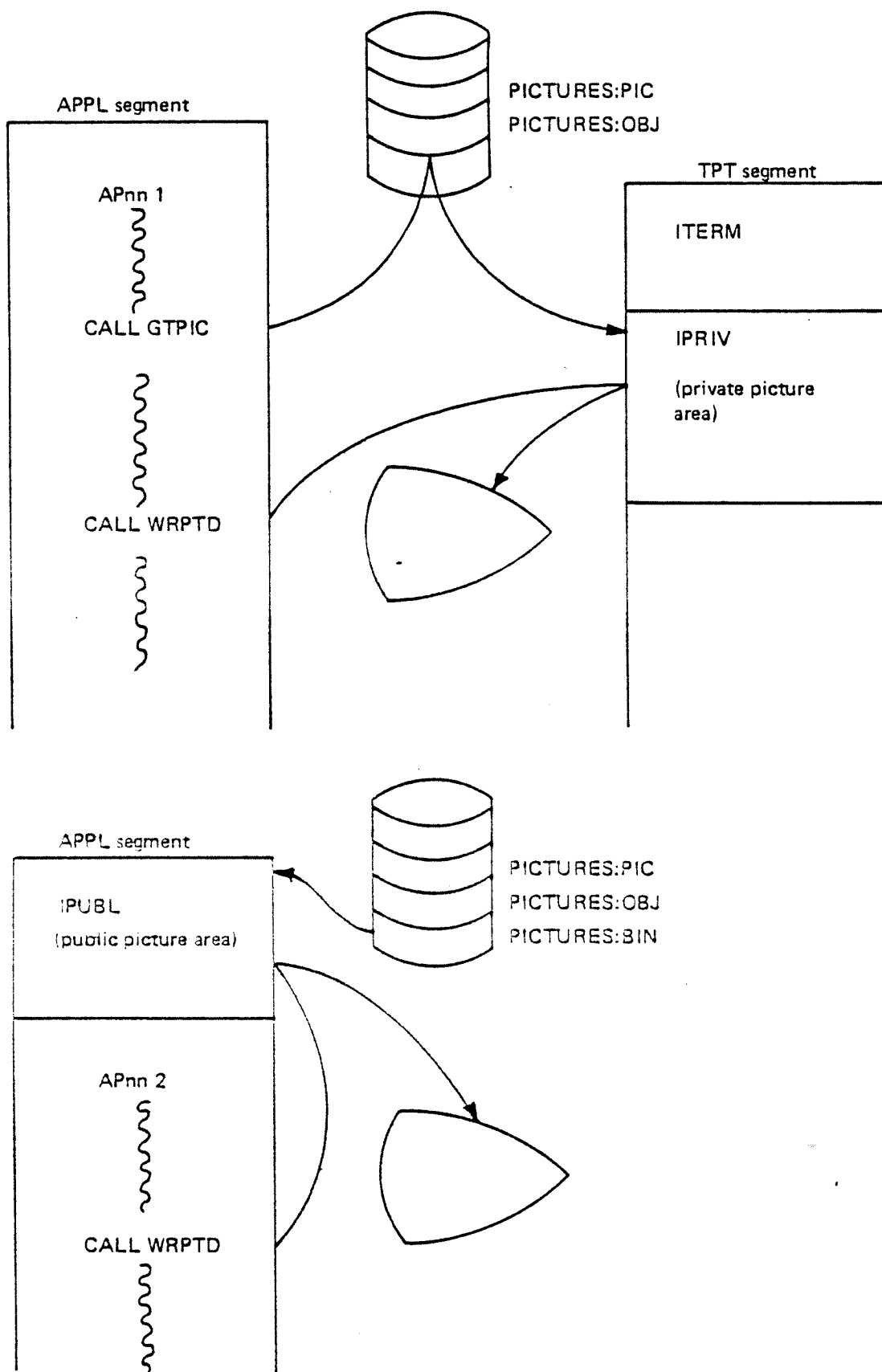


Figure 7.10: Private and Public Pictures

7.7.2

Object pictures defined in the above manner may be used by TPS application programs as private pictures. However, for the reasons mentioned above, it may be desirable to use public pictures instead. NORD TPS supplies a background program, PUBLIC—PICTURE, which produces a public picture data file from object pictures. This file can be loaded together with the application code on an application segment to form the public picture area.

The user must specify the following parameters which are asked for:

- picture dump file: the file where the picture data is to be dumped for loading on to the segment
- number of pictures
- picture file name(s): the object picture file(s) produced by SCREEN—DEFINITION
- picture name(s): the name(s) of the picture(s), which are to be loaded together in the Public Picture area.

Example

@PUBLIC-PICTURE

```

%NORD-TPS PUBLIC PICTURE LOADING%

```

ENTER PICTURE DUMP FILE NAME: PUBLIC:BIN

ENTER NO. OF PICTURES: 2

ENTER PICTURE FILE NAME (MAX 20 CHARACTERS)
OR '@' TO TERMINATE LOADING: TPS-PIC:OBJ

ENTER PICTURE NAME (MAX 8 CHARACTERS)
OR '@' FOR NEW PICTURE FILE NAME: SIGNON

ENTER PICTURE NAME (MAX 8 CHARACTERS)
OR '@' FOR NEW PICTURE FILE NAME: MENU

PICTURES LOADED. OCTAL SIZE: 000601

STOP 0

9

7.7.3 Loading Public Pictures for NSHS

The picture dump file, which is produced by the PUBLIC—PICTURE program, is used as an input file when loading the applications. It comprises the picture data to be loaded together with one or more applications on one or more segments. The public picture area will always start in location 3 of the application segment. It is defined with a macro statement of the form:

↑LOAD—SEGMENT, seg-no, public-pic-file, reent seg-name, sub-macros;

See Section 8.3.1 for use of this macro

In summary, the steps in defining and loading public pictures are these:

- Use the SCREEN—DEFINITION program to define pictures on object picture files.
- Use the program PUBLIC—PICTURE to produce picture dump files.
- Define the picture dump files to be input files for loading. This information is to be entered into the file
(TPS—USER) SPECIFY—LUAP:SYMB
- Prepare the application programs to refer to the Public Picture area by giving the value 1 to ITERM(5). When using the Private Picture area this value is to be set to 0.
- Load the public pictures and applications by running the mode files
(TPS—USER) BUILD—LUAP:MODE
(TPS—USER) LOAD—USER—APPL:MODE
as described in section 8.3.1.

7.8 PICTURES FOR FOCUS

Forms used by the FOCUS screen handling system can be either public or private. Private forms are stored in a file and read from the file when they are used. Public forms are stored together with the application code on the application segment. Several forms can be stored together with an arbitrary number of applications. When running those applications together with FOCUS, the form data need not be read in at run-time. Thus the execution is speeded up by omitting mass storage access of the form file.

Further, several applications can share forms or form elements. Thus swapping activity is essentially decreased due to the fact that:

- less physical memory is used
- public form areas are not written into and the pages not written back to disk.

Forms are defined through use of an interactive utility program, FOCUS-DEF, at the display terminal. Form definitions are stored in ordinary SINTRAN files. Several forms can be stored in the same file, identified by their individual form names. Comprehensive editing facilities allow the operator to create, replace, repeat or remove any part of a form. For a detailed description of how to create and maintain form definitions, see the FOCUS screen handling system manual (ND-60.137).

Using private forms, the form file name must be declared by the call FCDECF in FOCUS.

7.8.1 Public Pictures for FOCUS

It is desirable to use public forms in TPS, for the reasons mentioned above, when running the system in full production. In the test phase, however, it is easier to use private forms.

ND TPS supplies a background program, FC-PUBLIC, which converts the form file into a public form data file. This file can be loaded together with the application code on an application segment to establish the public forms area.

Use the program FC-PUBLIC-100 or FC-PUBLIC-500 in order to produce a public form data file to be loaded to the ND100 or ND500 respectively. The public form data file is used when loading applications, by inserting the file name for the parameter "publ-pic-file" in these macros:

```
↑LOAD-SEGMENT,seg-no,publ-pic-file,reent-seg-name,sub-macros;
↑LOAD-500SEGMENT,domain-name,seg-name,publ-pic-file,sub-macros;
```

See section 8.3.1 for use of these macros.

The public area will always start in location 4 of the application segment in the ND100, and location 0 in the ND500.

The application programs must call FCINITE with the parameter init-array(6) ≠ 0 in order to use public forms. Then the FOCUS call FCDECFF must not be called.

In the ND100, init-array(6)=4 (address of first location of the public form area).

In the ND500, init-array(6)=segment number of the public form area. Segment number equal to 0 (zero) must not be used. The application segment number may be obtained by the TSR-call TAPST.

In summary, the steps in defining and loading public forms are these:

- use the FOCUS-DEF program to define pictures on form files.
- use the FC-PUBLIC-XXX program to produce public form data files.
- define the public form data files to be input files for loading, by editing the file (TPS-USER)SPECIFY-LUAP:SYMB.
- prepare the application programs to refer to the public form area by giving the appropriate value to the parameter init-array(6) in the call to FCINITE.
- load the public forms and applications by running the mode files

(TPS-USER) BUILD-LUAP:MODE	(as user TPS-USER)
(TPS-USER) LOAD-USER-APPL:MODE	(as user RT)

 as described in chapter 8.3.1.

In the ND500, all forms used by all applications inside one domain may be gathered on only one public form file if desired. After running the FC-PUBLIC-500 program, the public form data file may be loaded to its own segment inside this domain by issuing these commands:

```

END LINKAGE-LOADER
ND-Linkage-Loader-
NLL: SET-DOMAIN ''domain-name''
NLL: SET-SEGMENT-NUMBER segm-no
NLL: OPEN-SEGMENT ''segment-name'',,
NLL: LOW-ADDRESS 0,0
NLL: LOAD-SEGMENT public-form-data-file
NLL: END-DOMAIN
NLL: EXIT

```

®

8

COMPILING AND LOADING PROGRAMS

This chapter shows how to compile and load application programs both as RT programs in the ND-100 and the ND-500 and as background programs in the ND-100.

8.1

TESTING OF ND-500—APPLICATIONS

When application programs are run under TPS in the ND-500, they must be compiled by an ND-500-compiler, loaded with the LINKAGE-LOADER, and started under the control of a TPT. Section 8.3 describes how to load the program to the TPS-system.

During the programming and testing phase in the ND-500, it may be easier to run them in a special Debug-mode in ND-500 under TPS, which allows you to use the ND Symbolic Debugger. The program then ought to be compiled with DEBUG-MODE ON.

In 500-debug-mode, the symbolic debugger is run on one terminal and the application on another terminal.

You start a transaction in Debug-mode in the ND-500 by activating application no. —3 from OPCOM, giving the terminal device number to be used by the symbolic debugger as parameter 1. Parameter 2 may be zero. Note that the debugger terminal must not be the same as the application terminal. The application —3 (on the debug terminal), will then prompt you according to the OPCOM-command ACTIVATE-APPLICATION.

In 500-debug-mode, you will enter the Symbolic Debugger on the debug terminal each time a 500-application is entered, and will be free to inspect/change locations and stop/run the application. See the ND Symbolic Debugger Manual ND-60.158.

8.2 BACKGROUND TESTING OF ND-100 APPLICATIONS

When application programs are run under TPS in the ND-100, they must be compiled as real time (RT), reentrant programs, loaded with the RT loader, and started under the control of a TPT. However, during the programming and testing phase, it may be easier to run them as background (time-sharing or batch) programs directly under SINTRAN. Loading and starting the program is simpler and the interactive debugging options of both FORTRAN and COBOL are available.

8.2.1 The TPS Background System

The TPS background system is a set of subroutines and programs running in ND-100 that simulate a real time TPS environment. The subroutines are loaded together with the user application programs and provide simulated TSR routines for the application program. Background versions of the SIGNON and SELECT special applications are also provided, plus a special program, TPS:PROG, which is used to initiate the transaction.

The background system is started by logging in as a SINTRAN timesharing user and giving the RECOVER TPS command. When this is done, functions covered by TPOPEN will be carried out for the background terminal. The terminal number is checked against the terminal configuration generated for TPS. If it is included, NSHS initialisation is done and SIGNON/SELECT started. As under TPS SIGNON will ask for and control the user name and password and SELECT will control the menu choice, as described in chapter 6. Finally the user application will be started as under TPS, with access to the task common data area.

8.2.2 Available Facilities in the TPS Background System

The available facilities include both TSR simulation routines and other special TPS functions, such as:

- control of name/password
- presentation of menu choices
- entry to user programs as under TPS
- initialising and use of NSHS according to TPS terminal configuration parameters
- full use of the task common data area between independent application programs (i.e. main programs)
- use of the interactive debugging options of FORTRAN and COBOL

Most TSR simulation routines will only consist of a return to the calling program, but a few will carry out an action similar to that under TPS. These routines are:

- TSWAP - the new application program will be started with access to the task common data area
- TSTOP - the transaction will be terminated normally or abnormally
- TWMSG - a message will be written to the user terminal instead of the TPS operator
- TACTV - if called by the TPOPEN application, the new application will be started as for TSWAP, else return to calling program with no action

8.2.3 Load-Common and Save-Common Routines

When application programs are run under the TPS background system, they may make free use of the TSWAP facility to switch from one main program to another. Since it is normally not possible to have common data areas between independent main programs in background mode, special provision must be made for making the task common data area available to all application programs. This is done by two routines, load-common (LCOMMO) and save-common (SCOMMO).

```
CALL LCOMMO ( <array> )
CALL 'LCOMMO' USING <array>.
```

```
CALL SCOMMO ( <array> )
CALL 'SCOMMO' USING <array>.
```

Parameters:

<array>	COBOL: the first variable at the beginning of the task common data area (i.e. WORKING-STORAGE) FORTRAN: the first variable in the task common data area (i.e. ITERM)
---------	--

Examples

```
CALL LCOMMO(ITERM)
```

```
CALL 'SCOMMO' USING TASK-COMMON-AREA.
```

Load-common (LCOMMO) loads the task common area from a file on the disk and should always be called as the first executable statement in the application program. Save-common (SCOMMO) saves the area back on the disk and should be called right before the program does a TSWAP to another program.

Dummy versions of the LCOMMO and SCOMMO routines are included in the TPS library routines for real-time programs. Therefore it is not necessary to remove these calls when the application program is done being tested and is loaded as a real-time program.

8.2.4 Running the Background System

When the debugging option is used, the program must be run under the control of the loader NRL, in order to use the debugging facilities, i.e. the loader must be dumped together with the program and the RUN command given to the loader after loading the program. This means that testing with the debugging option should be done in the following steps:

- Make sure that the following programs exist as PROG files:

```
TPS:PROG
SIGNON:PROG
SELECT:PROG
```

- Load applications with NRL
- Dump all of memory with the @MEMORY and @DUMP commands
- Start TPS:PROG by writing TPS on the terminal to SINTRAN
- Wait for SIGNON to send the (user-defined) screen picture and then give your name and password (you must be in the user-table)
- Wait for the menu picture and indicate the correct choice to start your application (the application must be in the menu table)
- If the application exists as a PROG file (i.e. has been loaded and dumped) it will be started *in the loader*
- The RUN command must now be given to the loader to start the application *in the debugger*
- Debugging commands can be given and the application itself started with the CONTINUE command

Examples of compiling, loading and running programs with the debugging option are given in section 8.1.6.

8.2.5 Testing in Background Mode

When application programs are tested in background mode, several facilities are available which are not available when running them as real-time programs, such as the possibility of initialising data areas at compile time, using the ACCEPT, DISPLAY and EXHIBIT statements in COBOL, etc. These may be used, but it is recommended that as few changes as possible be introduced during background testing since converting to RT programs should be as simple as possible.

However, some special considerations must be taken when testing in background mode. Among them are the following:

- the device number for the terminal (in ITERM (1) if NSHS is used) must be 1 (this is done automatically by the background system).
- initialised data areas will be cleared (but not COMMON areas)
- calls to NSHS, SIBAS and SINTRAN routines allowed in background will be executed in the usual way, but special TPS routines, such as TRMSG (read-message) and TACTV (activate concurrent task) may be different in background, as mentioned above. This may demand some changes to the program
- the FORTRAN compiler should not be set to reentrant mode, whereas the debug mode may be used. Using the debugging option is described in chapter 13 of the FORTRAN Reference Manual.
- the background versions of NSHS and the SIBAS interface module (DML-SIMULATOR-BACKGROUND) must be loaded.
- The TPS background library, the TPS user library and the FTN library (FORTRAN-1BANK) must be loaded with programs in all languages. COBLIB must be loaded with COBOL programs.
- if a SIBAS system is to be used that is already running (for example under TPS), no further action needs to be taken to use it from a background program. However, if SIBAS is not yet running, it must be started up as an RT program with the RT command (*See appendix A, SIBAS on NORD-10, of the SIBAS User's Guide*). The data base of course must have been defined and created. Doing this is discussed in chapter 9.
- the TPS background system can be used to start the transaction and save the task common data area

These points are summarized in Figure 8.1.

Makes as few changes as possible
Uninitialised data areas are cleared
Do not use reentrant mode
Debug mode may be used
Load background versions of SIBAS, NSHS
Load TPS background library TPS — USER library FORTRAN-1BANK
Start SIBAS
Use the TPS background system to start the transaction

Figure 8.1: Testing in Background Mode

8.2.6 Compile and Load Examples

Examples are given here for compiling FORTRAN and COBOL programs in the debug mode, starting the background system, loading the programs with the relocating loader and running them. Together with the program itself is loaded NSHS (background version) or FOCUS, the SIBAS DML simulator (background version), the TPS background library, the TPS user library and the COBOL runtime system. There are many possibilities in the debugging mode: setting breakpoints, inspecting locations, tracing, stepping through the program. The examples show running the program with the trace mode on.

Example - FORTRAN

@FORTRAN-100
ND-100 ANSI 77 FORTRAN COMPILER

FTN: DEBUG-MODE ON
FTN: COMPILE AP022-SYMB,L-P,AP022-BRF

265 STATEMENTS COMPILED

FTN: EXIT

@NRL

- NORD-10 RELOCATING LOADER -

*LOAD AP022-BRF
*LOAD DML-SIMULATOR-BACKGROUND
*LOAD TPS-BACKGROUND-LIBRARY
*LOAD NSHS-BACKGROUND
*LOAD FORTRAN-1BANK
*EXIT
@MEMORY 0 177777
@DUMP AP022 1 1
@TPS

Wait for SIGNON picture, enter name and password.

Wait for MENU picture, enter correct choice to start appl.

*RUN
&TRACE AP022,100 AP022,200

&CONTINUE execute the application program

&EXIT
@

Example - COBOL

@COBOL

*** NORD-10 COBOL COMPILER ***

*DEBUG-MODE

*COMPILE AP023-SYMB,L-P,AP023-BRF

** 0 DIAGNOSTIC MESSAGE(S) **

@NRL

- NORD-10 RELOCATING LOADER -

*LOAD AP023-BRF

*LOAD DML-SIMULATOR-BACKGROUND

*LOAD TPS-BACKGROUND-LIBRARY

*LOAD TPS-USER-LIBRARY

*LOAD NSHS-BACKGROUND

*LOAD COBLIB

*LOAD FORTRAN-1BANK

*DUMP AP023:PROG

*EXIT

@TPS

Wait for SIGNON picture, enter name and password.

Wait for MENU picture, enter correct choice to start appl.

The application will start in the debug monitor.

--- NORD COBOL INTERACTIVE DEBUG MONITOR ---

*TRACE-ON

.

.

*RUN

execute the application program

.

.

*EXIT

@

8.3 REAL TIME PROGRAMS

This section describes how to load application programs as RT programs to the TPS system. When new application programs are loaded, application information must also be entered into the application table (TPS-TABLES).

This section describes the procedures for updating the application table and for loading applications, both new ones and replacements. A special program, GPM (General Purpose Macrogenerator), is available to simplify the commands to the RT loader and the Linkage Loader (for ND-500). This program should be used to generate mode files for loading applications, and the load procedure described in this manual assumes it is available. All TPS files in this section belong to a special user, called TPS—USER here. This user name is a system generation parameter, and if another name is specified, the correct user name will automatically be put into the files wherever necessary.

When loading a program for the first time, it may be best to "test load" it separately before loading it together with other programs. Error messages do not always indicate which program caused the error.

8.3.1 The Loading Procedure

The following steps should be followed when loading application programs: (summarised in Figure 8.2):

- Make any changes necessary to the ND-100-programs to run them under TPS instead of as background programs. Remember to declare them as RT programs with the proper program names.
- Compile the programs - in the reentrant mode if FORTRAN-100. The object program files can have any names and belong to any user. However, the file *type* must be BRF for ND-100 and NRF for ND-500.
- If new application programs are to be loaded, enter TPS—USER, fetch PED or QED and read the file TPS-TABLES:SYMB.

Compile the program Reentrant mode if FORTRAN-100
Update TPS-TABLES:SYMB
Run COMPILE-TABLES:MODE
Update SPECIFY-LUAP:SYMB
Run BUILD-LUAP:MODE
Run LOAD-USER-APPL:MODE

Figure 8.2: RT Load Procedure

- Make sure that the entries corresponding to the applications are present in the table and that the names of the applications correspond to the indexes in TPS-TABLES. Fill in the desired values in the application entries and make sure that all values are filled in - empty fields are not allowed. An entry has the following format:

ENTRY APXXX, SAWYY, SPRZZ

APXXX = the name of the application program: must be the same as the program name

SAWYY = the application checkpoint weight; "heavy" weight causes frequent checkpoints, light weight infrequent, allowed values SAW0 - SAW15

SPRZZ = the application priority within TPS, allowed values SPR0 - SPR15. All ND100-applications may be considered as one group and ND500-applications as another group. SPRZZ gives the priority inside the appropriate group.

Example:

ENTRY AP010, SAW9, SPR12

- After the updated TPS—TABLES file has been written back, it must be compiled using the mode file COMPILE-TABLES:MODE. This mode file places the BRF code in the file (TPS-USER)TPS-TABLES:BRF.
- The applications (both new ones and replacements) must now be loaded. Enter TPS-USER. Use PED or QED to read the file SPECIFY-LUAP:SYMB and specify the information necessary to load the applications. It is specified using these macros.

MAIN MACROS

- ↑ LOAD—SEGMENT, seg-no, public-pic-file, reent seg-name, sub-macros;
indicates that a ND-100 segment is to be cleared and loaded. The segment number is given and the public picture file, if the segment is to contain public pictures (*See Section 7.7 and 7.8*). The reentrant segment name must be specified if your TPS-configuration contains several reentrant segments. If so, this parameter names the reentrant segment which will be used when loading and running the applications on this application segment. The name must equal one of the names in the macro NAME-REENT-SEG in the file (TPS-USER) TPS-CONF. Consult your TPS system supervisor.

↑ **LOAD—500SEGMENT**, domain name, segment name, public-pic-file, sub-macros;

indicates that an ND500-segment is to be cleared and loaded. The domain and segment specified may exist or may be new. In the latter case, they will be created. The public picture file must be of type NRF, or it must be empty if the segment is not to contain public pictures.

SUB-MACROS

↑ **ADD—APPL**,file-name, prog-id;

indicates an application program that is to be loaded on the ND-100 segment given in the last load-segment macro. The name of the BRF file containing the application and the program name APXXX must be given.

↑ **ADD-500 APPL**, file name, prog-id;

indicates an application program that is to be loaded on the ND-500-segment given in the last load-500-segment macro. The name of the NRF file containing the application and the program name APXXX must be given.

↑ **ADD—COB—SUBROUTINES**, file-name;

indicates that user COBOL subroutines in the given file are to be loaded on the ND100-segment. This submacro can only exist once within a ↑ **LOAD—SEGMENT** macro.

↑ **ADD—UNIT**, file-name;

indicates that user subroutines in the given file are to be loaded on the ND-100 segment. These subroutines can be called by all application programs and other subroutines on the segment. The subroutines will be loaded regardless of whether they are called or not.

↑ **ADD—500UNIT**, file-name;

indicates that user subroutines in the given file are to be loaded on the ND-500-segment. These subroutines can be called by all application programs and other subroutines on the segment. The subroutines will be loaded regardless of whether they are called or not.

An example of the macro input to the SPECIFY-LUAP:SYMB file with 4 ND-100-applications and 2 ND-500 applications follows. One ND-100-segment is loaded with 2 FORTRAN applications, one with 2 COBOL applications and one ND-500-segment with 1 COBOL and 1 FORTRAN application. Both use subroutines and public pictures.

Example

```

^CRM0D;
@ENTER RT,,,30
@HEAD LUAP
^ICRM0D;
^%,Load 2 FORTRAN programs on segment 205;
^LOAD-SEGMENT,205,(USER-NAME)PUBLIC:BIN,,
^ADD-APPL,(USER-NAME)FTN-APXX1,APXX1;
^ADD-APPL,(USER-NAME)FTN-APXX2,APXX2;
^ADD-UNIT,(USER-NAME)FTN-SUBS;
;
^%,Load 2 COBOL programs on segment 206;
^LOAD-SEGMENT,206,(USER-NAME)PUBLIC:BIN,,
^ADD-APPL,(USER-NAME)COB-APXX3,APXX3;
^ADD-APPL,(USER-NAME)COB-APXX4,APXX4;
^ADD-COB-SUBROUTINES,(USER-NAME)COB-SUBS;
^ADD-UNIT,(USER-NAME)FTN-SUBS;
;
^%, Load 1 Fortran and 1 Cobol program on the 500;
^LOAD-500SEGMENT,DOM-A,SEG-1,,
^ADD-500APPL,(USER-NAME)5-FTN-APXX5,APXX5;
^ADD-500UNIT,(USER-NAME)5-FTN-SUBS;
^ADD-500APPL,(USER-NAME)5-COB-APXX6,APXX6;
^ADD-500UNIT,(USER-NAME)5-COB-SUBS;
;
^CRM0D;
EXIT
^%,This MODE file must be run last;
@MODE (TPS-USER)SAVE-TATAB:MODE,,

```

- After the SPECIFY-LUAP:SYMB file has been updated and written back, the BUILD—LUAP:MODE file must be run. This activates the General Purpose Macrogenerator, GPM. GPM uses several system files and the above macros as input and produces a new mode file, LOAD—USER—APPL:MODE, as output. BUILD—LUAP:MODE is as follows:

@GPM	
YLOAD—USER—APPL:MODE	output file and "Y" answer to GPM question)
GPM—LIBRARY	system file
INIT—MD—ADDR	system file
TPS CONF	system file
GLOBAL—MACROS	system file
USER—MACROS	system file
SPECIFY-LUAP	user file

- Finally the mode file LOAD—USER—APPL:MODE must be run under the user RT. Before it is run, make sure that no applications on the segments to be loaded are active if TPS is running and give the SET—UNAVAILABLE operator command.

There is one exception to this rule. It is possible to load a new version of a application program while the old one is active if it is done as follows:

- the new version must be loaded on a different segment
- both segments must only contain the one application program (plus subroutines if used).

8.3.2 Programs and Files Required

The loading procedure described here requires the following programs and files:

Programs:

PED or QED
RT-loader
GPM
HEAD
ND500 LINKAGE-LOADER (If ND500-TPS)

Files:

(USER—NAME)user-progs:BRF
(USER—NAME)user-subs:BRF
(USER—NAME)public-pics:BIN
(USER—NAME)public-pics:NRF (If ND500-TPS)

(TPS—USER)TPS—TABLES:SYMB
(TPS—USER)COMPILE—TABLES:MODE

(TPS—USER)SPECIFY—LUAP:SYMB
(TPS—USER)BUILD—LUAP:MODE
(TPS—USER)LOAD—USER—APPL:MODE

(TPS—USER)GPM—LIBRARY:SYMB
(TPS—USER)INIT—MD—ADDR:SYMB
(TPS—USER)TPS—CONF:SYMB
(TPS—USER)GLOBAL—MACROS:SYMB
(TPS—USER)USER—MACROS:SYMB
(TPS—USER)SAVE—TATAB:MODE

Note that there may be several sets of the 3 files

(TPS—USER)SPECIFY—LUAP:SYMB
(TPS—USER)BUILD—LUAP:MODE
(TPS—USER)LOAD—USER—APPL:MODE

varying the names. Each set of 3 can be used to load a group of application programs. In this manner, the load files can be saved and used again if a particular set of application programs is to be reloaded without changing the files every time.

8.3.3 Compile and Load Example

The complete procedure for loading the 2 FORTRAN and 2 COBOL programs in the ND-100 and 1 FORTRAN and 1 COBOL program in the ND-500 from section 8.3.1 is given here. It is assumed that they are all new applications (TPS—TABLES must be updated), and that the FORTRAN programs are put on one segment and the COBOL programs on another. They both use a set of FORTRAN subroutines and public pictures.

Example

Compile the programs

ESC

ENTER USER-NAME

.

.

@FORTRAN-100

ND-100 ANSI 77 FORTRAN COMPILER

FTN: REENTRANT-MODE ON

FTN: COMPILE FTN-APXX1:SYMB,L-P,FTN-APXX1:BRF

NNN STATEMENTS COMPILED

FTN: COMPILE FTN-APXX2:SYMB,L-P,FTN-APXX2:BRF

NNN STATEMENTS COMPILED

FTN: COMPILE FTN-SUBS:SYMB,L-P,FTN-SUBS:BRF

NNN STATEMENTS COMPILED

FTN: EXIT

@COBOL

*** NORD-100 COBOL COMPILER ***

*SEPARATE-CODE-DATA 64KW

*COMPILE COB-APXX3:SYMB,L-P,COB-APXX3:BRF

** 0 DIAGNOSTIC MESSAGE(S) **

*COMPILE COB-APXX4:SYMB,L-P,COB-APXX4:BRF

** 0 DIAGNOSTIC MESSAGE(S) **

*EXIT

@ND-500

ND-500 MONITOR

N500: FORTTRAN-500

ND-500 ANSI 77 FORTRAN COMPILER

FTN: COMPILE 5-FTN-APXX5,L-P,5-FTN-APXX5:NRF

FTN: COMPILE 5-FTN-SUBS,L-P,5-FTN-SUBS:NRF

FTN: EXIT

N500: COBOL-500

ND-500 COBOL COMPILER

*COMPILE 5-COB-APXX6,L-P,5-COB-APXX6:NRF

*COMPILE 5-COB-SUBS,L-P,5-COB-SUBS:NRF

*EXIT

N500: EXIT

@LOG

Update and compile TPS-TABLES

ESC

ENTER TPS-USER

.

@PED or QED

edit TPS-TABLES

.

list TPS-TABLES

ENTRY AP001,SAW10,SPR10

ENTRY AP002,SAW5,SPR2

.

ENTRY APXX1,SAW8,SPR2

ENTRY APXX2,SAW8,SPR5

ENTRY APXX3,SAW5,SPR2

ENTRY APXX4,SAW2,SPR15

ENTRY APXX5,SAW4,SPR9

ENTRY APXX6,SAW7,SPR11

)LINE

*W TPS-TABLES

*EXIT

@MODE COMPILE-TABLES:MODE L-P

Update SPECIFY-LUAP:SYMB

@PED or QED

edit SPECIFY-LUAP:SYMB

list SPECIFY-LUAP:SYMB

```

^CRMOD;
@ENTER RT,,,30
@HEAD LUAP
^ICRMOD;
^%, Load 2 Fortran programs on segment 205;
^LOAD-SEGMENT,205,(USER-NAME)PUBLIC:BIN,,
^ADD-APPL,(USER-NAME)FTN-APXX1,APXX1;
^ADD-APPL,(USER-NAME)FTN-APXX2,APXX2;
^ADD-UNIT,(USER-NAME)FTN-SUBS;
;
^%, Load 2 Cobol programs on segment 206;
^LOAD-SEGMENT,206,(USER-NAME)PUBLIC:BIN,,
^ADD-APPL,(USER-NAME)COB-APXX3,APXX3;
^ADD-APPL,(USER-NAME)COB-APXX4,APXX4;
^ADD-UNIT(USER-NAME)FTN-SUBS;
;
^%, Load 1 Fortran and 1 Cobol program on the 500;
^LOAD-500SEGMENT,DOM-A,SEG-1,,
^ADD-500APPL,(USER-NAME)5-FTN-APXX5,APXX5;
^ADD-500UNIT,(USER-NAME)5-FTN-SUBS;
^ADD-500APPL,(USER-NAME)5-COB-APXX6,APXX6;
^ADD-500UNIT,(USER-NAME)5-COB-SUBS;
;
^CRMOD;
EXIT
^%, This mode file must be run last;
@MODE (TPS-USER)SAVE-TATAB:MODE,,

```

write SPECIFY-LUAP:SYMB

list BUILD-LUAP:MODE

@GPM

YLOAD-USER-APPL:MODE

GPM-LIBRARY
INIT-MD-ADDR
TPS-CONF
GLOBAL-MACROS
USER-MACROS
SPECIFY-LUAP

output file
(after Y answer to GPM question)
system file
system file
system file
system file
system file
user file

exit

Run BUILD-LUAP

@MODE BUILD-LUAP:MODE L-P
@LOG

Run LOAD-USER-APPL

ESC

ENTER RT
@MODE (TPS-USER)LOAD-USER-APPL:MODE L-P
or
@APP-BATCH 1 (TPS-USER)LOAD-USER-APPL:MODE L-P

APPENDIX A**APPLICATION NUMBERS FOR SPECIAL APPLICATIONS**

SINOF	0	
SLECT	- 1	
SINON	- 2	
TPMON	- 3	(ND500-TPS only)
ABEND	- 4	
RSTRT	- 5	
CHECK	- 6	
ROLBK	- 7	
RCOVR	- 8	
TPCLO	- 9	
TPOPEN	-10	

APPENDIX B

SAMPLE PROGRAMS

To be inserted later

APPENDIX C

ERROR MESSAGES

1. COMPILE TIME ERRORS

Compile time error messages are described in the manual for the language the program is written in, i.e.

ND FORTRAN Reference Manual
 ND COBOL Reference Manual
 ND PLANC Reference Manual
 NORD PL User's Guide
 MAC User's Guide

2. BUILDING THE LOAD FILE

When running BUILD-LUAP:MODE, the GPM program may give the following error messages:

UNMATCHED >.
 PROBABLY MACHINE ERROR.

Usually caused by a missing semicolon in the source-file (not machine error!).

NON-DIGIT IN NUMBER.

A numeric parameter contains non-numeric characters.

UNDEFINED NAME.

Unrecognised macro-name encountered. Usually due to incorrect spelling of the macro names LOAD-SEGMENT, LOAD-500SEGMENT, ADD-APPL, ADD-500APPL, ADD-UNIT, ADD-COB-SUBROUTINES and ADD-500UNIT.

There are more GPM error-messages, but they are not likely to appear while building the load-file. However, GPM may produce a load file that the RT-loader does not accept. The errors may then be of three kinds:

- 1) Incorrect spelling of the macro parameters.
- 2) Missing parameter in a macro call. The string NIL is then substituted for the missing parameter in the load-file.
- 3) Superfluous character(s) in the macro file. The same characters are copied to the load-file. Occurs, for instance, where a macro call is terminated by two semicolons instead of one.

3. LOAD TIME ERRORS

Error messages from the real-time loader are described in the manual

SINTRAN III Real Time Loader.

and messages from the ND-500 Loader are described in the manual

ND-500 Loader/Monitor

In addition, the TPS load program LOTAB may give the following error messages:

APPL. MISSING, CHECK LOAD LISTING

The application has not been loaded for some reason or the program name declared in the program is not found in TPS-TABLES. Check the load listing for an error message from the RT loader.

ERROR IN OBJECT CODE

The application program being loaded does not start with the standard FORTRAN or COBOL entry point coding

TOO LARGE WORKING STORAGE FOR COBOL APPL.

The total size of the data area (or working storage for the main program) exceeds the fixed maximum.

ERROR IN OPENING TCF FILE

TCF file (checkpoint file) can not be opened. If already open, close it and try again.

WRITE ERROR ON TCF FILE

Error return from file system when attempting to write on the TCF file.

APPLICATION SEGMENT SIZE EXCEEDED

The available space for application programs is exceeded.

These error messages may also appear when loading ND-500 applications:

ND500 DESCRIPTION FILE ACCESS ERROR

Error return from filesystem when attempting to access the file:(RT)DESCRIPTION-FILE:DESC

DOMAIN NAME NOT FOUND IN DESCR. FILE

Some error occurred in the LINKAGE-LOADER when running the mode-file: LOAD-USER-APPL:MODE. Check the load listing.

SEGMENT NAME NOT FOUND IN DESCR. FILE

Some error occurred in the LINKAGE-LOADER when running the mode-file: LOAD-USER-APPL:MODE. Check the load listing.

ND500 LINK FILE ACCESS ERROR

Error return from filesystem when attempting to access the file: seg-name:LINK. Check the load listing.

ND500-APPL.MISSING, CHECK LOAD LISTING

Same as "APPL.MISSING, CHECK LOAD LISTING"

ND100-APPL. EXISTS AS RT-PROGRAM (APXXX)

An RT-program with name APXXX exists. This is fatal for the TPS-load-procedure. Use the command @LIST-RT-PROGRAM to detect such programs, then use the RT-loader command DELETE-PROGRAM in order to remove such program(s).

4. RUN TIME ERRORS

Run time error messages can come from several sources:

- The application program itself may write error messages on the user terminal using normal output statements. It may send error messages to the TPS operator using the write-message TSR. The contents of these error messages are determined by the programmer.
- The application program may call ERMON to write the standard error message on the SINTRAN error device (usually terminal 1):

hh.mm.ss ERROR nn IN rr AT ll;

USER ERROR. SUBERROR:ss

where

hh.mm.ss	time when the message is printed
nn	user error number
rr	TPT identification
ll	address of error in application program
ss	user suberror number

- The application program may call ERMSG or QERMS after an error return from a SINTRAN routine. A SINTRAN error message will be written on the SINTRAN error device. These messages are described in the SINTRAN III User's Guide.
- The FORTRAN or COBOL runtime systems may write error messages. FORTRAN messages will be written on the SINTRAN error device (usually terminal 1), while COBOL messages will always be written on terminal 1. These error messages are described in the manuals

ND FORTRAN Reference Manual
 ND COBOL Reference Manual
 ND-PLANC Reference Manual

APPENDIX D

TPS SEGMENT STRUCTURE IN ND-100

In the drawing, the dark line shows how the 64K address space is used when application programs are running. The application program goes from 0 to P2, the TPT plus user data area from P2 to P3, the reentrant segment from P3 to P1.

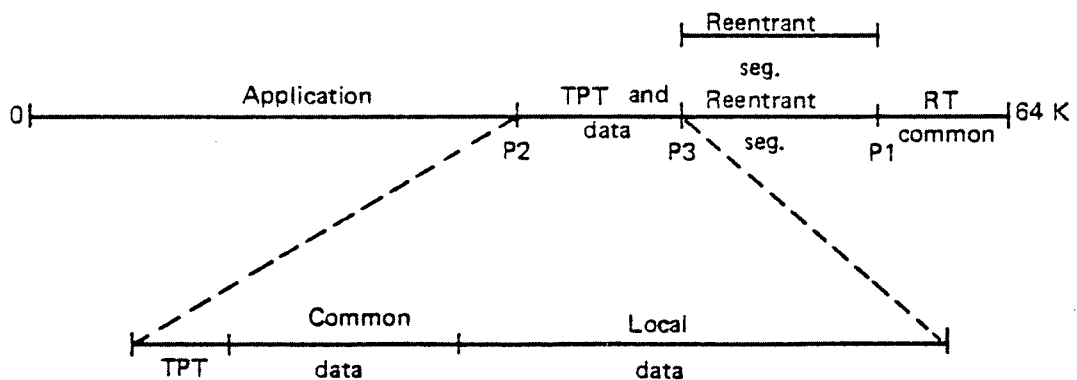
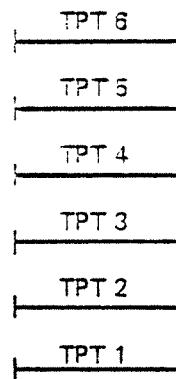
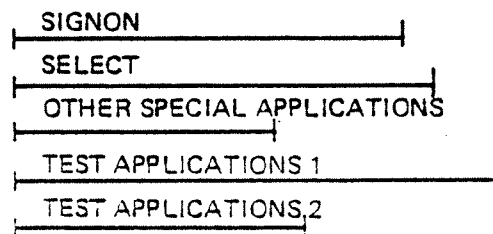
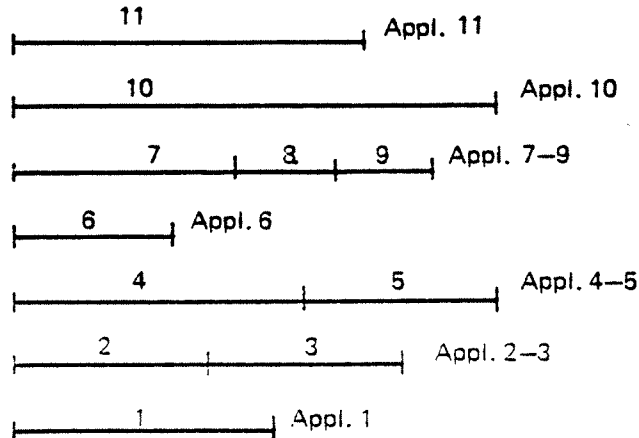
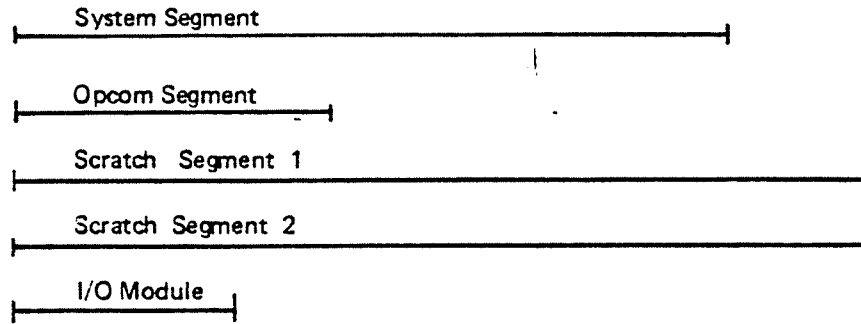
The other lines show the segments used for:

- reentrant segments from 1 to 9 ($R=1$ in example)
- TPT segments ($T = 6$ in example)
- special and test application segments ($S = 5$ in example)
- user application segments ($U = 7$ in example)
- system segment (1), OPCOM segment (1) and scratch segments (2)

Total number of segments:

$$4 + R + T + S + U \text{ (23 in example)}$$

In addition, a TPT segment is shown in more detail.



APPENDIX E**MONITOR CALLS AND LIBRARY CALLS****1. MONITOR CALL ROUTINES ALLOWED IN TPS PROGRAMS:**

NUMBER	NAME	NUMBER	NAME
0	LEAVE	76	SETBS
1	INCH	77	SETBL
2	OUTCH	104	HOLD
3	ECHOM	111	UPDAT
4	BRKM	112	CLADJ
11	TIME	113	CLOCK
26	LASTC		
42	OPEN	117	RFILE
43	CLOSE	120	WFILE
62	RMAX	122	RESRV
64	ERMSG	123	RELES
65	QERMS	124	PRSRV
66	ISIZE	125	PRLS
67	OSIZE	134	RTEXT
73	SMAX	142	ERMON
74	SETBT	162	OUTST
75	REABT		

2. LIBRARY CALL ROUTINES

All standard library routines are allowed in TPS programs with the exception of the following DOUBLE word and COMPLEX routines (these can be obtained upon special request):

DINT	CSQRT
DFLOAT	CEXP
DNINT	CLOG
DMOD	CSIN
DDIM	CCOS
DMAX1	CABS
DMIN1	
DSQRT	
DEXP	
DLOG	
DLOG10	
DSIN	
DCOS	
DATAN	
DTAN2	
DABS	

APPENDIX F

SCREEN—HANDLING CALLS

FOCUS-CALLS

FCINITE (init FOCUS)

CALL FCINITE(initiation-arrayt,private-form-buffer,status)

FCDECFF (declare form file)

CALL FCDECFF(form-file-name,status)

FCDECFN (declare form name)

CALL FCDECFN(form-name,mode,status)

FCDECRC (declare record)

CALL FCDECRC(number-of-field-names,field-names,
first-occurrence-number, last-occurrence-number,status)

FCEREC (edit record)

CALL FCEREC(data-record,edit-mode,status)

FCWREC (write record)

CALL FCWREC(data-record,status)

FCEFLD (edit one field)

CALL FCEFLD(field-name,occurrence-number,data-element,
data-element-length,edit-mode,status)

FCWFLD (write one field)

CALL FCWFLD(field-name,occurrence-number,data-element,
data-element-length,status)

FCEDSTA (get edit status)

CALL FCEDSTA(number-of-fields-edited,record-status-array,
terminating-code,status)

FCCLREC (clear data record)

CALL FCCLREC(data-record,status)

FCCLFDS (clear fields)

CALL FCCLFDS(mode,status)

FCESFLD (set edit start field)

CALL FCESFLD(field-name,occurence-number,status)

FCOPEN (open file)

CALL FCOPEN(file-name,access-code,file-number,status)

FCCLOSE (close file)

CALL FCCLOSE(file-number,status)

FCPRDOC (print document on file)

CALL FCPRDOC(data-record,file-number,status)

FCZMSGE (send message)

CALL FCZMSGE(message,status)

FCGMSGE (get message)

CALL FCGMSGE(leading-text,message,status)

FCCLSCR (clear rectangular area on screen)

CALL FCCLSCR(from-line,from-column,to-line,to-column,status)

FCWTXT (write text)

CALL FCWTXT(line,column,text,leng,status)

FCRTXT (read text)

CALL FCRTXT(line,column,text,leng,status)

NSHS-CALLS IN ND100*GTPIC and CGTPIC (get picture)*

CALL GTPIC (picture-file-name, number-of-pictures, picture-name-string,
picture-number-array, status)

CALL 'CGTPIC' USING picture-file-name, number-of-pictures,
picture-name-string, picture-number-array, status.

RMPIC (remove picture)

CALL RMPIC (number-of-pictures, picture-number-array, status)

GFTDN (get field numbers)

CALL GFTDN (picture-number, number-of-field-indicators,
field-indicator-array, field-number-array, number-of-fields, status)

WRPTD (write picture to display)

CALL WRPTD (picture-number, status)

WRPTF and CWRPTF (write-picture-to-file)

CALL WRPTF (file-number, flag, code, picture-number, number-of-fields,
field-number-array, record, data-element-index-array, status)

CALL 'CWRPTF' USING file-name, flag, code, picture-number,
number-of-fields, field-number-array, record, data-element-index-array,
status.

WMSGF and CWMSGF (write-message)

CALL WMSGF (text)

CALL 'CWMSGF' USING text.

WFLDS (write fields to VDU)

CALL WFLDS (code, picture-number, number-of-fields,
field-number-array, record, data-element-index-array, status)

RFLDS (read fields)

CALL RFLDS (code, picture-number, number-of-fields, field-number-array,
record, data-element-index-array, number-of-fields-read,
terminating-character, status)

CFLDS (clear fields)

CALL CFLDS (picture-number, number-of-fields, field-number-array,
status)

CLSCR (clear-screen)

CALL CLSCR (code, first-line, last-line, start-or-end-position, status)

CLBUF (clear-buffer)

CALL CLBUF (picture-number, number-of-fields, field-number-array,
record, data-element-index-array, status)

ZREAD/RREAD,ZLOCK/RLOCK,ZMUST/RMUST (set/remove — XXXX)

CALL Z/R XXXX (picture-number, number-of-fields, start-index,
field-number-array, status)

APPENDIX G

SIBAS CALLS

SUMMARY OF THE DML-STATEMENTS

OPEN-DATA-BASE

CALL SOPDB (mode, data base name, password, status)

CLOSE-DATA-BASE

CALL SCLDB (data base name, status)

READY-REALM

CALL SRRLM (no.-of-realms, realm-names, usage-modes,
protection-modes, status)

FINISH-REALM

CALL SFRLM (no. of realms, realm names, status)

FIND-USING-KEY

CALL SFTCH (realm name, key name, key value, status, key length)

FIND-FIRST-BETWEEN-LIMITS-USING-KEY

CALL SFEBL (realm name, key name, low limit, high limit, status, key
length)

FIND-LAST-BETWEEN-LIMITS-USING KEY

CALL SFLBL (real name, key name, low limit, high limit, status, key
length)

FIND-FIRST-IN-REALM

CALL SRFIR (realm name, status)

FIND-FIRST-IN-SET

CALL SRFSM (temporary data base key, set name, status)

FIND-LAST-IN-SET

CALL SRLSM (temporary data base key, set name, status)

FIND-PRIOR-IN-SET

CALL SRPSM (temporary data base key, set name, status)

FIND-NEXT-IN-SET

CALL SRNSM (temporary data base key, set name, status)

FIND-NEXT-IN-SEARCH-REGION

CALL SRNIS (temporary data base key, temporary search region indicator, status)

FIND-PRIOR-IN-SEARCH-REGION

CALL SRPIS (temporary data base key, temporary search region indicator, status)

FIND-SET-OWNER

CALL SRSOW (temporary data base key, set name, status)

GET

CALL SGET (temporary data base key, no. of items, item list, item values, status)

GETN

CALL SGETN (temporary data base key, temporary search region indicator, no. wanted, no. of items, item list, item values, no. found status)

GET-INDEXES

CALL SGIXN (temporary data base key, temporary search region indicator, no. wanted, item values, no. found, status)

MODIFY

CALL SMDFY (temporary data base key, no. of items, item list, item values, status, value length)

STORE

CALL STORE (realm name, no. of items, item list, item values, status, value length)

ERASE

CALL SRASE (temporary data base key, option code, status)

CONNECT

CALL SCONN (temporary data base key 1, set name, status)

CONNECT-BEFORE

CALL SCONB (temporary data base key 1, temporary data base key 2, set name, status)

CONNECT AFTER

CALL SCONA (temporary data base key 1, temporary data base key 2, set name, status)

DISCONNECT

CALL SDCON (temporary data base key, set name, status)

INSERT

CALL SINSR (temporary data base key, key name, status)

REMOVE

CALL SREMO (temporary data base key, key name, status)

REMEMBER

CALL SREMB (temporary id, option code, status)

FORGET

CALL SFORG (temporary id, option code, status)

LOCK

CALL SLOCK (temporary data base key, option code, status)

UNLOCK

CALL SUNLK (status)

CHANGE-PASSWORD

CALL SCHPW (new password, status)

ACCEPT

CALL SDBEC (set name, realm name 1, realm name 2, item name,
dmlstatement code, dbec)

ERASE-ELEMENT

CALL SEREL (temporary data base key, no. of items, item list, status)

ACCUMULATE INTEGER, FLOATING OR DOUBLE WORD

CALL ACCID/ACCFD/ACCDD (temporary data base key, no. of items, item
list, increments, new values, status)

CHANGE-THE-SIBAS-SYSTEM

CALL SETDV (system-no.)

EXECUTE-MACRO

CALL SEXMC (input, length of input, output, length of output, status)

UPDATE-DATA-BASE-IN-PLACE

CALL SUPLA (update ratio, trigger code, checkpoint id, status)

RESTRICTED SIBAS CALLS

CHANGING STATES

START
STOPS
SRUN
SPAUS
SRECO
SFINI
STREP
SPASS

LOGGING

INLOG
OFLOG/ONLOG
BSEQU/ESEQU

CHECKPOINT/ROLLBACK/REPROCESS

SCHPO/GCHPO
SROLL
SICON
SREPR

MISCELLANEOUS

RESIB/RELSI
SABOR
CHCOM
SIBIO
STRLG
RBLAN/SBLAN/ZTRB

APPENDIX H

TSR CALL FORMATS

A complete format description of all TSR calls in alphabetical order is given in this appendix. Examples are given in FORTRAN and COBOL.

If called from PLANC, all routines must be declared as ROUTINE STANDARD..... in IMPORT statement. PLANC-routines should call CGBRD and CWMSG instead of TGBRD and TWMSG, using type "BYTES" for the parameter <text>.

Note: When called from ND-500, all parameters in TSR-calls must have a word-length of 32-bits. Text strings are packed 4 bytes into each word.

Also in ND500, no "INTEGER" data element may exceed $2^{15}-1$ in absolute value, due to the data-range in ND100. If this range is exceeded, TSR-calls having a *return status*, will be given the value -10: INTEGER range overflow.

CGBRD

Get a broadcasted message, if there is any for this TPT (COBOL form).

CALL 'CGBRD' USING <text> <status>.

Exit Parameters:

<text>	A character string where the received text is to be placed. The record should have a length of 72 characters
<status>	=0, OK - text placed in array =-1, No broadcast message has arrived.

Rules:

- CGBRD must be used for character strings without descriptor words (i.e. COBOL and PLANC programs)
- The text may not exceed 72 characters; it will be terminated by an apostrophe and padded with blanks
- The broadcast may have come from an application program using the TBRDC TSR or from the TPS operator using the BROADCAST command or the MESSAGE-TO-UNIT command

Example:

COBOL: CALL 'CGBRD' USING MESSAGE-AREA STATUS-CODE.

PLANC: IMPORT (ROUTINE STANDARD VOID, VOID (BYTES, INTEGER WRITE) :
CGBRD)
CGBRD (TEXT, STATUS)

CWMSG

Write a message on the operator's console

CALL 'CWMSG' USING <text string>.

Entry Parameter:

<text string>	a character string containing the text to be written out.
---------------	---

Rules:

- CWMSG must be used for character strings without descriptor words (i.e. COBOL and PLANC programs)
- The text string may contain CR and LF. It must be terminated by a ' and may not exceed 256 bytes. Bit 7 (parity bit) in each byte must be 0.
- The message will be supplied with time, date and source identity

Example:

COBOL: CALL 'CWMSG' USING MESSAGE-TEXT.

PLANC: IMPORT (ROUTINE STANDARD VOID, VOID (BYTES): CWMSG)
CWMSG (TEXT)

TAAVA

Set an application available.

```
CALL TAAVA ( <application number>, <status> )
CALL 'TAAVA' USING <application number> <status>.
```

Entry Parameter:

<application number> The TPS application number

Exit Parameter:

<status> =0, OK
 =-1, illegal application number

Rules:

- TAAVA indicates the same action as the corresponding operator command (SET-AVAILABLE)
- Callable from special applications only

Examples:

```
CALL TAAVA (12, ISTAT)
CALL 'TAAVA' USING APPL-NO STATUS-CODE.
```

TABND

Stop the TPS system immediately (abnormal end).

```
CALL TABND ( <scope> )
CALL 'TABND' USING <scope> .
```

Entry parameter

<scope>	=0 if global abend (all TCMs)
	=1 if local to current TCM

Rules:

- TABND indicates the same action as the corresponding operator command (ABEND—TPS)
- Callable from special applications only
- No return to the application after calling this TSR
- Only global action allowed at present

Examples:

```
CALL TABND (0)
CALL 'TABND' USING ZERO.
```

TABST

Read the abend status of the current task.

CALL TABST (<record>)
CALL 'TABST' USING <record>.

Entry Parameters:

None

Exit Parameters:

record(1)	Previous appl. number
record(2)	Previous appl. status (Not active =0, active =1, active in a TSR-call =2)
record(3)	Data base activity for previous application (none =0, DB opened =1, DB updated =2)
record(4)	Current TPT no (2-63 if normal processing - special applications TPOPN, CHECK, TPCLO, ROLBK, RCOVR will always be executed on TPT no 1).
record(5)	ABEND activated by TPS (=0), previous appl. (=1), or by run-time system (=2).
record(6)	<p>ABEND information:</p> <p>If abended by TPS: (record(5) = 0)</p> <p>0 = abended by operator. 1 = impossible to activate application. 2 = illegal use of TSRs. 3 = subroutine not loaded. 4 = application timeout. 5 = internal TPS error 6 = operator timeout. 7 = attempt to restore 500-application 8 = error from 500-monitor</p> <p>If abended by an application: (record(5) = 1) record(6) contains the parameter reasonused by previous application when calling the TSR TSTOP.</p> <p>If abended by run-time system: (record(5) = 2) Standard SINTRAN error code.</p>
record(7)	ABEND—location:

Latest link register.

record(8)	First application activated for this TPT
record(9)	Termination strategy (see TSTST)
record(10)	Termination application
record(11)	Abend strategy (see TSAST)
record(12)	Abend application
record(13)	Restart strategy (see TSRST)
record(14)	Restart application
record (15)	Close strategy (see TSCST)
record(16-30)	Unused

Rules:

- Callable from ABEND only
- Words 1-3 refer to the *previous* application, not the calling application
- Words 5-7 are 0 if ABEND has not been activated

Examples

```
DIMENSION IREC (30)
CALL TABST (IREC)
```

```
CALL 'TABST' USING ABEND—RECORD.
```


TACTV

Activate a concurrent task.

```
CALL TACTV ( <application number>, <record>, <size>, <status>
CALL 'TACTV' USING <application number> <record> <size>
<status>.
```

Entry Parameters

<application number>	The TPS application number
<record>	Data array/record that is to be transferred to the new, activated task. This record is treated as a contiguous string of bytes. Be aware of ND-100/ND-500 difference in word-length.
<size>	Size of <record> in bytes. Size may not exceed 2000 bytes (decimal).

Exit Parameters:

<status>	=0, OK -task activated. =-1, Parameter error (ill. appl. no/record size too large) =-2, No TPT available at present. Another attempt may be performed after an appropriate pause. =-3, The application is unavailable
----------	--

Rules:

- Up to 2000 bytes of data can be transferred from the activating to the activated task. The data will be placed at the beginning of the task common data area
- The new task will be given a TPT from the current TCM
- If no TPT is available, an error code is returned

Examples:

```
DIMENSION IDATA(5)
CALL TACTV (52,IDATA,10,ISTAT)

CALL 'TACTV' USING APPL=52 DATA=REC TEN STAT=CODE.
```

TAPST

Read the status of an application.

CALL TAPST (<application number>, <record>, <status>)
CALL 'TAPST' USING <application number> <record> <status>.

Entry Parameters:

<application number> The TPS application number.

Exit Parameters:

record(1)	Application state: 2 = ND-500 application ready 1 = ND-100 application ready 0 = Not available.
record(2)	If ND-100, application language 1 = COBOL 0 = FORTRAN/PLANC If ND-500, 0
record(3)	If ND-100, start address If ND-500, Left byte: ND-500 segment number Right byte: ND-500 appl.no. in current segment.
record(4)	If ND-100, application segment number If ND-500, ND-500 domain number.
record(5)	Application checkpoint weight.
record(6)	Application SINTRAN priority and ND500 priority.
record(7)	Current number of TPTs active on this application.
record(8)	Size of maximum task common
record(9)	ND100: Reentrant segment number used by this application. ND500: = 0
record(10-30)	Unused
<status>	=0. OK. =-1. Parameter error (ill.application number)

Rules:

- If the application does not exist, record(1) will be 0.
- MAC/NPL application will be given as FORTRAN because they must be FORTRAN—compatible.

Examples:

```
DIMENSION IREC(30)  
CALL TAPST (20, IREC, ISTAT)
```

```
01 APPL-RECORD.  
05 RECORD COMP OCCURS 30.  
CALL 'TAPST' USING 20 APPL—RECORD.
```

TASET

Set the absolute execution time for an application to be started at some future time.

```
CALL TASET (<module>, <application number>, <param 1>, <param
2>, <time>, <status>)
```

```
CALL 'TASET' USING <module> <application number> <param 1>
<param 2> <time> <status>.
```

Entry Parameters:

<module>	The module number of a TCM
<application number>	The TPS application number
<param 1> <param 2>	Decimal integer parameters that are placed at the beginning of the task common data area
<time>	Data array containing the absolute execution time as 6 decimal integers (second, minute, hour, day, month, year)

Exit Parameters:

<status>	= 0, OK — execution time set
	= -4, No time queue element available
	= -5, Parameter error

Rules:

- If no TPT is available when the task is to be started, an error message will be written on the TPS operator console
- The time resolution is 5 seconds

Examples:

```
DIMENSION ITIME(6)
CALL TASET (33,2,9,5, ITIME, ISTAT)
```

```
CALL 'TASET' USING TCM1 APPL-2 TERM-9 TYPE-5 ABSOLUTE-TIME
STATUS-CODE.
```

TAUNA

Set an application unavailable.

CALL TAUNA (<application number>, <status>)
CALL 'TAUNA' USING <application number> <status>.

Entry Parameter:

<application number> The TPS application number

Exit Parameter:

<status> =0, OK
 = -1, illegal application number

Rules:

- TAUNA indicates the same action as the corresponding operator command (SET-UNAVAILABLE)
- Callable from special applications only

CALL TAUNA (15, ISTAT)
CALL 'TAUNA' USING APPL-FIVE STATUS-CODE.

TBRDC

Broadcast a message to terminals connected to an IOM or TPTs controlled by a TCM.

CALL TBRDC (<module> , <sub-address> , <text> , <units> , <status>)
 CALL TBRDC' USING <module> <sub-address> <text> <units>
 <status> .

Entry Parameters:

<module>	As array/record identifying the IOM or TCM module to which the broadcast should be sent. If module (1)=0, the IOM to which this application is connected will receive the broadcast, otherwise the array/record should be prepared with these elements:
module (1)	Address type. These types are allowed: Type 1: Address is a number which identifies the module within TPS. Type 2: Address is a string of alphanumeric characters which identifies the module by its name.
module (2)	Size of address in bytes.
module (3-n)	Actual address of module.
<sub-address>	The terminal or TPT to which the broadcast should be sent. The construction of this parameter is identical to that of <module> , except that it identifies a unit within the environment of a module - only applicable if the message is to be sent to one specific unit only.
sub-address(1)	Address type. These types are allowed: Type 1: Address is a number which identifies the unit within the module. Type 2: Address is a string of alphanumeric characters which identifies the unit by its name. Type 3: Address is in any format (alphanum./integer/comp./BCD) which is relevant for the addressing of units in the given environment. This address type is denoted as the "native" address type.

sub-address(2)	Size of address in bytes.
sub-address(3-n)	Actual address of unit.
<text >	An array/record with the text to be written. The text should not exceed 72 characters, and should be terminated by an apostrophe (').
<units>	=0, means broadcast to all units connected to this module. =1, means broadcast to all active units connected to this module. =2, means that the message should be sent to a specific unit within the module, as specified in <sub-address>.

Exit Parameters:

<status >	=0, OK - Text written as specified. =-1, Parameter error - nothing written.
-----------	--

Rules:

- The message is written on the terminals on the broadcast line (usually the bottom line)
- Messages sent to TPTs can be read by the application program with the TGBRD/CGBRD TSR
- The message may not be more than 72 characters long and should be terminated by a apostrophe
- All texts should be defined as arrays or Hollerith strings, not character strings, in FORTRAN

Examples

```
CALL TBRDC (MODULE,0,ITEXT,0,ISTAT)
```

```
CALL 'TBRDC' USING MODULE SUB-ADDRESS TEXT—STRING TWO  
STATUS— CODE.
```

TBSEQ

Marks the beginning of a critical sequence according to SIBAS.

CALL TBSEQ

CALL 'TBSEQ'.

Parameters

None

Rules

- This TSR should be used with care since the critical sequence facility is used by TPS itself.

TCHCK

Take a synchronised checkpoint of the TPS system.

CALL TCHCK (<scope>)

CALL 'TCHCK' USING <scope>.

Entry Parameter:

<scope>	=0 if global checkpoint (all TCMs) =1 if local to current TCM
---------	--

Rules:

- TCHCK initiates the same action as the corresponding operator command (CHECKPOINT—TPS)
- Only global action allowed at present

Examples:

CALL TCHCK (0)

CALL 'TCHCK' USING ZERO.

TCLOS

Close the TPS system in a controlled manner (normal end).

CALL TCLOS (<scope>)
CALL 'TCLOS' USING <scope>.

Entry Parameter:

<scope>	=0 if global close (all TCMs)
	=1 if local to current TCM

Rules:

- TCLOS initiates the same action as the corresponding operator command (CLOSE—TPS)
- Callable from special applications only
- Only global action allowed at present

Examples:

CALL TCLOS (0)
CALL 'TCLOS' USING ZERO.

TCONF

Get the values of certain configuration parameters.

CALL TCONF (<record>)
CALL 'TCONF' USING <record>.

Exit Parameters:

record(1)	Number of TPTs belonging to the current TCM
record(2)	Number of applications in this TPS system
record(3)	Device number of operator console
record(4)	Application time-out in seconds
record(5)	Operator time out in seconds
record(6)	TCM number for this TPT
record(7-30)	Unused

Rules:

- Callable from special applications only.

Examples:

```
DIMENSION IREC(30)
CALL TCONF (IREC)

CALL 'TCONF' USING CONF-RECORD.
```

TCONT

Continue normal TPS operation.

CALL TCONT (<scope>)
CALL 'TCONT' USING <scope>

Entry Parameter:

<scope> =0 if global continue (all TCMs)
 =1 if local to current TCM

Rules:

- TCONT initiates the same action as the corresponding operator command (CONTINUE—TPS)
- Callable from special applications only
- Only global action allowed at present

Examples:

CALL TCONT (0)

CALL 'TCONT' USING ZERO.

TDCNT

Remove (disconnect) an application from the time queue and the interval table.

CALL TDCNT (<module>, <application number>, <status>).
CALL 'TDCNT' USING <module> <application number> <status>.

Entry Parameters:

<module>	The module number of a TCM
<application number>	The TPS application number

Exit Parameters:

<status>	= 0, OK, application disconnected
	= -5, Parameter error

Examples:

CALL TDCNT (32, 25, ISTAT)
CALL 'TDCNT' USING TCMO APPL-25 STATUS-CODE.

TESEQ

Makes the end of a critical sequence according to SIBAS.

CALL TESEQ.
CALL 'TESEQ'.

Parameters:

None.

Rules:

- This TSR should be used with care since the critical sequence facility is used by TPS itself.

TGBRD

Get a broadcast message, if there is any for this TPT.

CALL TGBRD (<text>, <status>)

Exit Parameters:

<text>	A string of the type CHARACTER where the received text is to be placed. The record should have a length of 72 characters
<status>	=0, OK - text placed in array =-1, No broadcast message has arrived

Rules:

- TGBRD must be used for character strings with descriptor words (i.e. FORTRAN character strings)
- The text may not exceed 72 characters; it will be terminated by an apostrophe and padded with blanks
- The broadcast may have come from an application program using the TBRDC TSR or from the TPS operator using the BROADCAST command or the MESSAGE-TO-UNIT command
- Should not be called from PLANC (use CGBRD).

Example:

```
CHAR BTEXT * 72
CALL TGBRD (BTEXT, ISTAT)
```

THALT

Halt the TPS system temporarily.

CALL THALT (<scope>)
CALL 'THALT' USING <scope>.

Entry Parameter:

<scope>	=0 if global halt (all TCMs) =1 if local to current TCM
---------	--

Rules:

- THALT initiates the same action as the corresponding operator command (HALT—TPS)
- Callable from special applications only
- Only global action allowed at present

Examples:

CALL THALT (0)
CALL 'THALT' USING ZERO.

THSYN

Do not allow a synchronised checkpoint to be taken until the next TTRAN or TTSYN call

CALL THSYN
CALL 'THSYN'.

Parameters:

None

Rules:

- If a checkpoint message comes, the checkpoint will not be taken until the application program calls TTRAN or TTSYN. Other TPS modules will however be frozen in the meantime.

TINTV

Set the execution interval for a periodic application. If this application is activated once, it will continue periodically.

CALL TINTV (<module>, <application number>, <param 1>, <param 2>, <interval>, <status>).

CALL 'TINTV' USING <module> <application number> <param 1> <param 2> <interval> <status>.

Entry Parameters:

<module>	The module number of a TCM
<application program>	The TPS application number
<param 1> <param 2>	Decimal integer parameters that are placed at the beginning of the task common data area
<interval>	Data array containing the execution interval as 4 decimal integers (seconds, minutes, hours, days)

Exit Parameters:

<status>	= 0, OK — interval set
	= -4, No interval table element available
	= -5, Parameter error

Rules

- TINTV will not itself start periodic execution of the specified application program. This must be done by some other means.
- A new interval starts at each time of activation.
- If the application already has an execution interval, the specified interval will replace the old one.
- If no TPT is available when the task is to be started, an error message will be written on the TPS operator console.
- The time resolution is 5 seconds.

Examples:

```
DIMENSION INTVL (4)
CALL TINTV (32, 14, 0, 0, INTVL, ISTAT)
```

```
CALL 'TINTV' USING TCMO APPL-14 ZERO ZERO INTERVAL STATUS-CODE.
```

TMISC

Read the miscellaneous TPT information.

CALL TMISC (<record>)

CALL 'TMISC' USING <record>

Exit parameters:

record (1)	RT description address of calling TPT.
record (2)	Type of last message
record (3)	Number of TSWAP's since TPT was allocated.
record (4)	Operator timeout.
record (5)	Application timeout.
record (6)	Packet size.

TPASZ

Set the packet size for session data.

CALL TPASZ (<packet size>, <status>).
CALL 'TPASZ' USING <packet size> <status>.

Entry Parameter:

<packet size>	A decimal integer containing the packet size in bytes
---------------	---

Exit Parameter:

<status>	0, OK -1, error
----------	--------------------

Rules:

- Maximum permitted packet size = 2047
- Default packet size is specified at system generation
- The value specified with TPASZ is used only for current transaction

Examples

CALL TPASZ (512, ISTAT)

CALL 'TPASZ' USING SIZE—1000 STATUS—CODE.

TR5MG

Get error message from the ND-500-monitor

```
CALL TR5MG (<record>,<size> )
CALL 'TR5MG' USING <record>  <size>
```

Entry Parameter

None

Exit Parameters:

<record>	An array where the message will be placed. The record must be able to hold a minimum of 200 characters.
<size>	Indicates number of characters in the message.

Rules:

- Callable from special application ABEND only, and only by "abend cause" = 8 (error from 500-monitor) when abended by TPS.

Examples:

```
DIMENSION IREC(100)
CALL TR5MG(IREC,ILEN)

CALL 'TR5MG' USING TEXT-STRING LENGTH
```

TRMSG

Read a message from session partner.

```
CALL TRMSG ( <record > , <size > , <more > , <status > )
CALL 'TRMSG' USING <record > <size > <more > <status>.
```

Entry Parameters

<size > Size of <record > in bytes.

Exit Parameters:

<record >	Data will be placed in this data area as it arrives. No formatting is performed by TPS, the record is treated as a contiguous string of bytes. Be aware of ND-100/ND-500 differences in word-length.
<size >	Indicates actual size of received message. Maximum size is 2047 bytes. If over-flow, <size > is unchanged.
<more >	Indicates that session partner has more to send if 1, that you are free to send data if 0.
<status >	=0, OK - <record > contains data. =-1, Session broken. =-2, <record > too small for the message (overflow). =-3, Direction = output (Input not allowed).

Rules:

- The message is received exactly as it was sent from the IOM. No formatting or editing is performed
- If no message has been sent, the program will wait in TRMSG until a message arrives

Examples:

```
CALL TRMSG (IREC, ISIZE, MORE, ISTAT)
```

```
CALL 'TRMSG' USING MESSAGE SIZE MORE STATUS—CODE.
```

TROLS

Roll the TPS system back to the last synchronised checkpoint.

CALL TROLS (<scope>)
CALL 'TROLS' USING <scope>.

Entry Parameter:

<scope> =0 if global rollback (all TCMs)
 =1 if local to current TCM

Rules:

- TROLS initiates the same action as the corresponding operator command (ROLLBACK—TPS)
- Callable from special applications only
- No return to the application after calling this TSR
- Only global action allowed at present

Examples:

CALL TROLS (0)
CALL 'TROLS' USING ZERO.

TROLT

Roll the TPS system back to the last transaction checkpoints (recovery).

CALL TROLT (<scope>)
CALL 'TROLT' USING <scope>.

Entry Parameter:

<scope> =0 if global recovery (all TCMs)
 =1 if local to current TCM

Rules:

- TROLT initiates the same action as the corresponding operator command (RECOVER—TPS)
- Callable from special applications only
- No return to the application after calling this TSR
- Only global action allowed at present

Examples:

CALL TROLT (0)

CALL 'TROLT' USING ZERO.

TRRST

Read the restart status of the current task.

CALL TRRST (<record>)
CALL 'TRRST' USING <record>.

Entry Parameters:

None

Exit Parameters:

record(1)	Previous appl. number (at latest valid checkpoint).
record(2)	Previous appl. status (not active =0, active =1, active in a TSR-call =2)
record(3)	Database activity for previous application (none =0, DB opened =1, DB updated =2)
record(4)	Latest valid checkpoint was a synchronised one (=0) or a transaction one (=1)
record(5-11)	SINTRAN time array - indicating time for latest valid checkpoint.
record(12)	Current task no (2-63 if normal processing - special applications TPOPN, CHECK, TPCLO, ROLBK, RCOVR will always be executed on task no 1).
record(13)	Restart strategy (<i>see TSRST</i>).
record(14)	Restart application.
record(15)	Termination strategy (<i>see TSTST</i>).
record(16)	Termination application.
record(17)	First application activated for this TPT.
record(18)	Close strategy (see TSCST)
record(19-30)	Unused.

Rules:

- Callable from the RESTART special application only
- Words 1-3 refer to the *previous* application, not the calling application

Examples:

DIMENSION IREC (30)
CALL TRRST (IREC)

01 RESTART-RECORD.
05 RECORD COMP OCCURS 30.
CALL 'TRRST' USING RESTART—RECORD.

TRSES

Restore a broken session if possible.

```
CALL TRSES ( <status> )
CALL 'TRSES' USING <status>.
```

Exit Parameters:

<status>	=0, Ok - session re-established.
	=-1, TPS closed.
	=-2, Module closed.
	=-3, Unit temporarily not available.
	=-4, Unit permanently not available.
	=-5, Parameter error.
	=-6, Session terminated.
	=-7, Not called from RESTART.

Rules:

— Callable from the RESTART application only.

Examples:

```
CALL TRSES (ISTAT)
IF (ISTAT NE 0) GO TO no-session

CALL 'TRSES' USING STATUS—CODE.
IF STATUS—CODE NOT= 0 GO TO NO—SESSION.
```


TRSTO

Restore the ND100 user application program and restart it at checkpoint.

CALL TRSTO
CALL 'TRSTO'.

Parameters:

None

Rules:

- Callable from the RESTART special application only.
- Not callable from the ND-500 (not possible to restart a ND-500-application at a point inside the application). If it is called from an application running in the ND500, the ABEND-application will be activated with the information "Attempt restore ND500-application."

TSAST

Set the abend strategy for the TPT.

```
CALL TSAST ( <abend strategy>, <abend application> )
CALL 'TSAST' USING <abend strategy> <abend application>.
```

Entry Parameters:

<abend strategy>	<p>The abend strategies used by the standard version of the ABEND special application are:</p> <ol style="list-style-type: none"> 1. switch to SIGNOFF 2. send an error message to the terminal operator, switch to SIGNOFF 3. dump the data area for the TPT on the printer, switch to SIGNOFF 4. switch to <abend application> 5. Halt the TPS system
<abend application>	<p>The TPS application number of the program to be started if strategy 4</p>

Rules:

- The abend strategy determines the action taken by the ABEND special application in a transaction abnormal end situation.
- The default value when a task is started is 1.
- For all strategies, an error message will be sent to the TPS operator console.
- If <abend strategy> = 0, it will not be changed; if <abend application> = 0, it will not be changed.
- No validation check of <abend application> is performed by TSAST. If it is an illegal application, SIGNOFF will be activated instead at abend.

Examples:

```
CALL TSAST(3,0)
```

```
CALL 'TSAST' USING FOUR USER-ABEND-NO.
```

TSCIN

Set checkpoint interval for synchronized checkpoints.

CALL TSCIN (<time in minutes>)
CALL 'TSCIN' <time in minutes> .

Entry Parameter:

<time in minutes> Checkpoint interval =0 no synchronized checkpoints will be taken until the interval has been changed.

Examples:

CALL TSCIN (30)
CALL 'TSCIN' USING THIRTY.

TSCLO

Close a session.

CALL TSCLO (<status>)
CALL 'TSCLO' USING <status> .

Exit Parameters:

<status> =0, OK - session orderly closed
 = -1, Session already closed
 = -2, Session considered closed, but not confirmed by partner.

Rules:

- The connection between the session partners will be completely broken and cannot be restored.
- The session may be closed by either of the session partners.

Examples:

CALL TSCLO (ISTAT)

CALL 'TSCLO' USING STATUS—CODE.

TSCST

Set the close strategy for the TPT.

```
CALL TSCST (<close strategy>)
CALL 'TSCST' USING <close strategy>.
```

Entry Parameter:

<close strategy>	=0, normal termination
	=1, immediate termination

Rules:

- The close strategy determines the action to be taken by TPS when a close command is given.
- The default value when a task is started is 0.
- If a close command is pending, a close strategy of 1 will cause immediate termination.

Examples:

```
CALL TSCST(0)
CALL 'TSCST' USING ZERO.
```

TSEQU

Marks the end of a critical sequence and the beginning of another critical sequence according to SIBAS.

```
CALL TSEQU.
CALL 'TSEQU'.
```

Parameters:

None.

Rules:

- This TSR should be used with care since the critical sequence facility is used by TPS itself.

TSEST

Read the session status of the current task.

```
CALL TSEST ( <record > )
CALL 'TSEST' USING <record >.
```

Exit parameters:

record (1)	Current session state: =0, No session active. =1, Session request pending. =2, Session established. =3, Session terminate command pending.
record (2)	Current direction of session (=1: inbound, =0: out-bound)
record (3)	Total no. of input messages so far in this session.
record (4-10)	Time for latest input message.
record (11)	Total no. of output messages so far in this session.
record (12-18)	Time for latest output message.
record (19)	Session partner, module number.
record (20)	Session partner, unit number.
record (21-30)	Unused

Rules:

- Used mainly by the RESTART special application but also available to other application programs.

Examples:

```
DIMENSION ISES (30)
CALL TSEST (ISES)

CALL 'TSEST' USING SESSION—STATUS.
```

TSMMSG

Send a message to the session partner.

```
CALL TSMMSG ( <record > , <size > , <more > , <status > )
CALL 'TSMMSG' USING <record > <size > <more > <status > .
```

Entry parameters:

<record >	Array/Record to be transmitted to session partner. This record is treated as a contiguous string of bytes. Be aware of ND-100/ND-500 difference in word-length.
<size >	Size of <record> to be transmitted (in bytes). Maximum size is 2047 bytes.
<more >	Calling application should indicate whether more data will follow (1) or not (0) in a following call.

Exit Parameters:

<status >	=0, OK - successful transmission. =-1, Session broken. =-2, Parameter error (Too large <record >) =1, You have got a message.
-----------	---

Rules:

- The message is sent exactly as prepared by the application program - no formatting or editing is performed
- Return to the application program will be immediate without waiting for an answer

Examples:

```
CALL TSMMSG (ITEXT,ISIZE,1,ISTAT)
```

```
CALL 'TSMMSG' USING MESSAGE—TEXT MESSAGE—SIZE ONE
STATUS—CODE.
```

TSOPN

Open a session with a new session partner.

```
CALL TSOPN ( <module>, <sub-address>, <record>, <size>, <more>,
<status> )
CALL 'TSOPN' USING <module>    <sub-address>    <record>    <size>
<more> <status>.
```

Entry Parameters:

<module >	The IOM or TCM controlling the session partner, or the intersystem communication IOM if the session partner is in another TPS system. This parameter is an array (FORTRAN) or record (COBOL) with the following elements:
module (1)	Address type. These types are allowed: Type 1: Address is a number which identifies the module within TPS Type 2: Address is a string of alphanumeric characters which identifies the module by its name.*
module (2)	Size of address in bytes.
module (3-n)	Actual address of module.
<sub-address>	The device or application program to be session partner. The construction of this parameter is identical to that of <module> , except that it identifies a unit belonging to the module.
sub-address (1)	Address type. These types are allowed: Type 1: Address is the unit within the module. Type 2: Address is a string of alphanumeric characters which identifies the unit by its name.*
sub-address (2)	Size of address in bytes.
sub-address (3-n)	Actual address of unit.
<record>	Array/record to be transmitted to session partner. Maximum size is 2000 bytes (may be less in some cases, for example, with an X-25 permanent virtual channel). This record is treated as a contiguous string of bytes. Be aware of ND-100/ND-500 difference in word-length.

<size> Size of record in bytes.

<more> Indicates whether more data will follow (=1) or not (=0) in a following call.

 * not yet implemented.

Exit Parameters:

<status> =0, OK - session established.
 <0, Not successful.
 =-1, TPS closed, terminate your task as soon as possible.
 =-2, Module closed.
 =-3, Unit temporarily not available, try again.
 =-4, Unit permanently not available.
 =-5, Parameter error.
 =-6, Session already established.
 =-7, No TPT available

Rules:

- Only one session is allowed at a time
- A session request for a device controlled by an IOM will result in the allocation of the device to the requesting application program
- A session request for an application program will result in the allocation of a TPT controlled by the given TCM - this does not have to be the current TCM
- Up to 2000 bytes of data may be sent with TSOPN
- At present the session partner must be in the same TPS system. Sessions between two TPS systems will be implemented later

Examples:

```
DIMENSION IMOD (3), IUNIT (3)
CALL TSOPN (IMOD,IUNIT,0,0,0,ISTAT)
IF (ISTAT.NE.0) GO TO error
```

```
CALL 'TSOPN' USING MODULE SUB-ADDRESS DATA-REC
                      REC-SIZE MORE-BIT STATUS-CODE.
```


TSOPT

Set the operator timeout for the TPT.

CALL TSOPT (<time in minutes>)
CALL 'TSOPT' USING <time in minutes>.

Entry parameter:

<time in minutes>	Operator timeout time.
=0	The operator timeout is turned off.

Rules:

- The operator timeout could be changed depending on the current application.

Examples:

CALL TSOPT (20)
CALL 'TSOPT' USING TWENTY.

TSRST

Set the restart strategy for the TPT.

```
CALL TSRST ( <restart strategy>, <restart application> )
CALL 'TSRST' USING <restart strategy> <restart application>.
```

Entry Parameters:

<restart strategy>	<p>The restart strategies used by the standard version of the RESTART special application are:</p> <ol style="list-style-type: none"> 1. Restart from checkpoint 2. Switch to <restart application> 3. Automatic termination 4. The terminal operator chooses the restart action
<restart application>	<p>The TPS application number of the program to be started if strategy 2</p>

Rules:

- The restart strategy determines the action taken by the RESTART special application in a restart situation
- The default value of <restart strategy> when a task is started is 2; the default value of <restart application> is the first application that is activated (normally SIGNON)
- If <restart strategy> = 0, it will not be changed; if <restart application> = 0, it will not be changed
- No validation check of <restart application> is performed by TSRST. If it is an illegal application, ABEND will be activated instead of restart

Examples:

```
CALL TSRST(2, -1)
```

```
CALL 'TSRST' USING THREE.
```

TSTAT

Read the status of the current task.

```
CALL TSTAT ( <record> )
CALL 'TSTAT' USING <record> .
```

Exit Parameters:

record(1)	Current TPS state, 0 = ready, 1 = close requested.
record(2)	Current TCM number.
record(3)	Current TPT number.
record(4)	First application activated on current TPT.
record(5-11)	Time array with TPT allocation time.
record(12)	Number of TSWAPs on this TPT since allocation.
record(13)	Database activity (none = 0, DB opened = 1, DB updated = 2)
record(14)	Number of SIBAS calls since TPT allocation.
record(15)	Number of SIBAS calls since sync. checkpoint.
record(16)	Number of SIBAS calls since checkpoint.
record(17)	Number of SIBAS update calls since TPT allocation.
record(18)	Number of SIBAS update calls since sync. checkpoint.
record(19)	Number of SIBAS update calls since checkpoint.
record(20)	Message indicator (no msg = 0, msg/broadcast arrived, = 1)
record(21)	Termination strategy (see TSTST)
record(22)	Termination application.
record(23)	Previous application activated on this TPT.
record(24)	Close strategy (see TSCST)

record(25)	ND100: Not used. ND500: Terminal device number of Symbolic Debugger. (Debug mode only, else = 0.)
record(26-30)	Unused.

Examples:

```
DIMENSION IREC(30)  
CALL TSTAT(IREC)
```

```
CALL 'TSTAT' USING TASK-STATUS.
```

TSTOP

Terminate the transaction.

```
CALL TSTOP ( <stop code> )
CALL 'TSTOP' USING <stop code >.
```

Entry Parameters:

<stop code>

=0, Normal transaction termination. This has the same effect as STOP RUN (COBOL) or END (FORTRAN).

>0, ABEND is activated and may obtain the stop code from the TSR-routine TABST.

<0, ABEND is activated and a formatted printout is given according to the stop code.

—1, for NSHS errors; ITERM(7) is displayed.

—2, for SIBAS errors; names of realms, items and DBEC codes are displayed.

Rules:

- Normal termination will result in activation of the SIGNOFF special application which will carry out the action indicated by the termination strategy
- Abnormal termination will result in activation of the ABEND special application which will carry out the action indicated by the abend strategy
- The application program should release resources and close files before calling TSTOP as this is not always done automatically
- TSTOP(0) from special applications will result in complete termination and release of TPT

Examples:

```
CALL TSTOP (3)
```

```
CALL 'TSTOP' USING ABEND—CODE.
```

TSTST

Set the termination strategy for the TPT.

```
CALL TSTST ( <termination strategy>, <termination application> )
CALL 'TSTST' USING <termination strategy> <termination application>.
```

Entry Parameters:

<termination strategy> The termination strategies used by the standard version of the SIGNOFF special application are:

1. Complete termination and release of the TPT
2. Switch to SIGNON
3. Switch to SELECT
4. Switch to <termination application>

<termination application> The TPS application number of the program to be started if strategy 4

Rules:

- The termination strategy determines the action taken by the SIGNOFF special application when a transaction terminates
- The default value of <termination strategy> when a task is started is 1
- If <terminate strategy> = 0, it will not be changed; if <terminate application> = 0, it will not be changed
- No validation check of <termination application> is performed by TSTST. If it is an illegal application, there will be complete termination with release of the TPT by SIGNOFF
- The <termination application> must not itself terminate "normally" unless it has changed the termination strategy, since this will result in an endless loop. It may for example terminate by switching to SIGNON.

Examples:

```
CALL TSTST(2)
```

```
CALL 'TSTST' USING ONE ZERO.
```

TSWAP

Switch to a new application program.

```
CALL TSWAP ( <application number>, <status> )
CALL 'TSWAP' USING <application number> <status>.
```

Entry Parameters:

<application number> the TPS application number

Exit Parameters:

<status> if not successful switch of application, return to the
calling application is performed with cause in
status.

- 1 = illegal application number
- 2 = application not available

Rules:

- The new application will be started from the beginning. It will have access to the data in the task common data area for the transaction. Sessions will not be broken and resources will not be released when switching applications

Examples:

```
CALL TSWAP (APPL, ISTAT)
```

```
CALL 'TSWAP' USING NEXT-APPL STATUS-CODE.
```

TTERM

Terminate this task directly and completely.

CALL TTERM (<checkpoint>)
CALL 'TTERM' USING <checkpoint>.

Entry Parameter:

<checkpoint>	=0, do not take a transaction checkpoint
	=1, take a transaction checkpoint

Rules:

- termination is immediate and direct, i.e. SIGNOFF will not be activated
- termination is complete, i.e. the TPT will be freed
- a transaction checkpoint may be taken

Examples:

CALL TTERM(1)
CALL 'TTERM' USING ONE.

TTEXT

Send a text message to a terminal connected to an IOM or a TPT controlled by a TCM.

CALL TTEXT (<module>, <sub-address> <text>, <length>, <status>)

CALL 'TTEXT' USING <module>, <sub-address> <text>, <length>, <status>.

Entry Parameters:

<module>	<p>An array/record identifying the IOM or TCM module to which the broadcast should be sent.</p> <p>If module (1)=0, the IOM to which this application is connected will receive the broadcast, otherwise the array/record should be prepared with these elements:</p>
module(1)	<p>Address type. These types are allowed:</p> <p>Type 1: Address is a number which identifies the module within TPS.</p> <p>Type 2: Address is a string of alphanumeric characters which identifies the module by its name.</p>
module(2)	Size of address in bytes
module(3-n)	Actual address of module
<sub-address>	<p>The terminal or TPT to which the broadcast should be sent.</p> <p>The construction of this parameter is identical to that of <module>, except that it identifies a unit within the environment of a module - only applicable if the message is to be sent to one specific unit only.</p>
sub-address(1)	<p>Address type. These types are allowed:</p> <p>Type 1: Address is a number which identifies the unit within the module.</p> <p>Type 2: Address is a string of alphanumeric characters which identifies the unit by its name.</p> <p>Type 3: Address is in any format (alphanum./integer/comp./BCD) which is relevant for the addressing of units in the given environment. This address type is denoted as the "native" address type.</p>

sub-address(2)	Size of address in bytes.
sub-address(3-n)	Actual address of unit
< text >	An array/record with the text to be written.
< length >	A decimal integer specifying the message length in bytes

Exit Parameters:

< status >	= 0, OK — Text written as specified
	= -1, Parameter error — nothing written

Rules:

- The message is written on the terminals wherever the cursors happen to be positioned
- Messages sent to TPTs can be read by the application program with the TGBRD/CGBRD TSR
- The message may not be more than 72 characters long
- All texts should be defined as arrays or Hollerith strings, not character strings, in FORTRAN

Examples:

```
CALL TTEXT (MODULE,0,ITEXT,100,ISTAT)
```

```
CALL 'TTEXT' USING MODULE SUB-ADDRESS TEXT—STRING LENGTH  
STATUS—CODE.
```

TTOFF

Turn off application time out for this TPT.

CALL TTOFF
CALL 'TTOFF'.

Parameters:

None

Rules:

- The application time out is turned off until the TTONS TSR is called.

TTONS

Turn on application time out.

CALL TTONS
CALL 'TTONS'.

Parameters:

None

Rules:

- Should be used to turn on the application time out after previous use of TTOFF
- The application time out is set to the default value

TTPST

Read the status of the specified TPT.

CALL TTPST (<TPT no>, <appl no>, <status>)
 CALL 'TTPST' USING <TPT no> <appl no> <status>.

Entry Parameter:

<TPT no>	The TPT number (1-63). 0 means "this TPT".
----------	---

Exit Parameters:

<appl no>	The TPS application number of the application activated by the TPT
<status>	=0, OK = -1, TPT number out of range

Examples:

CALL TTPST(20,APPL NO, ISTAT)

CALL 'TTPST' USING SYS-TPT APPL-NO STATUS-CODE.

TTRAN

Take a transaction checkpoint.

CALL TTRAN
CALL 'TTRAN'.

Parameters:

None

Rules:

- A transaction checkpoint is the point-of-restart after a recovery operation
- The transaction checkpoint data will overwrite the data from the previous transaction checkpoint
- For ND-500-applications, the local data and transaction register block will not be saved on the checkpoint file. Only the contents of task-common and TPT-data will be saved.

TTRON

Turn on the packet log function.

```
CALL TTRON (<CPU-number>)
CALL 'TTRON' <CPU-number>
```

Entry parameter

<CPU-number>	The CPU-number where the packet log will be turned on. =0 in a single CPU-system.
--------------	--

Example:

```
CALL TTRON (0)
CALL 'TTRON' USING ZERO.
```

TTROF

Turn off the packet log function.

```
CALL TTROF (<CPU-number>)
CALL 'TTROF' <CPU=NUMBER>
```

<CPU-number>	The CPU-number where the packet log will be turned off. =0 in a singel CPU-system.
--------------	---

Example:

```
CALL TTROF (0)
CALL 'TTROF' USING ZERO.
```

TTSYN

Allow a synchronised checkpoint

CALL TTSYN
CALL 'TTSYN'.

Parameters:

None

Rules:

- A synchronised checkpoint is the point-of-restart after a rollback operation
- A synchronised checkpoint will be taken if a checkpoint message has arrived, else there will be an immediate return to the application program

TWMSG

Write a message on the operator's console

CALL TWMSG (<text string>)

Entry Parameters:

<text string>	a string of the type CHARACTER containing the text to be written out
---------------	--

Rules:

- TWMSG must be used for character strings with descriptor words (i.e. FORTRAN CHARACTER strings)
- The text string may contain CR and LF. It must be terminated by a ' ' and may not exceed 256 bytes. Bit 7 (parity bit) in each byte must be 0.
- The message will be supplied with time, date and source identity
- Should not be called from PLANC (use CWMSG)

Example:

```
CHAR MTEXT * 80
CALL TWMSG (MTEXT)
```

APPENDIX I:

TSR CALLS – FUNCTIONAL LIST

A list of all TSR calls ordered by function is given in this appendix.

Name	Function	Callable from user application
Task administration TSRs		
TACTV	Activate a concurrent task	Y
TSWAP	Switch to another application	Y
TSTOP	Terminate the transaction	Y
TTERM	Terminate the task	Y
Set strategy TSRs		
TSAST	Setabend strategy	Y
TSTST	Set termination strategy	Y
TSRST	Set restart strategy	Y
TSCST	Set close strategy	Y
Session TSRs		
TSOPN	Open session	Y
TSCLO	Close session	Y
TRMSG	Read message from session partner	Y
TSMSG	Send message to session partner	Y
TSEST	Read session status	Y
TPASZ	Set packet size	Y

Timing TSRs

TASET	Set execution time	Y
TINTV	Set execution interval	Y
TDCNT	Disconnect execution time/interval	Y
TSOPT	Set operator timeout	Y
TTOFF	Turn off application time out	Y
TTONS	Turn on application time out	Y

Message TSRs

TWMSG	Write message to operator	Y
CWMSG	Write message to operator (COBOL)	Y
TTEXT	Write message to unit	Y
TBRDC	Broadcast message	Y
TGBRD	Get broadcast message	Y
CGBRD	Get broadcast message (COBOL)	Y
TR5MG	Get ND-500-monitor error message	N

Checkpoint/restart TSRs

TTRAN	Take a transaction checkpoint	Y
TTSYN	Allow a synchronised checkpoint	Y
THSYN	Do not allow a synchronised checkpoint	Y
TRSES	Restore broken session	N
TRSTO	Restart user application	N
TSCIN	Set checkpoint interval	Y

Critical sequence TSRs

TBSEQ	Marks beginning of critical sequence	Y
TESEQ	Marks end of critical sequence	Y
TSEQU	Marks end and beginning of critical sequence	Y

Status TSRs

TSTAT	Read task status	Y
TABST	Read abend status	N
TRRST	Read restart status	N
TAPST	Read application status	Y
TTPST	Read TPT status	Y
TMISC	Read miscellaneous TPT information	N
TCONF	Read configuration parameters	N

Operator function TSRs

TCHCK	Take a synchronised checkpoint	Y
TROLS	Roll back to synchronised checkpoint	N
TROLT	Rollback to transaction checkpoint	N
TABND	Stop TPS immediately (abnormal end)	N
TCLOS	Close TPS (normal end)	N
THALT	Halt TPS temporarily	N
TCONT	Continue normal TPS operation	N
TAAVA	Set application available	N
TAUNA	Set application unavailable	N
TTROF	Turn off packet log	Y
TTRON	Turn on packet log	Y

APPENDIX J

TPS ON ND-500

The TPS/500-system contains:

- An ordinary TPS/100-system with a 500-monitor running as a special application (TPMON) in the ND-100.
- Applications that may run in the ND-100 and/or in the ND-500.
- SIBAS and screen-handling that may run in either or both machines.

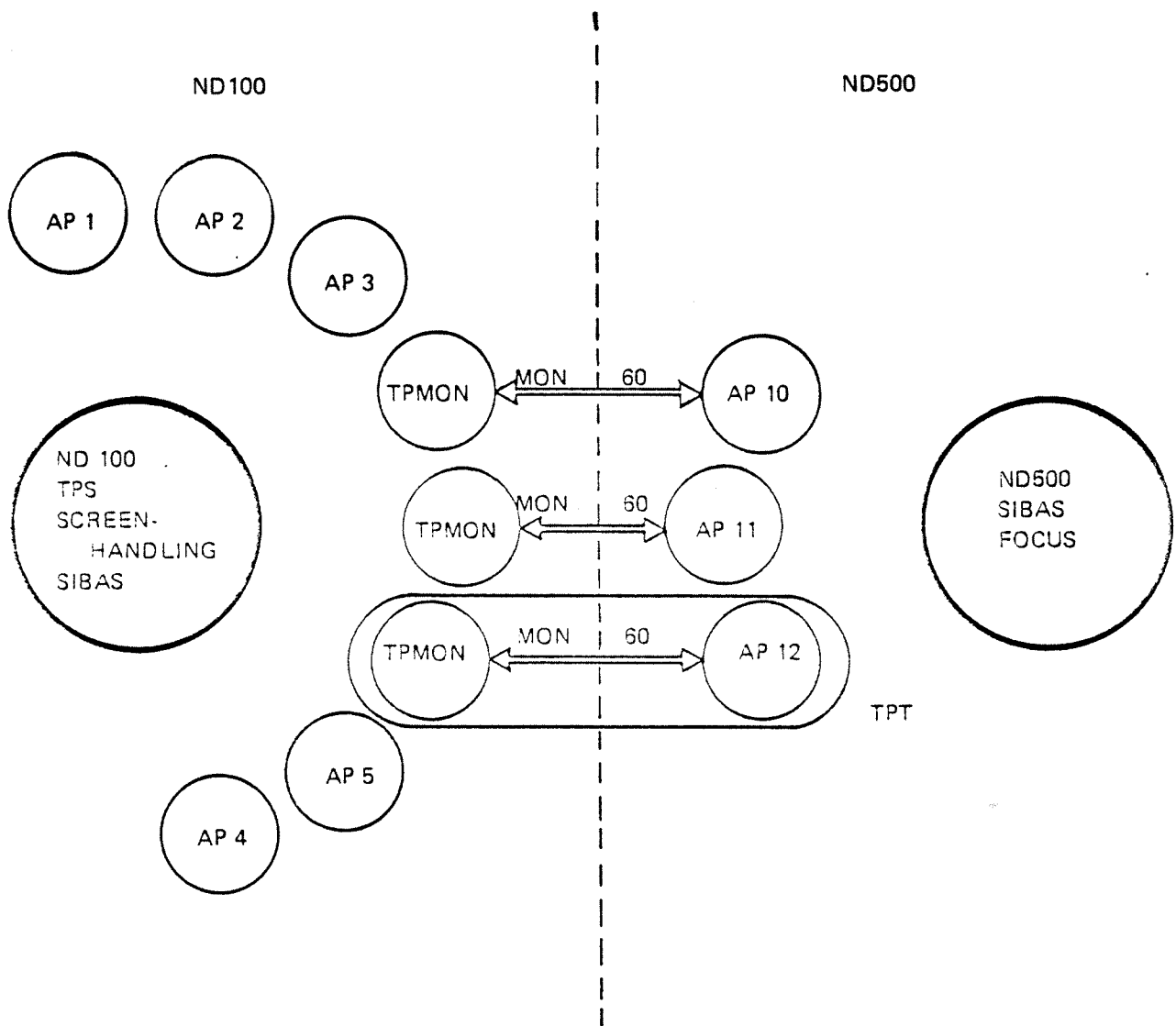


Figure J.1.

Each application running in the ND-500 has a corresponding 500-monitor-application, TPMON, running in the ND-100. The *TPS-system* in the ND-100 sees only this 500-monitor-application. This monitor has total control of the ND-500-process where the application is running. However, other *TPS user* applications in the ND-100 or ND-500 will not see this monitor, they see the ND-500-application only, and in the same way as any other ND-100-application.

One TPT running an ND-500-application has this segment structure:

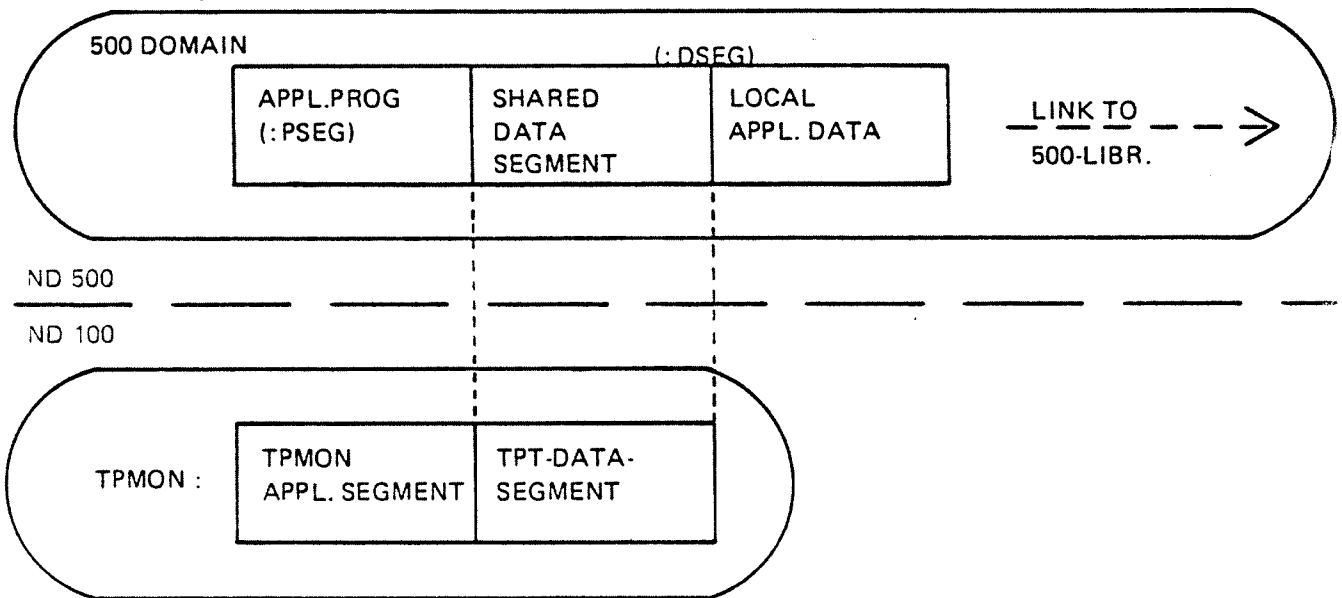
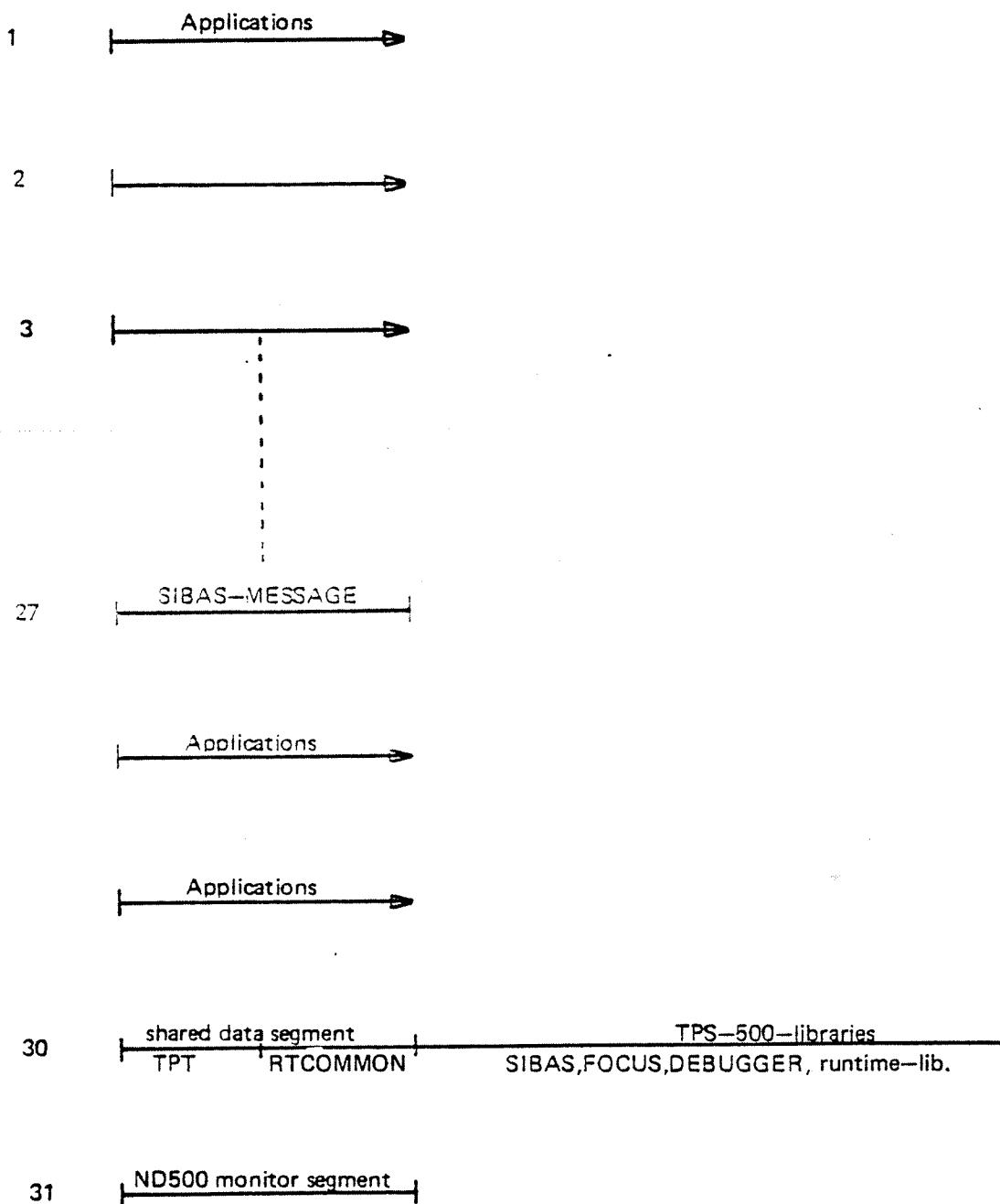


Figure J.2

All data transfer between the ND-100 and ND-500 is done by using the shared segment, shown in the figure above, in order to minimize the system overhead. The task common (i.e. COMMON/PRIVATE/) area is located inside the shared segment area which survives when using the TSR-call TSWAP for switching to another application in the ND-100 or ND-500. This shared segment is contiguously fixed in memory during the whole "life" of an application running in the ND500.

The segment structure in one domain in the ND500 is shown below. There may be several domains:

Segment-no.



134 Mbyte

APPENDIX K

GLOSSARY

abend- abnormal termination of an application program due to an error situation

application- a program run under the control of TPS to do the actual processing of a transaction

application, special- see special application

background- a collective term for timesharing and batch

backup- a copy of the data base, taken regularly and used to restore the data base if it is destroyed

batch- the processing of data that has been collected over a period of time to be processed later in a single run of the application program — done in SINTRAN by using the SINTRAN batch processor

BIM log- SIBAS before-image log, a log of SIBAS records that have been changed, logged before the change is made, in order to be able to roll the data base back to a previous state

checkpoint- the saving on a file of all data used by a program in order to later be able to restore the program to its state when the checkpoint was taken

conversational- a program with the ability to carry on a dialogue (see dialogue)

devices, standard- devices with which an application program may communicate directly through SINTRAN

devices, special- devices with which an application program must communicate through a TPS input/output module

dialogue- the exchange of messages between an application program and a user at a terminal, each message depending on the answer received from the partner

DML- the SIBAS data manipulation language used by an application program

failure, system- see system failure

DRL- the SIBAS data definition/redefinition language

FOCUS - the ND screen handling system (replaces NSHS)

input/output module- a TPS module that communicates with devices such as networks and special terminals externally and TPS modules internally, an interface between the devices and application programs

interactive- a direct connection between a user and a program so that immediate interactions are possible (see dialogue)

IOM- see input/output module

menu- a picture on a display terminal showing the applications available to the terminal user and allowing him to choose one of them

mode file- a symbolic file containing commands and responses to a program usually used interactively, making it possible to run the program in batch mode

monitor call- a call to a routine in the SINTRAN operating system

network- a group of communication devices, such as terminals and concentrators, connected by communication lines, and connected to a computer through a modem device

NSHS- the NORD screen handling system used to control display terminals

OPCOM- the operator communication module of TPS containing commands for controlling TPS and routines for sending messages to the operator from TPS modules

point-of-failure- the state of the TPS system and application programs when a system failure occurs. See system failure.

point-of-restart- the state of the TPS system and application programs after a system restart procedure has been carried out. See system restart.

real time program- a program that is activated by an event external to the computing system, fast enough for the program to exert control over the event

recovery- the process of restoring the data base after a system failure by rapid updating from a checkpoint or backup copy. The point-of-restart after recovery is at the last transaction checkpoint

reentrant- the facility of a program to be used by several users concurrently. Each user has his own data area, but only one copy of the program itself is needed.

restart, system- see system restart

rollback- the process of restoring the data base to its state at a checkpoint, by undoing the updating after the checkpoint. The point-of-restart after rollback is at the last synchronized checkpoint.

ROUTINE log- log of all SIBAS routine calls, used after system failure for data base recovery

segment- an area on mass storage of up to 64K words, containing one or more programs and subroutines to be run as a single load unit

SELECT application- a special application to determine which user application is to be used and switch to the application

session- a connection between an application program and another application program or a device controlled by an IOM. Communication between them is through the TPS message routing system

SIBAS- the data base management system used by TPS

SIGNOFF application- a special application that is given control when a transaction terminates

SIGNON application- a special application used to start a transaction by controlling the terminal user's identity

SINTRAN- the NORD operating system, supporting real time, timesharing and local and remote batch processing

special application- application programs supplied with TPS and used to perform standard functions, such as SIBAS system calls, signon and transaction restart after a system failure

stack- an area (in the data part of the TPT) for the dynamic allocation of variable data for a main program and its subroutines

synchronised checkpoint- a checkpoint taken by all TPS and application programs at the same time

system directives- a set of TPS operator commands to perform system control functions such as start, stop, checkpoint, rollback, recovery

system failure- an error situation resulting in the inability of TPS to continue normal processing. The data base may or may not be intact, but transaction processing is interrupted in both cases

system restart- the process of starting the TPS system after a system failure, including repairing damage to the data base, restoring the data base to a consistent state and restarting transaction processing

task- the processing done by a TPT from the time it is allocated by the TCM until it is freed again

TCM- the transaction control module of TPS used to control the allocation of TPTs

timesharing- the use of a computer by several users at the same time by giving each user in turn control over the computer for a certain length of time — done in SINTRAN through the SINTRAN background processor

TPT- see transaction processing task

transaction- an interaction between one or more I/O devices and an on-line data base, usually involving a dialogue between a user at a terminal and an application program, resulting in some activity on the data base and a response to the user

transaction checkpoint- a checkpoint taken by an individual application program at suitable points in processing

transaction processing system- an on-line computer system providing the facilities needed for the immediate access to a data base either to update the data base or to retrieve information from it

transaction processing task- a TPS unit allocated to a transaction when it is started to control the application program and provide it with a data area and an interface to TPS

transaction service routines- TPS routines called by an application program to perform functions such as communicating with I/O devices and administering task control

TSR- see transaction service routine

unit- either a device controlled by an input/output module (device unit) or a transaction processing task controlled by a transaction control module (TPT unit)

update file- a file that SIBAS writes updated records on instead of updating the data base directly. The data base itself is updated at suitable intervals

INDEX

Section:

abend	
causes	2.3.2, 6.2.2
error message	6.2.2.1
strategy	6.2.2
set abend strategy TSR (TSAST)	2.3.2.2
user abend application	6.2.2
ABEND special application.....	2.3.2.1, 6.2.2
abnormal termination - see abend	
ACCEPT statement (COBOL).....	3.4.3
access control system	6.1.3
activate-concurrent-task TSR (TACTV)	2.2.1.1
ADD-APPL macro	8.3.1
ADD-COB-SUBROUTINES macro.....	8.3.1
ADD-UNIT macro	8.3.1
Alfascope 3500 terminals	3.3.10.4
allocating devices	3.4.1
allow-synchronised-checkpoint TSR (TTSYN)	5.2.3.1
application program	2.1
general description	1.2.1.4
name.....	7.4.1
number.....	7.4.1, App.A
priority	8.3.1
special applications.....	1.2.1.5, 6
application table - see also TPS-TABLES	8.3
asynchronous terminals (FOCUS)	3.2.3.2
background (timesharing and batch)	
programs	7.6
system	3.2.1
library	3.2.5
testing	8.2.5
using SIBAS	3.1.2
backup, data base	5.2.1
batch - see background	
before-image log (BIM log)	
logging updated records.....	5.2.2
restoring data base	5.3.1
BLOCK DATA.....	7.1
BRF format	
application program file type.....	8.3.1
broadcast-message TSR (TBRDC)	4.1.2
BROADCAST command	4.1.3
BSEQU call (SIBAS)	3.1.6
BUILD-LUAP:MODE file	8.3.1
CGBRD (get broadcasted message TSR-COBOL).....	4.1.4
checkpoint	5.2.3
efficiency considerations	7.5.1

Section:

checkpoint, synchronised	5.2.3
allowing	5.2.3.1
taking	5.2.3.3
holding (preventing)	5.2.3.2
use in rollback	5.3.1
checkpoint, transaction	5.2.4
taking	5.2.4.1
use in recovery	5.3.1
checkpoint file	
use at synchronised checkpoint	5.2.3
use at transaction checkpoint	5.2.4
checkpoint weight	
using	5.2.3.3
defining	8.3.1
CHECKPOINT command	5.2.3.3
CHECKPOINT special application	5.2.3, 6.4
clock	
adjust (CLADJ)	4.2
examine (CLOCK and TIME)	4.2
update (UPDAT)	4.2
close data base - see data base, closing	
close-session TSR (TSCLO)	3.3.4
CLOSE file statement	3.4.3
CLOSE Strategy	2.3.1.5
CLOSE - TPS command	6.3
COBOL programs	
input/output	3.4
data areas	7.1.2-7.1.3
special considerations	7.3.2
common area	
task common data area	7.1.2
initialising constant data	7.1
compiling programs	
background	8.2
real time	8.3
communication	
multi-CPU	1.2
special terminals	3.3.10.3,
.....	3.3.10.4
systems	3.3.10.1,
.....	3.3.10.2
concurrent task	2.2.1.1
CONTINUE-TPS command	5.3.2
control Q	3.2.2.2
cursor control	3.2.1, 3.2.2
CWMSG (write message to operator TSR-COBOL)	4.1.1

Section:

data	
constant.....	7.1
variable.....	7.1.1
common (task common data area)	7.1.2
local.....	7.1.3
size of data areas.....	7.1.4
data base	3.1
general description	1.2.2.5
opening	3.1.3
closing	3.1.3
accessing.....	3.1.1,
efficiency (open-close)	7.5.2
data definition/redefinition language (DRL)	1.2.2.5
data entry.....	1.1, 7.5
data manipulation language (DML).....	3.1.1
SIBAS DML statements.....	App. G
data definition/redefinition language (DRL)	1.2.2.5
data/time routine (CLOCK).....	4.2
debugging option (FORTRAN and COBOL)	8.2.4
delayed updating (SIBAS)	
updating records.....	5.2.2
restoring data base.....	5.3.1
descriptor word	
parameter.....	7.3.3.1
string	7.3.3.1
device	
general description	1.2.2.1
special	3.3
opening a session	3.3.2
session communication.....	3.3.6-3.3.7
standard	3.4
allocating	3.4.1
accessing.....	3.4.4
disconnect- application TSR (TDCNT)	2.2.2.3
display terminal - see screen handling	
DISPLAY statement (COBOL)	8.2.5
distributed processing.....	1.2.2.2, 3.3
efficiency	7.5
ERMON monitor call.....	4.1.5
ERMSG monitor call.....	4.1.5
ERRCODE (FORTRAN)	
I/O error code	3.4.2
error message	4.1.5
error messages	
ABEND message.....	6.2.2.1
application error messages	4.1
error message summary.....	App. C
ESEQU call (SIBAS)	3.1.6
execution	
time	2.2.2.1
interval.....	2.2.2.2
EXHIBIT statement (COBOL)	8.2.5

Section:

file	
allocating.....	3.4.3
accessing.....	3.4.4
file status word (COBOL)	
I/O error	3.4.2
error messages	4.1.5
FOCUS screen handling system.....	3.2.3
general description	1.2.2.4
defining forms.....	3.2.3.1
front end CPUs.....	3.2.3.2
FORMS-DEFINE system (FOCUS).....	3.2.3.1
FORTRAN library	
using.....	7.3.1-7.3.2
loading.....	8.2.5
FORTRAN programs	
input/output	3.4
data areas	7.1.2-7.1.3
special considerations	7.3.1
future task.....	2.2.2.1
general purpose macro generator (GPM)	8.3
get broadcasted message TSR (TGBRD/CGBRD).....	4.1.4
HOLD monitor call	4.3
hold synchronised checkpoint TSR (THSYN)	5.2.3.2
IBM-3270-CU input/output module	3.3.10.2
IBM-3270-HOST input/output module	3.3.10.3
illegal monitor call	2.3.2.3
INCH monitor call	3.4.3
input/output module (IOM)	
general description	1.2.2.2
device control.....	3.3
available IOMs	3.3.10
INPUT statement (FORTRAN)	3.4.3
internal devices.....	4.4
internal time.....	4.2
IOM - see input/output module	
IPRIV data area (NSHS)	7.1.2
ISO-1745-HOST input/output module.....	3.3.10.4
ITERM data area (NSHS)	7.1.2
LCOMMO subroutine.....	8.2.3
LEAVE monitor call.....	2.3.1
load-common subroutine - see LCOMMO	
loader	
relocating (NRL).....	8.2.2
real time (RT-L).....	8.3.1
loading application programs	
background	8.2
real time	8.3

Section:

LOAD-SEGMENT macro.....	8.3.1
LOAD-USER-APPL:MODE file.....	8.3.1
local data - see data, local	
MAC assembly language programs	7.3.3
main despatcher (MD)	1.2.4
menus	
master menu	6.1.2
submenu.....	6.1.2
message protocols	3.3
MESSAGE-TO-UNIT command.....	4.1.4
monitor calls	
input/output	3.4
other	4
illegal	2.3.2.3
summary.....	App. E
more flag	3.3.6
multi CPU systems	1.2
names	
application program names.....	7.4.1
user names.....	6.1.1
ND-500 TPS	
data areas	7.2
description	App. J
efficiency.....	7.5.1
FOCUS	3.2.3.4
SIBAS.....	3.1.5
testing applications.....	8.1
network	1.2.2.2, 3.3
NPL programs.....	7.3.3
NRL (relocating loader)	8.2.4
NSHS (see also picture).....	3.2
general description	1.2.2.3
screen definition system.....	7.7.1
screen library system.....	3.2.2
screen library call summary.....	App. F
open data base-see data base, opening	
open-session TSR (TSOPN)	3.3.2
OPEN file statement.....	3.4.1
operator, terminal (user)	1.1
identification	6.1.1
at restart.....	6.2.3, 3.2.2.2
operator, system	
commands.....	1.2.3
messages	1.2.3, 4.1.1
OUTCH monitor call	3.4.3
OUTPUT statement (FORTRAN).....	3.4.3

Section:

packet size	3.3.8
paging	7.5.4
password (user)	6.1.1
peripheral files	3.4.1
periodic task	2.2.2.2
pictures - see also NSHS	
defining	7.7.1
using	3.2.2
restoring	3.2.2.2
pictures, private	7.7
defining	7.7.1
pictures, public	7.7, 7.8.1
producing	7.7.2, 7.8.1
loading	7.7.3, 7.8.1
priority, application	8.3.1
printer output	3.4
PRIVATE common area	7.1.2
public pictures - see pictures, public	
PUBLIC-PICTURE program	7.7.2
 Q ^c Q ^c Q ^c	3.2.2.2
QERMS monitor call	4.1.5
 RAPPL (user restart application)	6.2.3
RFLDS call (NSHS)	3.2.2, App. F
read-message TSR (TRMSG)	3.3.7
READ statement (FORTRAN and COBOL)	3.4.3
real time loader	8.3.1
real time programs	8.3
loading	8.3
names	7.4.1
RECOVER special application	5.3.1, 6.4
recovery, system	5.3.1
reentrant programs	7.1
FORTRAN	7.3.1
COBOL	7.3.2
RELES device monitor call	3.4.3
RESRV device monitor call	3.4.1
restart, system	5.3.2
sessions at restart	3.3.9
files at restart	3.4.4
strategy	6.2.3
set restart strategy TSR (TSRST)	5.3.2.2
user restart application (RAPPL)	6.2.3
RESTART special application	5.3.2.1, 6.2.3
restore picture	3.2.2.2
REWRITE statement (COBOL)	3.4.3
RFILE monitor call	3.4.3
rollback, system	5.3.1
ROLLBACK special application	5.3.1, 6.4

	<i>Section:</i>
routine log	
logging SIBAS calls	5.2.2
restoring data base	5.3.1
RT-common (core common)	
in TPS segment structure	App. D
SIBAS interface	3.1.2
save - common subroutine - see SCOMMO	
SCHPO call (SIBAS)	6.4
SCLDB call (SIBAS)	3.1.3
SCOMMO subroutine	8.1.3
screen handling - see NSHS and FOCUS	
SCREEN-DEFINITION program	7.7.1
security codes	6.1.3
segments, SINTRAN	App. D
SELECT special application	1.3.1, 2.2.4, 6.1.2
semaphores	4.4
send-message TSR (TSMMSG)	3.3.6
send-text-message TSR (TTEXT)	4.1.3
session	1.2.2.2, 3.3
definition	3.3.1
session partner	3.3.1
session request	3.3.1, 3.3.3
session-status TSR (TSEST)	3.3.5
set-abend-strategy TSR (TSAST)	2.3.2.2
set-close-strategy TSR (TSCST)	2.3.1.5
set-execution-time TSR (TASET)	2.2.2.1
set-interval TSR (TINTV)	2.2.2.2
set-packet-size TSR (TPASZ)	3.3.8
set-restart-strategy TSR (TSRST)	5.3.2.2
set-termination-strategy TSR (TSTST)	2.3.1.4
SET - UNAVAILABLE command	8.3.1
SETDV call (SIBAS)	3.1.4
SIBAS DBMS	3.1
general description	1.2.2.5
data definition/redefinition language (DRL)	3.1.1
data manipulation language (DML)	3.1.1
DML call summary	App. G
interface routine	3.1.2, 8.2.5
data base logging	5.2.2
synchronised checkpoint	5.2.3
rollback and recovery	5.3.1
SIGNOFF special application	1.3.1, 2.3.1.3, 6.2.1
SIGNON special application	1.3.1, 2.2.4, 6.1.1
SOPDB call (SIBAS)	3.1.3
special applications	1.2.1.5, 6
special TPS device - see device special	
SPECIFY-LUAP:SYMB file	8.3.1

Section:

SREPR call (SIBAS).....	6.4
SROLL call (SIBAS).....	6.4
stack.....	7.1.3
standard device - see device, standard	
STANSAAB terminals.....	3.3.10.4
starting	
tasks.....	2.2
transactions.....	2.1, 1.3
application programs.....	2.1, 2.2.3
stop code.....	2.3.1.1
stop-transaction TSR (TSTOP)	2.3.1.1
subroutine.....	7.4.2
data area in stack.....	7.1.3
loading	8.3.1
switch-application-program TSR (TSWAP)	2.2.3.1
synchronised checkpoint - see checkpoint, synchronised	
synchronised-checkpoint TSR (TCHCK)	5.2.3.3
synchronous terminals (FOCUS).....	3.2.3.3
 TACTV (start-concurrent-task TSR)	2.2.1.1
TASET (set-execution-time TSR)	2.2.2.1
task	2.1
starting	2.2
concurrent	2.2.1.1
future	2.2.2.1
periodic.....	2.2.2.2
terminating.....	2.3, 6.2.1
task common data area.....	7.1.2
TBRDC (broadcast-message TSR)	4.1.2
TCHCK (synchronised checkpoint TSR).....	5.2.3.3
TCM - see transaction control module	
TDCNT (disconnect-application TSR)	2.2.2.3
terminals	1.1
asynchronous (in FOCUS)	3.2.3.2
display	3.2.1
synchronous (in FOCUS)	3.2.3.3
terminate-task TSR (TTERM)	2.3.1.2
termination	
task.....	2.3.1.2
transaction	2.3, 6.2.1
complete	2.3.1.1
normal	2.3.1
abnormal - see also abend	2.3.2
strategy	6.2.1
set termination strategy TSR (TSTST)	2.3.1.4
user termination application	6.2.1
testing programs	8.2
TGBRD (get broadcasted message TSR).....	4.1.4
THSYN (hold synchronised checkpoint TSR)	5.2.3.2
time/date routine (CLOCK).....	4.2

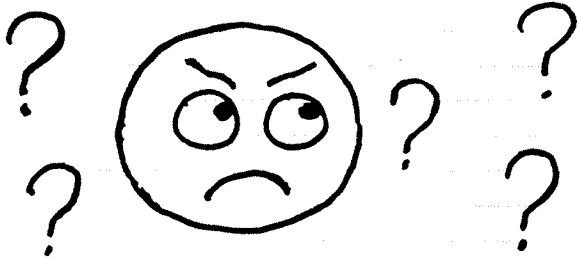
Section:

TIME monitor call	4.2
timeout	
application.....	2.3.2.4, 6.2.2
operator.....	2.3.2.4, 6.2.2
response at restart.....	6.2.3
timesharing - see background	
timing TSRs	2.2.2
TINTV (set-interval TSR)	2.2.2.2
TPASZ (set-packet-size TSR)	3.3.8
TPCLOSE special application	6.3, 3.1.3
TOPEN special application	6.3, 3.1.3
TPS:PROG background program.....	8.2.1
TPS-DEFAULT	6.1.3
TPS-MENUTAB	6.1.3
TPS-TABLES	8.3.1
TPS-TERMTAB	6.3
TPS-USER user name.....	8.3
TPS-USERTAB	6.1.3
TPT- see transaction processing task	
transaction	1.1, 2.1
starting	2.2
terminating	2.3
transaction checkpoint - see checkpoint, transaction	
transaction checkpoint TSR(TTRAN)	5.2.4.1
transaction control module (TCM)	1.2.1.1
transaction processing task (TPT)	1.2.1.2
transaction service routine (TSR).....	1.2.1.3
TSR call formats	App. H
TSR call summary	App. I
TRMSG (read-message TSR)	3.3.7, 3.3.10
TSAST (set abend strategy TSR)	2.3.2.2
TSCLO (close-session TSR)	3.3.4, 3.3.10
TSCST (set-close-strategy TSR)	2.3.1.5
TSEST (session-status TSR).....	3.3.5
TSMMSG (send-mesage TSR)	3.3.6, 3.3.10
TSOPN (open-session TSR)	3.3.2, 3.3.10
TSR-see transaction service routine	
TSRST (set restart strategy TSR).....	5.3.2.2
TSTOP (terminate-transaction TSR)	2.3.1.1
TSTST (set termination strategy TSR)	2.3.1.4
TSWAP (switch-application TSR)	2.3.1
TTERM (terminate-task TSR)	2.3.1.2

Section:

TTEXT (send-text-message TSR)	4.1.3
TTRAN (take transaction checkpoint TSR)	5.2.4.1
TTSYN (allow synchronised checkpoint TSR).....	5.2.3.1
TWMSG (write-message-to-operator TSR)	4.1.1
unit number device.....	1.2.2.2
TPT.....	1.2.1.2
UPDAT monitor call.....	4.2
updatefile (SIBAS).....	5.2.2
user name terminal-see SIGNON	
TPS files.....	8.3.2
user terminal-see operator,terminal	
wait option (of RESRV)	3.4.2
WFILE monitor call	3.4.3
working set	7.5.4
working storage	7.1.2-7.1.3
write-mesage-to-operator TSR (TWMSG/CWMSG)	4.1.1
WRITE statement (FORTRAN and COBOL)	3.4.3
X25LAPB input/output module.....	3.3.10.1

***** **SEND US YOUR COMMENTS!!!** *****

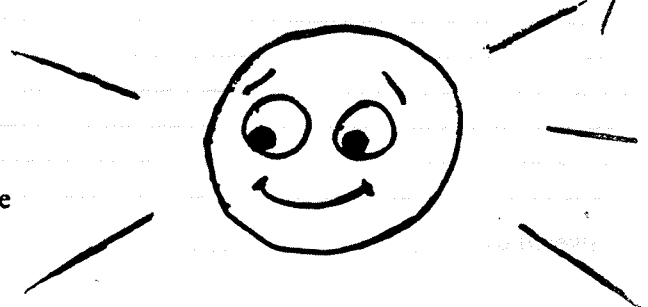


Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card - and an answer to your comments.

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: _____

Manual number: _____

What problems do you have? (use extra pages if needed) _____

Do you have suggestions for improving this manual? _____

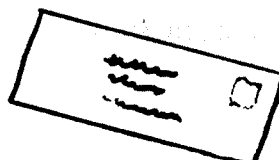
Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

What are you using this manual for? _____

Send to: Norsk Data A.S.
Documentation Department
P.O. Box 4, Lindeberg Gård
Oslo 10, Norway



Norsk Data's answer will be found on reverse side

1. What is the main purpose of the document?
 2. What are the key findings of the study?
 3. What are the limitations of the study?
 4. What are the implications of the study?
 5. What are the conclusions of the study?
 6. What are the recommendations of the study?
 7. What are the future research directions?
 8. What are the acknowledgments?
 9. What are the references?
 10. What are the appendices?

Oslo 10, Norway

The Competitive European Computer Company

NORSK DATA A.S JERIKOVN. 20 P.O. BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY
TEL.: 02 - 30 90 30 - TELEX: 18661