TPNS
Teleprocessing Network Simulator

Function and Service Enhancements-2001

Version 3 Release 5

SC31-8654-02

TPNS
Teleprocessing Network Simulator

IBM

Function and Service Enhancements-2001

Version 3 Release 5

SC31-8654-02

Note!

Before using this document, read the general information under "Appendix E. Notices" on page 117.

Second Edition (December 2001)

This document applies to the Teleprocessing Network Simulator Version 3 Release 5 (program number 5688-121), an IBM licensed program, which runs under the following operating systems:

MVS/370 (MVS/SP Version 1 or later)

MVS/Extended Architecture (MVS/SP Version 2 or later)

MVS/Enterprise System Architecture (MVS/SP Version 3 or later)

OS/390

z/OS

Any VM system (with or without High Performance Option) that supports Group Control System (GCS)

Publications are not stocked at the address given below. If you want more IBM publications, ask your IBM representative or write to the IBM branch serving your locality.

A form for comments has been provided at the back of this document. If this form has been removed, you may address comments to IBM Corporation, TPNS Development and Support, Dept. A30A/B503, P.O Box 12195, 4205 S. Miami Blvd., Research Triangle Park, North Carolina 27709, U.S.A. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Contents

About This Book

This book describes the function and service enhancements to TPNS Version 3 Release 5 that are being made available in TPNS Service Levels 9711 and 0110. The new features and updates included in these service levels are not explained in the existing TPNS product documentation.

Service Level 0110 Enhancements

Among the new function and service enhancements for Service Level 0110 are:

TPNS STL Script Generation Support
The TPNS Script Generator Utility has been enhanced with an option to generate scripts in Structured Translator Language (STL).

STL Scripting Data Manipulation Functions
Many enhancements have been made to extend the capabilities of STL. These enhancements make STL more REXX-like and provide the capability to do advanced data manipulation.

Local Port and Limited Server Enhancements
Two enhancements have been made to the TPNS TCP/IP simulation support. These enhancements include the following:

- The addition of the LOCLPORT operand which allows the specification of the local port number to be used by a Simple TCP (STCP) or Simple UDP (SUDP) device.
- The addition of the STCPROLE operand which allows you to specify whether an STCP device should act as a client or server.
- An exit routine, ITPGSIPA, has been added which can be used to glean the address information from received data and set the address information for data to be transmitted. This routine is used in conjunction with the limited server enhancement.

Enhancements to CPI-C Protocol Support
Enhancements have been made to the CPI-C Protocol Support to provide Single Session Support.

CPI-C Comparison Logic Enhancements
Enhancements have been made to the error handling logic when generating CPI-C scripts. Error handling logic is controlled by the SENDL and NOSENDL commands.

CPI-C Composite Fields
Support has been added for composite field creation when generating CPI-C scripts. Creation of composite fields is controlled by the FIELD and NOFIELD commands.

Miscellaneous Enhancements
Several other enhancements have been made for this service level of TPNS. These enhancements include the following:

- The STL PATHID function returns the name of the PATH statement currently being used.
- The ITPFIOX File I/O User Exit provides file I/O capabilities to a TPNS script.
- SAY and WTO enhancements which include the following:

  - The maximum length that is written to the operator has been increased from 50 to 100.
  - Data can be written with an abbreviated header containing only a TPNS message number preceding the data.
  - The addition of the WTOABRHD statement which indicates that only abbreviated headers are to be used.
- The NOWTOR TPNS execution parameter has been added to allow TPNS to be executed under MVS as a batch job without issuing a WTOR for operator control.
- View is now used instead of browse to look at data sets through the TPNS/ISPF Interface.
- The LOG operand has been added to the DSPLY loglist control command. This operand causes only log display records created by LOG DISPLAY scripting statements to be printed.

Service Level 9711 Enhancements

Among the new function and service enhancements for Service Level 9711 are:

TPNS CPI-C Script Generator support
You can now automatically generate TPNS scripts for CPI-C transaction programs (TPs).

Additional support for TCP/IP simulations
TPNS can now simulate the following types of clients:

- Telnet 3270E clients connecting to a Telnet 3270 or 3270E server
- Telnet Line Mode Network Virtual Terminal clients connecting to a Telnet server
- Telnet 5250 clients connecting to a Telnet 5250 server
- Simple UDP clients

Other TCP/IP-related enhancements included in this service level are:

- TPNS now uses the Macro API of the IBM TCP/IP product when it is available.
- The port number to be used for subsequent connections can now be altered for TCP/IP resources via PORT=nnnnn on the *a* (alter) TPNS operator command.

Scripting enhancements
A variety of changes have been made to extend the capabilities of the TPNS scripting languages. Among the enhancements and updates are:

- An OVERLAY function that allows you to write statements that replace or overlay an existing text string with new text.
- A delay cancellation function that enables you to write scripts that account for the possibility that long delays may need to be canceled based on operator or other input.
- The upper limits for index counters, switches, IFs, and save areas have been increased to enable you to define additional index counters, switches, IFs, and save areas for simulated resources.
- Named queue support has been added to provide an easier method for passing data between simulated resources by using named queues.

Who Should Read This Book

Read this book if you want to use the new TPNS functions and learn about TPNS service enhancements. The book assumes you are familiar with TPNS and with the statements and commands used in TPNS message generation decks and STL procedures.

If you are a new TPNS user, you may want to become familiar with the following books in the TPNS library before using this book:

- For an overview of TPNS and how to begin using it, refer to *TPNS Planning and Installation* and *TPNS Primer*.
- For information about creating message generation decks, refer to *Creating TPNS Message Generation Decks*.
- If you will be capturing data and using the captured data to produce message generation decks or STL procedures, refer to *TPNS Script Generating Utilities*.
- For more information about TPNS networks, see *Defining TPNS Networks*.
- Refer to *TPNS Language Reference* for information on the syntax of the TPNS scripting language statements and to *Using the TPNS Structured Translator Language (STL) and the STL Translator* for information on the syntax of STL statements.

How to Use This Book

The purpose of this book is to describe new function and service enhancements available for TPNS Version 3 Release 5. The book is intended for experienced TPNS users. If you are not already familiar with TPNS, you may want to refer to *TPNS Primer* and the other books in the TPNS library.

This book is divided into the following sections:

- “Chapter 1. TPNS CPI-C Script Generation Support” on page 1 describes TPNS enhancements that enable you to generate scripts for CPI-C TPs.
- “Chapter 2. TPNS STL Script Generation Support” on page 21 describes TPNS enhancements that enable you to generate scripts in Structured Translator Language.
- “Chapter 3. Additional TPNS TCP/IP Support” on page 27 describes enhancements that enable you to simulate the following types of TCP/IP clients:
  - Telnet 3270E clients connecting to a Telnet 3270 or 3270E server
  - Telnet Line Mode Network Virtual Terminal clients connecting to a Telnet server
  - Telnet 5250 clients connecting to a Telnet 5250 server
  - Simple UDP clients

  The chapter also describes the following other TPNS TCP/IP enhancements:
  - TPNS now uses the Macro API of the IBM TCP/IP product when it is available.
  - The port number to be used for subsequent connections can now be altered for TCP/IP resources via PORT=nnnnn on the *a* (alter) TPNS operator command.
- “Chapter 4. TPNS Scripting Enhancements” on page 41 describes a variety of changes that extend the capabilities of the TPNS scripting languages and address service needs.

- "Chapter 5. Miscellaneous Enhancements" on page 87 describes maintenance-related changes to TPNS Version 3 Release 5 Service Level 9711 and 0110.
- "Chapter 6. Migration Considerations" on page 97 describes considerations in migrating to TPNS Version 3 Release 5 Service Level 9711 and 0110.
- "Appendix A. Work Station Trace Reformatter Utility" on page 101 provides information on ITPWSTRF, a new TPNS utility used to reformat OS/2 Communications Manager (CM/2) and IBM Communications Server traces so they can be used to generate CPI-C scripts.
- "Appendix B. Messages and Return Codes" on page 105 describes messages and return codes added or changed by the TPNS enhancements.
- "Appendix C. Simple TCP Sample Script" on page 109 provides an example of a Simple TCP script that provides Telnet Line Mode Network Virtual Terminal support. Simple TCP is an alternative means of simulating the various Telnet protocols discussed in "Chapter 3. Additional TPNS TCP/IP Support" on page 27.
- "Appendix D. Miscellaneous" on page 115 addresses changes required to TPNS sample network models and new members of the TPNS Sample data set.

Where to Find More Information

The following list shows the books in the TPNS library. For information on other related publications, see the ″Bibliography″.

TPNS General Information, GH20–2487

TPNS Primer, SC31–6043

TPNS Planning and Installation, SH20–2488

Defining TPNS Networks, SC31–6008

Creating TPNS Message Generation Decks, SC31–6009

TPNS Language Reference, SH20–2489

TPNS Script Generating Utilities, SC30–3453

TPNS Samples, SC30–3454

TPNS STL Reference Card, SX75–0065

Using the TPNS Structured Translator Language (STL) and the STL Translator, SC31–6013

TPNS Operation, SC30–3289

TPNS Messages and Codes, SC30–3310

TPNS Test Manager User's Guide and Reference, SC31–8719

TPNS General Utilities, SC30–3290

TPNS User Exits, SC31–6009

TPNS Master Index, GC31–6059

IBM Online Library: IBM Networking Systems Softcopy Collection Kit (CD-ROM containing softcopy of all manuals except for TPNS Master Index and TPNS STL Reference Card), SK2T-6012

TPNS Library (all manuals except TPNS General Information, and TPNS Test Manager User's Guide and Reference), SB0F-1426

For more information about TPNS visit us on the Internet at `www.software.ibm.com/network/tpns`, or call the TPNS Hotline at 1-800-835-8767 (US and Canada) or (919)543-5983 (all others). You can also contact us through e-mail at `tpns@us.ibm.com` for questions on all matters related to TPNS.

Chapter 1. TPNS CPI-C Script Generation Support

This chapter describes enhancements to the TPNS Script Generator Utility (ITPSGEN) that enable you to generate CPI-C scripts using data traffic captured from live system runs. Refer to *TPNS Script Generating Utilities* and *Defining TPNS Networks* for detailed information on using ITPSGEN and on creating network definitions.

Function Overview

CPI-C script generation support has been added to TPNS. This enables you to use the TPNS Script Generator Utility (ITPSGEN) to generate CPI-C scripts from traces of CPI-C applications or other applications that produce LU 6.2 line flows.

The procedure for generating CPI-C scripts using ITPSGEN is similar to that used for generating other types of scripts. To generate CPI-C scripts, you must follow these steps:

1. obtain a trace of system activity
2. reformat the captured data using the appropriate utility program
3. sort the reformatted data
4. define a model network for your simulation
5. use ITPSGEN to generate the scripts in STL
6. translate the STL to TPNS scripting language using the STL Translator

Before using ITPSGEN to generate CPI-C scripts, you must make minor modifications to the existing script generation JCL, EXEC, or CLIST. Refer to “Changes to JCL, EXECs, and CLISTs” on page 5 for information on the required modifications. You can also create CPI-C scripts by invoking the script generator from the TPNS/ISPF Interface.

Note: If you are using the TPNS/ISPF Interface, you may need to prep the model network before translating the STL. Otherwise, the wait deck in the model network may not be found during the STL translation.

Tracing Considerations

The first step in generating CPI-C scripts is to capture a trace containing LU 6.2 line flows. The trace of system activity can be a VTAM buffer trace, an OS/2 Communications Manager (CM/2) trace, or an IBM Communications Server trace. The trace must be reformatted before it can be used for script generation.

Note: Traces must contain the FMH-5 allocate request for any conversations that are to be simulated. To ensure your trace contains the required allocate requests, activate the trace before establishing any conversations you want to simulate.

A new Work Station Trace Reformatter utility (ITPWSTRF) has been provided with TPNS to allow you to reformat OS/2 Communications Manager (CM/2) and IBM Communications Server traces. Refer to “Appendix A. Work Station Trace Reformatter Utility” on page 101 for more information on ITPWSTRF. The existing VTAM Buffer Trace Reformatter (ITPVTBRF) can be used to reformat VTAM buffer traces. For all tracing options, the reformatted trace file must be sorted before it can be used to generate CPI-C scripts. The files must be sorted in ascending order by resource name, session, date, and time fields.

VTAM Buffer Trace

You can use a VTAM buffer trace as your source for generating a CPI-C script. The trace may be recorded using either the GTF or the NPM data collection facility. You should set up the trace in the same manner as you would when generating other types of scripts (refer to Chapter 5 of the *TPNS Script Generating Utilities* manual for further information). Be sure to request a full buffer trace by specifying the AMOUNT=FULL parameter. After you have traced a scenario that involves CPI-C applications or other applications that produce LU 6.2 line flows, you must reformat the captured trace file using the VTAM Buffer Trace Reformatter (ITPVTBRF). Sort the resulting file as described in Chapter 5 of the *TPNS Script Generating Utilities* manual before running the TPNS Script Generator Utility.

OS/2 Communications Manager (CM/2) Trace

You can use the OS/2 Communications Manager (CM/2) trace facility to capture an LU 6.2 trace. After capturing a trace, upload the trace file to the host as an EBCDIC TEXT file. Reformat the trace file using the Work Station Trace Reformatter (ITPWSTRF) and then sort the reformatted file. Refer to “Appendix A. Work Station Trace Reformatter Utility” on page 101 for more information on ITPWSTRF.

IBM Communications Server Trace

You can also use the IBM Communications Server trace facility to capture an LU 6.2 trace. After capturing the trace, upload it to the host as an EBCDIC TEXT file. Reformat the trace file using the Work Station Trace Reformatter (ITPWSTRF) and then sort the reformatted file. Refer to “Appendix A. Work Station Trace Reformatter Utility” on page 101 for more information on ITPWSTRF.

Tracing Dependencies and Restrictions

Before attempting to generate CPI-C scripts, you should be aware of tracing issues that could affect whether CPI-C scripts accurately represent the intended testing scenario. There is no information in the trace file that differentiates transaction programs or conversations. The TPNS CPI-C Script Generator Utility makes the assumption that each session represents a unique transaction program and that each attach request on the session represents the start of a new serial conversation. The TPNS Script Generator Utility could produce unexpected results in scripts generated from traces containing multiple transaction programs (TPs) per LU, TPs processing multiple overlapping conversations, or full-duplex sessions. You also should be aware of the circumstances under which the VTAM buffer trace facility may fail to produce complete traces of conversations.

Traces Containing Multiple TPs or Conversations

The TPNS CPI-C script generation facility considers each unique session captured in a system trace to represent a TP. As a result, scripts might not accurately represent captured traces if the traces include any of the following:

- a TP processing multiple, overlapping conversations
- multiple TPs concurrently active on the same LU
- multiple instances of the same TP

If your trace includes a TP processing multiple conversations, the conversations should be serial, rather than overlapping. If a trace contains a TP processing multiple overlapping conversations or serial conversations using different sessions, the script generated from the trace will not accurately represent the captured trace because the script generator will create a different TP for each conversation.

Traces that include multiple TPs concurrently active on the same LU also can produce unexpected results in generated scripts. Depending on timing, two or more TPs concurrently active on the same LU can have conversations sharing the same session. If this is the case, the generated script will not accurately represent the traced scenario.

If the trace contains multiple instances of the same TP, the resulting script will not accurately represent the original traced scenario. If traces containing multi-instance TPs are used as input to the script generator, you will need to make numerous revisions to the generated scripts to make them accurately represent the original traced scenario.

Note: You will obtain the best results using the TPNS CPI-C script generation facility if your trace contains one single-instance TP per LU.

Full-Duplex Sessions

Trace files that contain one or more full-duplex sessions should be used with caution. TPNS simulations that use scripts generated from full-duplex sessions may not accurately reproduce the original traced scenario. However, if a session is identified as full-duplex but used as if it were half-duplex flip-flop, the generated scripts should accurately reproduce the original traced scenario.

VTAM Buffer Traces

If you will be using VTAM Version 4 Release 4 (or later) buffer traces as your source for generating CPI-C scripts, you should be aware of the circumstances under which using the traces could result in incomplete CPI-C scripts. Buffer traces produced by VTAM Version 4 Release 4 or later do not capture a complete trace of conversations using the APPCCMD interface when the origin and destination are within the same VTAM host. If the application being traced is using the VTAM APPCCMD interface to communicate with a partner APPLID defined on the same VTAM host, only the conversation setup information will be in the VTAM buffer trace data set; all other data sent and received are not captured in the trace. As a result, scripts generated from the trace will be incomplete.

Automatic Script Generation Considerations

As a general rule, scripts produced by the TPNS Script Generator Utility should be used as a base. It is unrealistic to expect to play-back hours of live data capture without modification to the generated scripts. If the interdependencies among LUs or TPs is minimal, the modification required should be minor. The best case scenario is LU names need to be updated to match your test system environment. However, as a general rule, there are timing relationships among traced LUs and TPs. For instance, one TP is dependent on receiving an attach request from another TP, or a particular LU makes a database update that is dependent on another LU having made an earlier update to the same field. The TPNS script generator creates a script to represent each SNA session as if it is totally independent of all other SNA sessions. Therefore, to accurately reproduce the timing relationships among the generated scripts, logic will need to be manually added to the scripts. This process will require an intimate knowledge of the traced applications.

A good approach to automatic script generation is to limit the trace file to a known transaction rather than tracing all transactions in a system simultaneously. Then add logic to the generated scripts to handle timing relationships for that transaction. Repeat this process for each transaction in the system to be simulated. By limiting the scope of the trace file, timing relationships are much easier to identify.

Network Definitions

Before you can use ITPSGEN to generate CPI-C scripts or scripts for other simulation types, you must define a model network. Refer to *Defining TPNS Networks* for detailed information on coding network definition statements.

When creating a network definition for a CPI-C simulation, you must define a single message generation sequence path as path 0. This will be the default path used by any resources for which no script is generated.

A CPI-C APPCLU definition must be specified for each traced resource for which a script will be generated. The APPCLU statement name must match a resource name in the input trace file. There must be at least one TP statement following each APPCLU statement. TP statement names can be selected at your discretion. If automatic network updating is requested, the recommended approach is to specify no operands on the TP statement.

Sample Model Network

The following is a sample of a network definition intended for use in generating CPI-C scripts. Note that the APPCLU statement names are LU1 and LU2. For a script to be generated for these resources, there must be records in the input trace file with the same resource names.

```
CPICSGEN NTWRK CONRATE=YES,            * Message rates to the console
               OPTIONS=(MONCMND),      * Monitor operator commands
               CPITRACE=VERB,          * Trace CPI-C verbs
               PATH=(0),               * Default path statement
               HEAD='CPI-C NETWORK'
*
*  Model network for CPI-C script generation.
*
0       PATH    SGENTXT
*
LU1     APPCLU
CLIENT  TP
*
LU2     APPCLU
SERVER  TP
*
SGENTXT  MSGTXT
        WAIT
        ENDTXT
```

Changes to JCL, EXECs, and CLISTs

Before using ITPSGEN to generate CPI-C scripts, you must add an STLTXT DD definition to the existing script generation JCL, EXEC, or CLIST. This DD is for an output partitioned data set that will contain the STL source. Each STL MSGTXT will be a member in this partitioned data set. The data set can have the same attributes as those used for the MSGTXT data set. The following sample JCL, EXEC, and CLIST illustrate the addition of the STLTXT DD definition.

Sample JCL

The example below shows JCL that has been modified to enable ITPSGEN to generate CPI-C scripts.

```
//SGENJOB  JOB
//*********************************************************************
//* Teleprocessing Network Simulator (TPNS)   5688-121               *
//*********************************************************************
//*                    SGENJOB JCL                                    *
//* Sample JCL to execute ITPSGEN.                                    *
//*********************************************************************
//JOBLIB   DD  DSNAME=TPNS.LOAD,DISP=SHR
//STEP1    EXEC PGM=ITPSGEN,PARM='CTL'
//RATEDD   DD  DSNAME=TPNS.RATETBLS,DISP=SHR
//INITDD   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//MSGDD    DD  DSNAME=TPNS.MSGFILE,DISP=SHR
//SYSUT2   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSPRINT DD  SYSOUT=A
//MSGTXT   DD  DSNAME=TPNS.MSGFILE,DISP=SHR
//STLTXT   DD  DSNAME=TPNS.STLIN,DISP=SHR
//NTWRK    DD  DSNAME=TPNS.TESTFILE,DISP=SHR
//SEQOUT   DD  DSNAME=TPNS.SEQOUT,DISP=SHR
//TAPEIN   DD  UNIT=TAPE,DISP=OLD,VOL=SER=TAPEIN,LABEL=(,NL)
//CTLIN    DD  *
* CONTROL STATEMENTS FOR ITPSGEN SCRIPT GENERATOR *
LIST
SEQOUT
NTWRK
REPORT FULL
/*
//SYSIN    DD  *
CPICSGEN NTWRK CONRATE=YES,            * Message rates to the console
               OPTIONS=(MONCMND),      * Monitor operator commands
               CPITRACE=VERB,          * Trace CPI-C verbs
               PATH=(0),                * Default path statement
               HEAD='CPI-C NETWORK'
*
*  Model network for CPI-C script generation.
*
0        PATH    SGENTXT
*
LU1      APPCLU
CLIENT   TP
*
LU2      APPCLU
SERVER   TP
*
SGENTXT  MSGTXT
         WAIT
         ENDTXT
/*
```

Sample REXX EXEC

The example below shows a REXX EXEC that has been modified to enable ITPSGEN to generate CPI-C scripts.

```
/*********************************************************************
 **  Teleprocessing Network Simulator (TPNS)   5688-121             **
 *********************************************************************
 **                     SGEN EXEC                                   **
 **                                                                 **
 **  Sample EXEC to execute ITPSGEN on CMS.  It can also be used to **
 **  execute ITPSGEN on GCS if the file type is changed from EXEC   **
 **  to GCS and the two ERASE commands at the end are removed.      **
 **                                                                 **
 **  Format:                                                        **
 **           SGEN prepin_name prepin_type trace_name trace_type    **
 **                                                                 **
 *********************************************************************/
arg prep_fn prep_ft trace_fn trace_ft .
'FILEDEF * CLEAR'
'FILEDEF CTLIN    DISK TPNS     SGENCNTL *'
'FILEDEF INITDD   DISK TPNS     TESTFILE A'
'FILEDEF MSGDD    DISK TPNS     MSGFILE  A'
'FILEDEF MSGTXT   DISK TPNS     MSGFILE  A'
'FILEDEF STLTXT   DISK TPNS     STLIN    A'
'FILEDEF NTWRK    DISK TPNS     TESTFILE A'
'FILEDEF RATEDD   DISK TPNS     RATETBLS * (BLKSIZE 2000)'
'FILEDEF SEQOUT   DISK TPNS     SEQOUT   A'
'FILEDEF SYSIN    DISK' prep_fn  prep_ft  '*'
'FILEDEF SYSPRINT PRINTER'
'FILEDEF TAPEIN   DISK' trace_fn trace_ft '*'
'GLOBAL  LOADLIB  TPNSLOAD'
'OSRUN   ITPSGEN'
rcode=rc
'ERASE FILE SYSUT2 A'
'ERASE FILE SYSUT3 A'
exit rcode
```

Sample TSO CLIST

The example below shows a CLIST that has been modified to enable ITPSGEN to generate CPI-C scripts.

```
/*********************************************************************/
/* Teleprocessing Network Simulator (TPNS)   5688-121               */
```

```
/*******************************************************************/
/*                      SGEN CLIST                                 */
/* Sample CLIST to execute ITPSGEN.                                */
/*******************************************************************/
FREE  DDNAME(SYSPRINT RATEDD INITDD MSGDD SYSUT2 SYSUT3)
FREE  DDNAME(MSGTXT STLTXT NTWRK SEQOUT TAPEIN CTLIN SYSIN)
ALLOC DDNAME(SYSPRINT) SYSOUT(A)
ALLOC DDNAME(RATEDD)   DATASET('TPNS.RATETBLS') SHR
ALLOC DDNAME(INITDD)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(MSGDD)    UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(SYSUT2)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(SYSUT3)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(MSGTXT)   DATASET('TPNS.MSGFILE')   SHR
ALLOC DDNAME(STLTXT)   DATASET('TPNS.STLIN')     SHR
ALLOC DDNAME(NTWRK)    DATASET('TPNS.TESTFILE')  SHR
ALLOC DDNAME(SEQOUT)   DATASET('TPNS.SEQOUT')    SHR
ALLOC DDNAME(TAPEIN)   DATASET('ITPSGEN.TAPEIN') SHR
ALLOC DDNAME(CTLIN)    DATASET('ITPSGEN.CTLIN')  SHR
ALLOC DDNAME(SYSIN)    DATASET('ITPSGEN.SYSIN')  SHR
CALL  'TPNS.LOAD(ITPSGEN)' 'CTL'
FREE  DDNAME(SYSPRINT RATEDD INITDD MSGDD SYSUT2 SYSUT3)
FREE  DDNAME(MSGTXT STLTXT NTWRK SEQOUT TAPEIN CTLIN SYSIN)
```

Changes to the TPNS/ISPF Interface

When generating CPI-C scripts using the TPNS/ISPF Interface, you must specify a data set to contain the generated STL source. Use the field labeled **Generated STL programs** on panel ITP0SGNP to specify this data set. A member will be created in this data set for each STL MSGTXT that is generated. The data set can have the same attributes as those used for the generated message decks data set. The following is an example of the ITP0SGNP panel as it might look when generating STL scripts.

```
ITP0SGNP          TPNS:  Generate Message Decks from Sorted Trace Data

Type information.  Then Press Enter
                                                                 More:   +
   Input Data Sets
      Sorted trace . . . . . . 'USERID.SORTED.TRACE'
        Tape:  Serial numbers           ,
               File number . .         (0-9999)
               Label type  . .         (NL or SL)
      Model script . . . . . . 'USERID.NETWORKS(MODEL)'
      Control commands . . . . 'USERID.CONTROL(SGEN)'

   Output Data Sets
      Generated message decks  'USERID.MSGFILE'
      Generated STL programs   'USERID.STLIN'
      Updated networks . . . . 'USERID.TESTFILE'
      Sequential output  . . . 'USERID.SEQOUT'
      Printer output . . . . . 'USERID.SYSPRINT'



Command ===>
F1=Help  F2=Split  F3=Exit  F4=Edit input  F5=Refresh  F6=Browse prt
F7=Bkwd  F8=Fwd    F9=Swap F10=Edit ctl   F11=Save    F12=Cancel
```

ITPSGEN Control Commands

This section describes new ITPSGEN control commands and explains how existing control commands are affected by the TPNS CPI-C script generation enhancement. There are five new control commands specifically for CPI-C scripts. All of the existing control commands are recognized by the CPI-C script generator. However, the RESP and SSCP commands are ignored during generation of CPI-C scripts. Refer to *TPNS Script Generating Utilities* for complete information on existing control commands.

COMP and NOCOMP

```
COMP   {ALL  [n] [WARNING | ERROR]}
       {DATA [n] [WARNING | ERROR]}
       {CONV     [WARNING | ERROR]}
```

Default: `COMP ALL 32767 ERROR`

This command specifies the level of comparison to be performed when comparing the actual send data length, and the actual received data, status, and conversation characteristics to the trace file data. Received status comparisons controlled by this command are: send received, confirm received, and conversation deallocated. Conversation characteristic comparisons controlled by this command are: partner LU name, mode name, conversation type, and conversation sync-level. This command also controls whether conversation failures are handled as errors or warnings. Warning conditions result in a message being issued, and the script continues to process. Error conditions result in a message being issued, however, the script is terminated.

COMP ALL [*n*][*WARNING* | *ERROR*]
The ALL operand compares send data length, received data, and received status, as well as conversation characteristics. Use *n* to specify the amount of received data to be compared. The range for *n* is 0 to 32767. The default value is 32767. When this control command is specified, the actual send data length is compared to the send data length from the trace. Also, the first *n* bytes of actual received data from each receive verb will be compared against the data from the trace. The actual received data length is also compared to the received data length from the trace. However, if *n* is less than the received data length from the trace, the comparison will only ensure that the actual received data is at least as large as *n*. If *warning* is specified, the generated script handles conversation failures by issuing a message and continuing with the next statement in the script. If *error* is specified, the generated script handles conversation failures by issuing a message and terminating the script.

COMP DATA [*n*][*WARNING* | *ERROR*]
The DATA operand compares only the received data to the trace file data. Use *n* to specify the amount of received data to be compared. The range for *n* is 0 to 32767. The default value is 32767. When this control command is specified, the first *n* bytes of actual received data from each receive verb will be compared against the data from the trace. The actual received data length also will be compared to the received data length from the trace. However, if *n* is less than the received data length from the trace, the comparison will only ensure that the actual received data is at least as large as *n*. If *warning* is specified, the generated script handles conversation failures by issuing a message and continuing with the next statement in the script. If *error* is specified, the generated script handles conversation failures by issuing a message and terminating the script.

COMP CONV [*WARNING* | *ERROR*]
The CONV operand compares only the conversation characteristics and received status to the trace file data. If *warning* is specified, the generated script handles conversation failures by issuing a message and continuing with the next statement in the script. If *error* is specified, the generated script handles conversation failures by issuing a message and terminating the script.

NOCOMP
The NOCOMP control command specifies that no comparisons are to be performed on send data length, received data, status, or conversation characteristics.

FIELD and NOFIELD

This command is used to control the creation of composite fields. Composite fields can be optionally generated for the attach request (FMH-5) extension and the send and receive buffers. Creating composite fields refers to mapping the variable length fields in the FMH-5 extension and the send and receive buffers into component pieces and creating STL variables to represent each component. Components that are length fields are scripted in such a way as to support dynamic recalculation when data field components are modified. For example, you can change the fully qualified LU name in the FMH-5 extension by changing the STL variable that represents this value. The FMH-5 length fields affected by the change are automatically recalculated by the script.

FIELD
FIELD specifies that the variable length fields in the FMH-5 extension and the send and receive buffers should be mapped into composite fields in the generated script. FIELD is the default value.

NOFIELD
NOFIELD specifies that the variable length fields in the FMH-5 extension and the send and receive buffers should not be mapped into composite fields in the generated script.

NOFIELD Example

The following is an example of the generated script for an FMH-5 extension, send buffer, and receive buffer. This example was generated without composite fields.

```
FMH5_extension = ,
 /* 0000 */  '100702C4E2E4E2C5D90701C4E2E4E2C5D91A11C3E6E2D5C5E3'x||,
 /* EBCDIC:   . . . D S U S E R . . D S U S E R . . C W S N E T      */
 /* 0019 */  'C3C14BC3E6F9F0F0C5F4C9AE08BF98C643000108B619DF7307'x||,
 /* EBCDIC:   C A . C W 9 0 0 E 4 I . . . q F . . . . . . . . .      */
 /* 0032 */  '20E813'x
 /* EBCDIC:   . Y .                                                  */
FMH5_extension_length = length(FMH5_extension)

send_buffer = ,
 /* 0000 */  '001A12210016FF000901E3C5E2E3F1F1C10902C2E4C4F3C2E4'x||,
 /* EBCDIC:   . . . . . . . . . . T E S T 1 1 A . . B U D 3 B U      */
 /* 0019 */  'C4'x
 /* EBCDIC:   D                                                      */
send_length = length(send_buffer)
CMSEND(conversation_ID,send_buffer,send_length,,
       request_to_send_received,return_code)

expected_receive_buffer = ,
 /* 0000 */  '002D12210029FF020300000A0207CD01070F3A29040A0307CC'x||,
 /* EBCDIC:   . . . . . . . . . . . . . . . . . . . . . . . . .      */
 /* 0019 */  '0C171630223D0A0407CD050400000000004050000'x
 /* EBCDIC:   . . . . . . . . . . . . . . . . . . . .                */
```

FIELD Example

The following is an example of the generated script for an FMH-5 extension, send buffer, and receive buffer. This example was generated with composite fields.

```
/* Create FMH-5 extension */
/* Note: If any data field in the FMH-5 extension is modified, all   */
/*       affected length fields will be dynamically recalculated.    */

/* Build access security information */
fm5asipr = '00'x                /* Profile field constant          */
fm5asipw = '01'x                /* Password field constant         */
fm5asiid = '02'x                /* User ID field constant          */
fm5asty1 = fm5asiid             /* User ID security subfield       */
fm5asda1 = 'C4E2E4E2C5D9'x      /* Security subfield data          */
 /* EBCDIC: D S U S E R                                             */
fm5asll1 = ,                    /* Security subfield length        */
           hex(length(fm5asty1)+length(fm5asda1))
fm5asi1  = ,                    /* Composite security subfield     */
           fm5asll1||fm5asty1||fm5asda1
fm5asty2 = fm5asipw             /* Password security subfield      */
fm5asda2 = 'C4E2E4E2C5D9'x      /* Security subfield data          */
 /* EBCDIC: D S U S E R                                             */
fm5asll2 = ,                    /* Security subfield length        */
           hex(length(fm5asty2)+length(fm5asda2))
fm5asi2  = ,                    /* Composite security subfield     */
           fm5asll2||fm5asty2||fm5asda2
fm5accse = fm5asi1||fm5asi2
fm5lnasi = hex(length(fm5accse))  /* Access Security Info length   */
fm5asi   = fm5lnasi||fm5accse   /* Composite Access Security Info  */

/* Build logical unit of work fields */
fm5fqnam = ,                    /* Fully Qualified Name            */
 /* 0000 */ 'C3E6E2D5C5E3C3C14BC3E6F9F0F0C5F4C9'x
 /* EBCDIC:  C W S N E T C A . C W 9 0 0 E 4 I                      */
fm5lnfqn = hex(length(fm5fqnam))  /* Fully Qualified Name length   */
fm5luwi  = fm5lnfqn||fm5fqnam   /* Composite LUOW ID               */
fm5luwin = 'AE08BF98C643'x      /* LUOW Instance Number            */
 /* EBCDIC: . . . q F .                                             */
fm5luwsn = '0001'x              /* LUOW Sequence Number            */
 /* EBCDIC: . .                                                     */
fm5luow2 = fm5luwin||fm5luwsn   /* Composite LUOW instance/seq     */
fm5lnluw = ,                    /* Logical Unit of Work length     */
           hex(length(fm5lnfqn)+length(fm5fqnam)+length(fm5luow2))
fm5luow1 = fm5lnluw||fm5luwi    /* Composite Logical Unit of Work  */

/* Build conversation correlator */
fm5ccs   = 'B619DF730720E813'x  /* Conversation Correlator data    */
 /* EBCDIC: . . . . . . Y .                                         */
fm5lnccs = hex(length(fm5ccs))  /* Conversation Correlator length  */
fm5cvcor = fm5lnccs||fm5ccs     /* Composite Conv Correlator       */

FMH5_Extension = fm5asi||fm5luow1||fm5luow2||fm5cvcor
 /* 0000    '100702C4E2E4E2C5D90701C4E2E4E2C5D91A11C3E6E2D5C5E3'x||, */
 /* EBCDIC:  . . . D S U S E R . . D S U S E R . . C W S N E T      */
 /* 0019    'C3C14BC3E6F9F0F0C5F4C9AE08BF98C643000108B619DF7307'x||, */
 /* EBCDIC:  C A . C W 9 0 0 E 4 I . . . q F . . . . . . . . .      */
 /* 0032    '20E813'x                                               */
 /* EBCDIC:  . Y .                                                  */
FMH5_extension_length = length(FMH5_extension)

/* Note: If the senddata field is modified, all affected length      */
/*       fields will be dynamically recalculated.                    */
sendid   = '1221'x              /* Probable ID field               */
 /* EBCDIC: . .                                                     */
senddata = ,
 /* 0000 */ '0016FF000901E3C5E2E3F1F1C10902C2E4C4F3C2E4C4'x
 /* EBCDIC:  . . . . . . T E S T 1 1 A . . B U D 3 B U D            */
```

```
sendll = hex(length(sendid)+length(senddata)+2,2)
send_buffer = sendll||sendid||senddata /* Send buffer data          */
send_length = length(send_buffer)
CMSEND(conversation_ID,send_buffer,send_length,,
       request_to_send_received,return_code)

/* Note: If the expected_receive_data field is modified, all         */
/*       affected length fields will be dynamically recalculated.    */
expected_receive_id = '1221'x     /* Probable ID field               */
 /* EBCDIC:            . .                                           */
expected_receive_data = ,
 /* 0000 */ '0029FF020300000A0207CD01070F3A29040A0307CC0C17163022'x||,
 /* EBCDIC:  . . . . . . . . . . . . . . . . . . . . . . . . . .    */
 /* 001A */ '3D0A0407CD0504000000000004050000'x
 /* EBCDIC:  . . . . . . . . . . . . . . . .                         */
expected_receive_ll = ,           /* Expected receive buffer length  */
  hex(length(expected_receive_id)+length(expected_receive_data)+2,2)
expected_receive_buffer = ,       /* Expected receive buffer data    */
  expected_receive_ll||expected_receive_id||expected_receive_data
```

HEXON and NOHEXON

This command is used to force certain data buffers to be generated in displayable hexadecimal form if they exceed a specified length threshold. The buffers controlled by this command are: *send data*, *receive comparison data*, *error log data*, and *FMH-5 extension data*. Optionally, the ASCII and/or EBCDIC translations of the displayable hexadecimal data can be echoed in the form of STL comments. This control command can be useful to facilitate the analysis of the data buffers in the generated script when viewing them in a printout or on a video display screen. Also, if long ASCII data streams are produced in the script, this command can be used to avoid the generation of excessively complex STL statements that exceed the capacity of the STL Translator.

HEXON ALL [*ASCII* | *EBCDIC* | *BOTH***]**
The ALL operand specifies that all *send data*, *receive comparison data*, *error log data*, and *FMH-5 extension data* will be generated in displayable hexadecimal. You may optionally use *ASCII*, *EBCDIC*, or *BOTH* to specify that the data will be followed with STL comments that echo the data in ASCII, EBCDIC, or both.

HEXON *n* **[***ASCII* | *EBCDIC* | *BOTH***]**
The *n* operand specifies that all *send data*, *receive comparison data*, *error log data*, and *FMH-5 extension data* should be generated in displayable hexadecimal if the data length is *n* bytes or more. Data streams less than *n* bytes in length will be generated as a combination of displayable character data and displayable hexadecimal as required for effective display of the data on a video screen. The value for *n* can range from 1 to 32767. You may optionally use *ASCII*, *EBCDIC*, or *BOTH* to specify that for data streams greater than *n* bytes in length, the data will be followed with STL comments that echo the data in ASCII, EBCDIC, or both.

NOHEXON
NOHEXON is the default for this control command. The NOHEXON command generates all *send data*, *receive comparison data*, *error log data*, and *FMH-5 extension data* as a combination of displayable character data and displayable hexadecimal as required for effective display of the data on a video screen.

SENDL and NOSENDL

```
SENDL [WARNING | ERROR]
```

Default: `SENDL WARNING`

This command is used to determine whether a generated script should report differences in send length compared to the trace file. Reported differences can be treated as warning conditions or error conditions. If *warning* is specified, the generated script handles length differences by issuing a message and continuing with the next statement in the script. If *error* is specified, the generated script handles length differences by issuing a message and terminating the script.

SENDL [*WARNING* | *ERROR*]
SENDL specifies that the generated script should report differences in the actual send length compared to the trace file.

NOSENDL
NOSENDL specifies that the generated script should not report differences in the actual send length compared to the trace file.

UCD and NOUCD

The UCD and NOUCD control commands are used to specify whether user control data should be processed or ignored.

UCD
UCD is the default for this control command. The UCD control command specifies that any user control data encountered in the trace should be processed as if they were application data. Also either UCD=YES or UCD=BOTH is added to the TP statement in the model network. If any true application data is encountered, UCD=BOTH is added. Otherwise, UCD=YES is added. Message ITP868I is issued to the SYSPRINT data set the first time user control data is encountered for a given TP. Refer to "Appendix B. Messages and Return Codes" on page 105 for more information about the message.

NOUCD
This control command specifies that any user control data encountered in the trace should be ignored. Also, UCD=NO is added to the TP statement in the model network. Message ITP867I is issued to the SYSPRINT data set the first time user control data is encountered for a given TP. Refer to "Appendix B. Messages and Return Codes" on page 105 for more information about the message.

Existing Control Commands

All of the current control commands are recognized by the CPI-C script generator. However, the RESP and SSCP commands are ignored during generation of CPI-C scripts. Refer to *TPNS Script Generating Utilities* for information on the RESP and SSCP commands and for an explanation of other existing control commands.

The DELAY command is handled differently when generating CPI-C scripts. Delays are calculated from the time of the last receive record or from the time of the last transmit record, whichever is the most recent. A delay is calculated for all CPI-C verbs that are TPNS delimiters (that is, verbs that cause a transmit to VTAM). These verbs are: CMALLC, CMCFM, CMCFMD, CMDEAL, CMFLUS, CMPTR, CMRTS, CMSEND, and CMSERR. The delay is generated in the CPI-C script as an STL SUSPEND statement, and is inserted at the point where the delay was encountered in the trace. When running the generated script, the delays should approximate what actually transpired when the trace was captured. However, no attempt is made to control any delays in responses from the partner. In addition to the SUSPEND statement, a DELAY(0) statement is generated prior to each CPI-C delimiter. This overrides any default delay that may have been set in the network definition, thus preventing a cumulative delay effect.

Changes to Reports

A *trace records eligible* field has been added to the detail and summary reports. This field contains a count of the total number of input trace records that were eligible for script generation processing.

Summary Report

The *trace records eligible* count will appear on the summary report for each network defined in the model network. An example of a summary report containing the *trace records eligible* field is shown below:

```
ITPSGEN GENERATE OUTPUT                                      TIME 13.45.09,     JUNE  3, 2001   PAGE     44
                       GENERATION REPORT - SUMMARY

               TERMINALS        TRACE RECORDS        MSGTXTS          LIMIT          PATHS
 NETWORK       ELIGIBLE           ELIGIBLE          GENERATED        REACHED         ADDED

 NET1                 3                 97                5               0              3
 NET2                 3                  0                0               0              0
```

Detail Report

The *trace records eligible* count will appear on the detail report for each terminal or resource in the model network. An example of a detail report containing the *trace records eligible* field is shown below:

```
ITPSGEN GENERATE OUTPUT                                      TIME 13.45.09,     JUNE  3, 2001   PAGE     45
                       GENERATION REPORT - DETAIL

                         TRACE RECORDS     MESSAGES       START           STOP
TERMINAL      NETWORK       ELIGIBLE       GENERATED      TIME            TIME          PATH

ITPECHO       NET1                77             37     16:09:05.80     16:09:42.21     ITPECHO
ITPECHO       NET2                 0              0
NET1TP1       NET1                 9              5     08:30:10.94     08:30:20.72     NET1TP1
NET2TP1       NET2                 0              0
NET1TP2       NET1                11              3     08:30:10.94     08:30:20.73     NET1TP2
NET2TP2       NET2                 0              0
```

SEQOUT Data Set

If the SEQOUT control command is specified, ITPSGEN produces a SEQOUT data set as output. If you are generating CPI-C scripts, the SEQOUT data set will contain network definition statements and STL code. The STL must be translated to the TPNS scripting language before you can run a simulation. If you generate scripts for other simulation types in the same run in which you generate CPI-C scripts, the SEQOUT data set will also contain the TPNS scripting language message decks for the other simulation types.

Note: SEQOUT data sets for CPI-C simulations are structured differently than SEQOUT data sets for other simulation types. The following are some of the differences of which you should be aware:

1. In the SEQOUT data set for a CPI-C simulation, the network definition is surrounded by the @NET and @ENDNET statements to inform the STL Translator to pass it to the TPNS preprocessor.

2. If TPNS scripting language message decks are generated during the same run that produces STL, the TPNS scripting language message decks will appear following the network definition and before the @ENDNET.
3. As with other types of script generation, the network definition will be updated to reflect the generated scripts if the NTWRK control command is specified. For CPI-C networks, this update involves adding the PATH statements and adding the PATH, TPTYPE, TPNAME, INSTANCE, and UCD operands to the TP statements.
4. The *allocate* and *receive* CPI-C functions are consolidated into separate STL message texts that are invoked as needed.

Sample SEQOUT Data Set

The following is a sample SEQOUT data set. Note that this data set contains STL that was generated by the script generator for a CPI-C simulation.

```
@NET
CPICSGEN NTWRK CONRATE=YES,             * Message rates to the console
               OPTIONS=(MONCMND),       * Monitor operator commands
               CPITRACE=VERB,           * Trace CPI-C verbs
               PATH=(0),                * Default path statement
               HEAD='CPI-C NETWORK'
*
*  Model network for CPI-C script generation.
*
CLIENT   PATH    CLIENT
SERVER   PATH    SERVER
0        PATH    SGENTXT
*
LU1      APPCLU
CLIENT   TP      PATH=(CLIENT),TPTYPE=CLIENT,TPNAME=CLIENT,INSTANCE=(1,1),UCD=NO
*
LU2      APPCLU
SERVER   TP      PATH=(SERVER),TPTYPE=SERVER,TPNAME=SERVER,INSTANCE=(0,1),UCD=NO
*
@ENDNET

@include cpicvara
@include cpiccon

CLIENT: MSGTXT
/*-------------------------------------------------------------------*/
/* ITPSGEN: RESOURCE=CLIENT     SESSION=00002    12:49:42.00 06/04/01 */
/*-------------------------------------------------------------------*/
/*  ----------------- CONVERSATION CHARACTERISTICS -----------------  */
/*  CONVERSATION NUMBER: 00001    CONVERSATION DIRECTION:   OUTBOUND  */
/*  CONVERSATION TYPE:  MAPPED    CONVERSATION SYNC-LEVEL:   CONFIRM  */
/*  MODE NAME:          #INTER    PARTNER LU NAME:               LU1  */
/*  PARTNER TP NAME:    SERVER    HEX TP NAME:       'E2C5D9E5C5D9'X  */
/*-------------------------------------------------------------------*/

conversation_error = on       /* Init conversation error switch  */

do forever                    /* Start do forever loop           */

 /*----- Allocate a conversation with the remote LU/TP -----*/
 partner_LU_name = 'LU1'
 mode_name = '#INTER'
 TP_name = 'SERVER'
 conversation_type = cm_mapped_conversation
 sync_level = cm_confirm
 call @CMALLC                 /* allocate conversation           */
 if return_code ¬= cm_ok then leave

 /*----- Send data to the remote TP -----*/
 send_type = cm_send_and_confirm
 CMSST(conversation_ID,send_type,return_code)
 if return_code ¬= cm_ok then leave

 prepare_to_receive_type = cm_prep_to_receive_sync_level
 CMSPTR(conversation_ID,prepare_to_receive_type,return_code)
 if return_code ¬= cm_ok then leave

 deallocate_type = cm_deallocate_sync_level
 CMSDT(conversation_ID,deallocate_type,return_code)
 if return_code ¬= cm_ok then leave

 send_buffer = ,
  /* 0000 */  'This is the data that is sent from the client trans'||,
  /* 0033 */  'action program to the server transaction program.  '||,
  /* 0066 */  'Here is some hex data followed by some more printab'||,
  /* 0099 */  'le EBCDIC: '‖'0102030405060708090A0B0C'x‖'End of '‖,
  /* 00B7 */  'data.'
 send_length = length(send_buffer)
 CMSEND(conversation_ID,send_buffer,send_length,,
        request_to_send_received,return_code)
 if return_code ¬= cm_ok then leave

 /*----- Deallocate the conversation -----*/
 deallocate_type = cm_deallocate_confirm
 CMSDT(conversation_ID,deallocate_type,return_code)
 if return_code ¬= cm_ok then leave
 CMDEAL(conversation_ID,return_code)
 if return_code ¬= cm_ok then leave

 /*----- Normal end of communications -----*/
 conversation_error = off     /* No conv error, reset switch     */
 leave                        /* Always leave do forever loop    */

end                           /* End do forever loop             */

if conversation_error = on then say 'CPI-C Conversation Error!!!'

ENDTXT

SERVER: MSGTXT
/*--------------------------------------------------------------------*/
/* ITPSGEN: RESOURCE=SERVER     SESSION=00002    12:49:42.00 06/04/01 */
/*--------------------------------------------------------------------*/
/*  ----------------- CONVERSATION CHARACTERISTICS -----------------  */
/*  CONVERSATION NUMBER: 00001    CONVERSATION DIRECTION:    INBOUND  */
/*  CONVERSATION TYPE:  MAPPED    CONVERSATION SYNC-LEVEL:   CONFIRM  */
/*  MODE NAME:          #INTER    PARTNER LU NAME:               LU2  */
/*  LOCAL TP NAME:      SERVER    HEX TP NAME:      'E2C5D9E5C5D9'X  */
/*--------------------------------------------------------------------*/

conversation_error = on       /* Init conversation error switch  */

do forever                    /* Start do forever loop           */

 /*----- Accept conversation from remote LU/TP -----*/
 CMACCP(conversation_ID,return_code)
 if return_code ¬= cm_ok then leave

 CMEPLN(conversation_ID,partner_LU_name,,
        partner_LU_name_length,return_code)
 if return_code ¬= cm_ok then leave
 if partner_LU_name ¬= 'LU2' then leave

 CMEMN(conversation_ID,mode_name,mode_name_length,return_code)
 if return_code ¬= cm_ok then leave
 if mode_name ¬= '#INTER' then leave

 CMECT(conversation_ID,conversation_type,return_code)
 if return_code ¬= cm_ok then leave
 if conversation_type ¬= cm_mapped_conversation then leave

 CMESL(conversation_ID,sync_level,return_code)
 if return_code ¬= cm_ok then leave
 if sync_level ¬= cm_confirm then leave

 /*----- Receive data from the remote TP -----*/
 receive_type = cm_receive_and_wait
 CMSRT(conversation_ID,receive_type,return_code)
 if return_code ¬= cm_ok then leave
 call @CMRCV                  /* receive data from the remote TP */
 if return_code ¬= cm_ok then leave
 if send_received = on then leave
 if confirm_received = off then leave
 if conversation_deallocated = on then leave
 expected_receive_buffer = ,
  /* 0000 */  'This is the data that is sent from the client trans'‖,
  /* 0033 */  'action program to the server transaction program.  '‖,
  /* 0066 */  'Here is some hex data followed by some more printab'‖,
  /* 0099 */  'le EBCDIC: '‖'0102030405060708090A0B0C'x‖'End of '‖,
  /* 00B7 */  'data.'
 if received_length ¬= 188 then leave
 if substr(receive_buffer,1,188) ¬= ,
    substr(expected_receive_buffer,1,188) then leave

 /*----- Send a confirmation reply -----*/
 CMCFMD(conversation_ID,return_code)
 if return_code ¬= cm_ok then leave

 /*----- Receive data from the remote TP -----*/
 call @CMRCV                  /* receive data from the remote TP */
 if return_code ¬= cm_ok then leave
 if send_received = on then leave
 if confirm_received = off then leave
 if conversation_deallocated = off then leave

 /*----- Send a confirmation reply -----*/
 CMCFMD(conversation_ID,return_code)
 if return_code ¬= cm_ok then leave

 /*----- Normal end of communications -----*/
 conversation_error = off     /* No conv error, reset switch     */
 leave                        /* Always leave do forever loop    */

end                           /* End do forever loop             */

if conversation_error = on then say 'CPI-C Conversation Error!!!'

ENDTXT

@CMALLC: MSGTXT
/*-----------------------------------------------------------*/
/*----- Allocate a conversation with the remote LU/TP -----*/
/*-----------------------------------------------------------*/

/* Initialize the conversation */
sym_dest_name = ' '
CMINIT(conversation_ID,sym_dest_name,return_code)
if return_code ¬= cm_ok then return

/* Set the partner LU name */
partner_LU_name_length = length(partner_LU_name)
CMSPLN(conversation_ID,partner_LU_name,,
       partner_LU_name_length,return_code)
if return_code ¬= cm_ok then return

/* Set the mode name */
mode_name_length = length(mode_name)
CMSMN(conversation_ID,mode_name,mode_name_length,return_code)
if return_code ¬= cm_ok then return

/* Set the TP name */
TP_name_length = length(TP_name)
CMSTPN(conversation_ID,TP_name,TP_name_length,return_code)
if return_code ¬= cm_ok then return

/* Set the conversation type */
CMSCT(conversation_ID,conversation_type,return_code)
if return_code ¬= cm_ok then return
```

```
/* Set the synchronization level */
CMSSL(conversation_ID,sync_level,return_code)
if return_code ¬= cm_ok then return

/* Allocate the conversation */
CMALLC(conversation_ID,return_code)

ENDTXT

@CMRCV: MSGTXT
/*-------------------------------------------*/
/*----- Receive data from the remote TP -----*/
/*-------------------------------------------*/

/* Initialize the receive parameter values */
receive_buffer = ''
requested_length = 32767
received_length = 0
data_received = cm_no_data_received

/* Initialize the receive status flags */
conversation_deallocated = off
confirm_received = off
send_received = off

/* Issue the receive request */
CMRCV(conversation_ID,receive_buffer,requested_length,,
      data_received,received_length,status_received,,
      request_to_send_received,return_code)

/* Examine the receive results and set the status flags */
select
 when return_code = cm_ok then
  select
   when status_received = cm_send_received then send_received = on
   when status_received = cm_confirm_received then confirm_received = on
   when status_received = cm_confirm_send_received then
    do
     confirm_received = on
     send_received = on
    end
   when status_received = cm_confirm_dealloc_received then
    do
     confirm_received = on
     conversation_deallocated = on
    end
   otherwise nop
  end
 when return_code = cm_deallocated_normal then
  do
   return_code = cm_ok
   conversation_deallocated = on
  end
 otherwise nop
end

ENDTXT
```

STL Translation

CPI-C scripts are generated by ITPSGEN in STL. Before you can run a CPI-C script generated by ITPSGEN, you must translate the STL into TPNS scripting language by following the steps given below:

1. The STL input data set must point to the data set produced by the script generation SEQOUT DD. See “Sample JCL for STL Translation” on page 19.

2. Add the CPICVARA and the CPICCON STL include files as members in your STL includes data set (SYSLIB DD) if they are not already in this data set. Both these files are members in the TPNS SAMPLE data set. CPICVARA is a new include file added as part of the CPI-C script generation support. See “CPICVARA STL Include File” on page 20.

Sample JCL for STL Translation

You can use the following sample when writing your own JCL to execute the TPNS STL Translator.

```
//STLJOB   JOB
//*********************************************************************
//* Teleprocessing Network Simulator (TPNS)   5688-121               *
//*********************************************************************
//*                      STLJOB JCL                                   *
//* Sample JCL to execute the TPNS STL Translator (ITPSTL).          *
//*********************************************************************
//STL      EXEC PGM=ITPSTL,REGION=4096K
//STEPLIB  DD  DSN=TPNS.LOAD,DISP=SHR
//PARMDD   DD  DSN=TPNS.PARMDD,DISP=SHR
//RATEDD   DD  DSN=TPNS.RATETBLS,DISP=SHR
//INITDD   DD  DSN=TPNS.TESTFILE,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//MSGDD    DD  DSN=TPNS.MSGFILE,DISP=SHR
//SEQOUT   DD  DSN=TPNS.STL.SEQOUT,DISP=SHR
//SYSLIB   DD  DSN=TPNS.STLIN,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSIN    DD  DSN=TPNS.SEQOUT,DISP=SHR
```

CPICVARA STL Include File

Before translating the STL, add the following CPICVARA STL include file as a member in your STL includes data set (SYSLIB DD) if it is not already a member of that data set.

```
/*******************************************************************/
/* STL variable allocations for CPI-C verb parameters & switches    */
/*******************************************************************/
/* String parameters                                                */
allocate conversation_ID          '1'       /* Conversation ID      */
allocate mode_name                '2'       /* Mode name            */
allocate partner_LU_name          '3'       /* Partner LU name      */
allocate sym_dest_name            '4'       /* Symbolic dest name   */
allocate TP_name                  '5'       /* TP name              */
allocate log_data                 '6'       /* Log data             */
allocate send_buffer              '7'       /* Send buffer          */
allocate receive_buffer           '8'       /* Receive buffer       */
allocate FMH5_extension           '9'       /* FMH-5 extension      */
/*******************************************************************/
/* Integer parameters                                               */
allocate conversation_state       'DC1'     /* Conversation state   */
allocate conversation_type        'DC2'     /* Conversation type    */
allocate data_received            'DC3'     /* Data received        */
allocate deallocate_type          'DC4'     /* Deallocate type      */
allocate error_direction          'DC5'     /* Error direction      */
allocate fill                     'DC6'     /* Fill value           */
allocate log_data_length          'DC7'     /* Log data length      */
allocate mode_name_length         'DC8'     /* Mode name length     */
allocate partner_LU_name_length   'DC9'     /* Partner LU name length*/
allocate prepare_to_receive_type  'DC10'    /* Prepare to RCV type  */
allocate receive_type             'DC11'    /* Receive type         */
allocate received_length          'DC12'    /* Received length      */
allocate request_to_send_received 'DC13'    /* Request-to-send rcvd  */
allocate requested_length         'DC14'    /* Requested length     */
allocate return_code              'DC15'    /* Return code          */
allocate return_control           'DC16'    /* Return control       */
allocate send_length              'DC17'    /* Send length          */
allocate send_type                'DC18'    /* Send type            */
allocate status_received          'DC19'    /* Status received      */
allocate sync_level               'DC20'    /* Sync-level           */
allocate TP_name_length           'DC21'    /* TP name length       */
allocate FMH5_extension_length    'DC22'    /* FMH-5 extension length*/
/*******************************************************************/
/* Device switches                                                  */
allocate send_received            'SW1'     /* Send token received   */
allocate confirm_received         'SW2'     /* Confirmation req rcvd */
allocate conversation_deallocated 'SW3'     /* Conv deallocated     */
allocate conversation_error       'SW4'     /* Conversation error   */
/*******************************************************************/
```

VTAM System Definitions

Prior to running a TPNS simulation using generated CPI-C scripts, make sure each APPC LU to be simulated by TPNS is defined to VTAM via APPL statements in the VTAMLST data set. The definition must specify APPC=YES. Each APPC LU must have a unique APPLID name. This is necessary since VTAM associates conversations to an APPLID name and receives attach requests by this name, rather than by the TP name.

Chapter 2. TPNS STL Script Generation Support

The TPNS Script Generator Utility has been enhanced with an option to generate scripts in Structured Translator Language (STL). In TPNS 3.5 Service Level 9711, CPI-C scripts are generated exclusively in STL. However, other types of scripts (VTAMAPPL LU0 or LU2 for instance) are generated in TPNS scripting language. This new support provides a user option for all types of script generation except CPI-C. CPI-C scripts are still exclusively generated in STL. All other types of scripts can be generated in either TPNS scripting language or STL. The target language for the script generator output is controlled by the following new control commands.

STL and NOSTL

This command is used to determine whether scripts are to be generated in the Structured Translator Language (STL). Previously, CPI-C scripts were generated exclusively in STL. However, other types of scripts (VTAMAPPL LU0 or LU2, for example) are generated in TPNS scripting language. This command provides you with an option for all types of script generation except CPI-C. CPI-C scripts are still exclusively generated in STL. All other types of scripts can be generated in either TPNS scripting language or STL.

STL
STL specifies that the target language for script generator output is Structured Translator Language (STL).

NOSTL
NOSTL specifies that the target language for script generator output is TPNS scripting language. NOSTL is the default value.

Defining the Network

The model network is defined the same for generating STL as for generating TPNS scripting language. Refer to *TPNS Script Generating Utilities* for detailed information. Note that if scripts are included with the model network they must be written in TPNS scripting language. Any scripts included with the model network are copied to the generated message decks data set during the script generation process.

Changes to JCL, EXECs, and CLISTs

Before using ITPSGEN to generate scripts in STL, you must add an STLTXT DD definition to the existing script generation JCL, EXEC, or CLIST. This DD is for an output partitioned data set that will contain the STL source. Each STL MSGTXT will be a member in this partitioned data set. The following sample JCL, EXEC, and CLIST illustrate the addition of the STLTXT DD definition.

Sample JCL

The example below shows a JCL that has been modified to enable ITPSGEN to generate scripts in STL.

```
//SGENJOB  JOB
//**********************************************************************
//* Teleprocessing Network Simulator (TPNS)   5688-121                 *
//**********************************************************************
//*                      SGENJOB JCL                                   *
//* Sample JCL to execute ITPSGEN.                                     *
//**********************************************************************
//JOBLIB   DD  DSNAME=TPNS.LOAD,DISP=SHR
//STEP1    EXEC PGM=ITPSGEN,PARM='CTL'
//RATEDD   DD  DSNAME=TPNS.RATETBLS,DISP=SHR
//INITDD   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//MSGDD    DD  DSNAME=TPNS.MSGFILE,DISP=SHR
//SYSUT2   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSPRINT DD  SYSOUT=A
//MSGTXT   DD  DSNAME=TPNS.MSGFILE,DISP=SHR
//STLTXT   DD  DSNAME=TPNS.STLIN,DISP=SHR
//NTWRK    DD  DSNAME=TPNS.TESTFILE,DISP=SHR
//SEQOUT   DD  DSNAME=TPNS.SEQOUT,DISP=SHR
//TAPEIN   DD  UNIT=TAPE,DISP=OLD,VOL=SER=TAPEIN,LABEL=(,NL)
//CTLIN    DD  *
* CONTROL STATEMENTS FOR ITPSGEN SCRIPT GENERATOR *
LIST
SEQOUT
NTWRK
REPORT FULL
/*
//SYSIN    DD  *
CPICSGEN NTWRK CONRATE=YES,             * Message rates to the console
               OPTIONS=(MONCMND),       * Monitor operator commands
               CPITRACE=VERB,           * Trace CPI-C verbs
               PATH=(0),                * Default path statement
               HEAD='CPI-C NETWORK'
*
*  Model network for CPI-C script generation.
*
0        PATH    SGENTXT
*
LU1      APPCLU
CLIENT   TP
*
LU2      APPCLU
SERVER   TP
*
SGENTXT  MSGTXT
         WAIT
         ENDTXT
/*
```

Sample REXX EXEC

The example below shows a REXX EXEC that has been modified to enable ITPSGEN to generate scripts in STL.

```
/*******************************************************************
 **  Teleprocessing Network Simulator (TPNS)   5688-121            **
 *******************************************************************
 **                   SGEN EXEC                                    **
 **                                                                **
 **  Sample EXEC to execute ITPSGEN on CMS.  It can also be used to **
 **  execute ITPSGEN on GCS if the file type is changed from EXEC   **
 **  to GCS and the two ERASE commands at the end are removed.      **
 **                                                                **
 **  Format:                                                       **
 **          SGEN prepin_name prepin_type trace_name trace_type    **
 **                                                                **
 *******************************************************************/
arg prep_fn prep_ft trace_fn trace_ft .
'FILEDEF * CLEAR'
'FILEDEF CTLIN    DISK TPNS     SGENCNTL *'
'FILEDEF INITDD   DISK TPNS     TESTFILE A'
'FILEDEF MSGDD    DISK TPNS     MSGFILE  A'
'FILEDEF MSGTXT   DISK TPNS     MSGFILE  A'
'FILEDEF STLTXT   DISK TPNS     STLIN    A'
'FILEDEF NTWRK    DISK TPNS     TESTFILE A'
'FILEDEF RATEDD   DISK TPNS     RATETBLS * (BLKSIZE 2000)'
'FILEDEF SEQOUT   DISK TPNS     SEQOUT   A'
'FILEDEF SYSIN    DISK' prep_fn  prep_ft  '*'
'FILEDEF SYSPRINT PRINTER'
'FILEDEF TAPEIN   DISK' trace_fn trace_ft '*'
'GLOBAL  LOADLIB  TPNSLOAD'
'OSRUN   ITPSGEN'
rcode=rc
'ERASE FILE SYSUT2 A'
'ERASE FILE SYSUT3 A'
exit rcode
```

Sample TSO CLIST

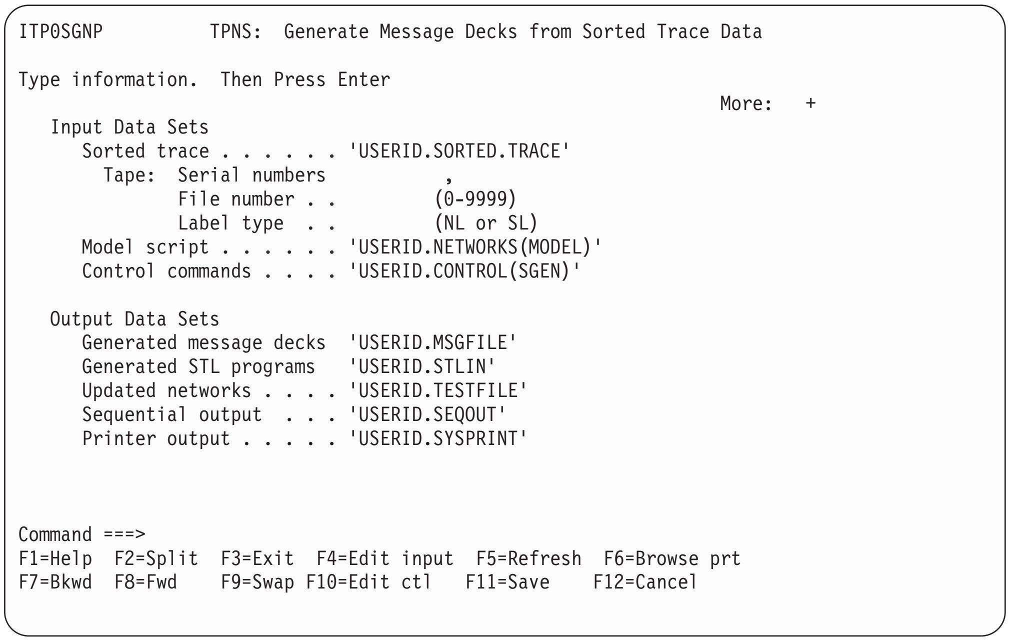
The example below shows a CLIST that has been modified to enable ITPSGEN to generate scripts in STL.

```
/*******************************************************************/
/* Teleprocessing Network Simulator (TPNS)   5688-121              */
/*******************************************************************/
/*                      SGEN CLIST                                 */
/* Sample CLIST to execute ITPSGEN.                                */
/*******************************************************************/
FREE  DDNAME(SYSPRINT RATEDD INITDD MSGDD SYSUT2 SYSUT3)
FREE  DDNAME(MSGTXT STLTXT NTWRK SEQOUT TAPEIN CTLIN SYSIN)
ALLOC DDNAME(SYSPRINT) SYSOUT(A)
ALLOC DDNAME(RATEDD)   DATASET('TPNS.RATETBLS') SHR
ALLOC DDNAME(INITDD)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(MSGDD)    UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(SYSUT2)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(SYSUT3)   UNIT(SYSDA) SPACE(10,10) DIR(3) TRACKS
ALLOC DDNAME(MSGTXT)   DATASET('TPNS.MSGFILE')   SHR
ALLOC DDNAME(STLTXT)   DATASET('TPNS.STLIN')     SHR
ALLOC DDNAME(NTWRK)    DATASET('TPNS.TESTFILE')  SHR
ALLOC DDNAME(SEQOUT)   DATASET('TPNS.SEQOUT')    SHR
ALLOC DDNAME(TAPEIN)   DATASET('ITPSGEN.TAPEIN') SHR
ALLOC DDNAME(CTLIN)    DATASET('ITPSGEN.CTLIN')  SHR
ALLOC DDNAME(SYSIN)    DATASET('ITPSGEN.SYSIN')  SHR
```

```
CALL  'TPNS.LOAD(ITPSGEN)' 'CTL'
FREE  DDNAME(SYSPRINT RATEDD INITDD MSGDD SYSUT2 SYSUT3)
FREE  DDNAME(MSGTXT STLTXT NTWRK SEQOUT TAPEIN CTLIN SYSIN)
```

Changes to the TPNS/ISPF Interface

When generating STL scripts using the TPNS/ISPF Interface, you must specify a data set to contain the generated STL source. Use the field labeled **Generated STL programs** on panel ITP0SGNP to specify this data set. A member will be created in this data set for each STL MSGTXT that is generated. The data set can have the same attributes as those used for the generated message decks data set. The following is an example of the ITP0SGNP panel as it might look when generating STL scripts.



```
ITP0SGNP          TPNS:  Generate Message Decks from Sorted Trace Data

Type information.  Then Press Enter
                                                                  More:   +
   Input Data Sets
      Sorted trace . . . . . . 'USERID.SORTED.TRACE'
        Tape:  Serial numbers           ,
               File number . .         (0-9999)
               Label type  . .         (NL or SL)
      Model script . . . . . . 'USERID.NETWORKS(MODEL)'
      Control commands . . . . 'USERID.CONTROL(SGEN)'

   Output Data Sets
      Generated message decks  'USERID.MSGFILE'
      Generated STL programs   'USERID.STLIN'
      Updated networks . . . . 'USERID.TESTFILE'
      Sequential output  . . . 'USERID.SEQOUT'
      Printer output . . . . . 'USERID.SYSPRINT'



Command ===>
F1=Help  F2=Split  F3=Exit  F4=Edit input  F5=Refresh  F6=Browse prt
F7=Bkwd  F8=Fwd    F9=Swap F10=Edit ctl   F11=Save    F12=Cancel
```

SEQOUT Data Set

If the SEQOUT control command is specified, ITPSGEN produces a SEQOUT data set as output. If you are generating STL scripts, the SEQOUT data set will contain network definition statements and STL code. The STL must be translated to the TPNS scripting language before you can run a simulation.

Note: SEQOUT data sets that contain STL source code are structured differently than SEQOUT data sets that contain TPNS scripting language source code. In SEQOUT data sets that contain STL source, the network definition is surrounded by the @NET and @ENDNET statements to inform the STL Translator to pass it to the TPNS preprocessor. As with non-STL script generation, the network definition will be updated to reflect the generated scripts if the NTWRK control command is specified.

Sample SEQOUT Data Set

The following is a sample SEQOUT data set.

```
@NET
IDCTSO     NTWRK     UTI=100,LOGDSPLY=BOTH
ITPIDC     PATH      ITPIDC
0          PATH      SGENTXT
```

```
* Model network for ITPECHO via IDC.
VAPPL1     VTAMAPPL  APPLID=ITPIDC,INIT=SEC,BUFSIZE=1920
ITPIDC     LU        PATH=(ITPIDC),
                     LUTYPE=LU2
*
@ENDNET
ITPIDC:  MSGTXT
  suspend(3)
  delay(4)
  erin
  cursor(1,27)
  type 'userid'
  cursor(2,7)
  transmit using ENTER
  delay(5)
  erin
  cursor(8,20)
  type 'ABCDEF'
  transmit using ENTER
  delay(21)
  erin
  cursor(5,6)
  transmit using ENTER
  delay(6)
  erin
  cursor(4,14)
  type '9.6;log'
  transmit using ENTER
  delay(3)
  erin
  cursor(4,21)
  type '/d a,1'
  transmit using ENTER
  delay(2)
  erin
  cursor(4,21)
  transmit using PF3
  delay(3)
  erin
  cursor(4,21)
  transmit using PF3
  delay(2)
  erin
  cursor(4,14)
  transmit using PF3
  delay(2)
  erin
  cursor(1,9)
  type 'logoff'
  cursor(2,7)
  transmit using ENTER
  do forever
    wait
  end
  ENDTXT
```

STL Translation

Before you can run a script that was generated in STL, you must translate the STL into TPNS scripting language. If you want to translate all generated scripts during the same STL invocation, point the STL input data set to the data set produced by the script generator SEQOUT DD. Use the sample in “Sample JCL for STL Translation” on page 26 when writing your own JCL to execute the TPNS STL Translator.

Sample JCL for STL Translation

You can use the following sample when writing your own JCL to execute the TPNS STL Translator.

```
//STLJOB   JOB
//*********************************************************************
//* Teleprocessing Network Simulator (TPNS)   5688-121                *
//*********************************************************************
//*                     STLJOB JCL                                    *
//* Sample JCL to execute the TPNS STL Translator (ITPSTL).           *
//*********************************************************************
//STL      EXEC PGM=ITPSTL,REGION=4096K
//STEPLIB  DD  DSN=TPNS.LOAD,DISP=SHR
//PARMDD   DD  DSN=TPNS.PARMDD,DISP=SHR
//RATEDD   DD  DSN=TPNS.RATETBLS,DISP=SHR
//INITDD   DD  DSN=TPNS.TESTFILE,DISP=SHR
//SYSPRINT DD  SYSOUT=A
//MSGDD    DD  DSN=TPNS.MSGFILE,DISP=SHR
//SEQOUT   DD  DSN=TPNS.STL.SEQOUT,DISP=SHR
//SYSLIB   DD  DSN=TPNS.STLIN,DISP=SHR
//SYSUT1   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT2   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSUT3   DD  UNIT=SYSDA,SPACE=(TRK,(10,10,3))
//SYSIN    DD  DSN=TPNS.SEQOUT,DISP=SHR
```

Chapter 3. Additional TPNS TCP/IP Support

TPNS has been updated to enable you to simulate the following types of TCP/IP clients:

- Telnet 3270E clients connecting to a Telnet 3270 or 3270E server
- Telnet Line Mode Network Virtual Terminal clients connecting to a Telnet server
- Telnet 5250 clients connecting to a Telnet 5250 server
- Simple User Datagram Protocol (UDP) clients

Other TCP/IP-related enhancements included are:

- The LOCLPORT operand has been added to allow the specification of the local port number to be used by a Simple TCP (STCP) or Simple UDP (SUDP) device.
- The STCPROLE operand has been added to allow you to specify whether STCP devices are to act as a client or server.
- ITPGSIPA, an exit routine, has been added to glean the address information from received data and set the address information for data to be transmitted.
- TPNS now uses the Macro API of the IBM TCP/IP product when it is available.
- The port number to be used for subsequent connections can now be altered for TCP/IP resources via PORT=nnnnn on the *a* (alter) TPNS operator command.

Telnet 3270E Support

TPNS now simulates Telnet 3270E clients connecting to a Telnet 3270 server or 3270E server. This enhancement enables you to simulate Telnet 3270E terminals or Telnet LU3 printers. The following functions are supported:

Device naming

Most host applications behave differently depending on the network name of the terminal. The TPNS 3270E support allows you to simulate a 3270 client requesting that a connection be associated with a given 3270 device name. This capability is particularly useful when simulating a Telnet 3270E printer since many host applications pre-define printer destinations. To query the *Connect* or *Associate* name of the TCP/IP terminal or printer, query the TPNS TN3270E terminal or printer name using the TPNS operator command **Q** and look for the LUNAME= operand in the returned query response lines.

SYSREQ key

TPNS supports the SYSREQ key. The Telnet Abort Output (AO) command is sent when SYSREQ is coded in a TPNS message generation deck or STL procedure.

SNA positive/negative response process

The SNA positive/negative response process is supported. A positive response may be sent (depending on function negotiations) to indicate that previously received data has been successfully processed. A negative response may be sent to indicate that an error occurred while processing the previously received data.

Telnet 3270E suboption negotiations

TPNS provides a means to simulate Telnet 3270E suboption negotiations. Suboption negotiations are handled in two steps. The first step is a device-type negotiation that is similar to, but more complicated than the Telnet terminal-type option. The second step involves the negotiation of a set of supported 3270 functions. If these two suboption negotiations are successful, the 3270 data stream transmission may begin. Each data message in TN3270E is prefixed by a header containing flags and indicators that convey positive and negative responses and the type of data that follows — that is, 3270 data stream, SNA character stream, or device status information. Once the DEVICE-TYPE information has been successfully negotiated, the client and server will exchange the FUNCTIONS information.

Suboption Negotiation Commands

The following are descriptions of the suboption negotiation commands:

DEVICE-TYPE

The server transmits the DEVICE-TYPE SEND command and the TPNS TN3270E client responds with the DEVICE-TYPE REQUEST command, which includes the device-type and may include a device-name request.

IBM-3287-1 printers are supported. Terminal types supported include:

IBM-3278-2	IBM-3278-2-E	(24 rows x 80 cols display)
IBM-3278-3	IBM-3278-3-E	(32 rows x 80 cols display)
IBM-3278-4	IBM-3278-4-E	(43 rows x 80 cols display)
IBM-3278-5	IBM-3278-5-E	(27 rows x 132 cols display)
IBM-3278-2	IBM-3278-2-E	(default if none of above)

The ″E″ indicates that extended data stream capability is supported. At a minimum, this usually includes support for extended colors and highlighting, but may include other functions.

CONNECT The TPNS TN3270E client requests that the server assign a specific device-name to this Telnet session. This is implemented in TPNS by the use of the RESOURCE= operand to specify the name and ASSOC=NO operand value. The device-name must not conflict with the device-type. For example, the client may request DEVICE-TYPE IBM-3287-1 which is a printer and specify CONNECT T1000001, but T1000001 is defined as a terminal. This would result in a denial of the request. If the requested device-name is already associated with some other Telnet session, or if it is not defined to the server, the request will be denied.

ASSOCIATE The TPNS TN3270E client requests a DEVICE-TYPE that represents a printer. This requests that the session be assigned the device-name of the printer that is paired with the terminal named in the request. This is implemented by the use of the RESOURCE= operand for the terminal name and ASSOC=YES operand value. If the device-type does not represent a printer, or if the device-name is not that of a terminal, the request will be denied.

FUNCTIONS The FUNCTIONS REQUEST command contains a list of the 3270 functions that the sender would like to see supported on this session. The list of 3270 functions that may be negotiated in the function list are specified by the FUNCTS= operand. All functions not in the FUNCTIONS REQUEST command list are considered not supported. See the FUNCTS= operand description for the description of the 3270 functions.

Defining a Telnet 3270E Client

This section illustrates how to define a Telnet 3270E client. A sample TPNS network definition and TPNS script for a Telnet 3270E client are provided. For additional information about defining display characteristics and 3270 characteristics for Telnet 3270E clients, refer to *Defining TPNS Networks*, Chapter 7, ″Simulating TCP/IP Devices″ and to the *TPNS Language Reference* manual.

Sample Network Definition for a Telnet 3270E Simulation

To simulate a Telnet 3270E client, code the following operands in your network definition. These operands can be coded on the DEV statement in a TCPIP group definition, or on the TCPIP statement in order to establish operand values for any DEV statements in that group that do not specify certain operands.

```
ASSOC={YES|NO}

     Function: Specifies whether the ASSOCIATE command is to be used
               when requesting a DEVICE-TYPE that represents a printer.
               Only valid when TYPE=TN3270P.

     Format:   YES or NO.

     Default:  NO



FUNCTS={integer,...}

     Function: Specifies the list of 3270 options supported for the
               specific FUNCTIONS REQUEST command that the sender would
               like to see supported on this session.

     Format:   Integers between 0 and 4 or a single integer value of 5.

               0 - Bind image, allows the server to send the SNA Bind
                   image and Unbind notification to the client.

               1 - Data stream control, for TYPE=TN3270P only.  Allows
                   the use of standard 3270 data stream.  This
                   corresponds to LU type 3 SNA sessions.

               2 - Responses, provides support for positive and negative
                   response handling.  Allows the server to reflect to
                   the client any and all definite, exception, and no
                   response requests sent by the host application.

               3 - SNA character stream control codes for printer
                   sessions only.  Allows the use of the SNA Character
                   Stream (SCS) and SCS control codes on the session.
                   SCS is used with LU type 1 SNA sessions.

                   Note:  TPNS can receive LU type 1 data but nothing is
                          done except logging and logic testing.  There
                          is no printer speed delay generated.  No
                          module is called to check the data or set a
                          delay.

               4 - Sysreq, allows the client and server to emulate some
```

```
                    (or all, depending on the server) of the functions of
                    the SYSREQ key in an SNA environment.

                5 - Null list, no options are supported.

      Default:  0,1,2,3,4



 PRTSPD=integer

      Function: Specifies the speed at which the device being simulated
                will print the data received.  The device buffer is
                assumed to be busy for the time taken to print the data.
                A TN3270E response will be sent after the data has
                been printed.

      Format:   An integer from 0 to 32767 that specifies the print speed
                for the device in characters per second.  PRTSPD=0 means
                "immediate completion of print operation."

                Note:  This option is only valid for TYPE=TN3270P and LU3
                       printer types.  This option does not take into
                       account the extra time required for color printers.

      Default:  0



 RESOURCE=name

      Function: Specifies the TN3270E LU name to connect or associate
                with.

      Format:   A 1- to 8- character name.

      Default:  None.



 TYPE={TN3270E|TN3270P}

      Function: TN3270E specifies a Telnet 3270E terminal.
                TN3270P specifies a Telnet 3270E printer.

      Format:   TN3270E or TN3270P
```

Sample TPNS Script for a Telnet 3270E Simulation

Below is an example of a TPNS script simulating two Telnet 3270E printers receiving data and two Telnet 3270E terminals connecting to an application logon screen.

```
************************************************************************
* Network Configuration:  Telnet 3270E simulation                      *
*                                                                      *
* Description:  This TPNS script will simulate two Telnet 3270E        *
*               terminals connecting to an application logon           *
*               screen and logging back off.  This script also         *
*               simulates two Telnet 3270E printers receiving          *
*               data. The SERVADDR operand specifies the IP            *
*               dotted address of the host to which the                *
*               terminals and printers will connect.                   *
*               Some values may need to be changed in this data set    *
*               in order to operate in your environment.  They are     *
*               indicated by the "<== " string.                        *
*                                                                      *
* TPNS CP Gen:  None                                                 *
* NCP Gen:      None                                                 *
*                                                                    *
* Publications Cross Reference:                                      *
*  1) Defining TPNS Networks       - Information on this particular  *
*                                    network configuration           *
*  2) TPNS Message Generation Decks - Information on writing message *
*                                    text decks                      *
*  3) TPNS Language Reference      - Details on how to code TPNS     *
*                                    statements                      *
**********************************************************************

*--------------------------------------------------------------------*
* Network statement operands.                                        *
*--------------------------------------------------------------------*
TN3270E  NTWRK HEAD='TEST NETWORK', * Set the title line
               CONRATE=YES,         * Print message rates on console
               ITIME=1,             * Interval report every 1 minute
               MSGTRACE=YES,        * Log message generation trace
               LOGDSPLY=BOTH,       * Log formatted 3270 displays
               BUFSIZE=2048,        * Specify buffer size
               THKTIME=UNLOCK,      * Wait for keyboard unlock
               UTI=100,             * User time interval is 1 second
               SEQ=0,               * Clear network sequence counter
               TCPNAME=TCPIP,       * <== Default name of the local
                                    *     TCPIP virtual machine
               SERVADDR=9.67.6.1    * <== Default IP server address
                                    *     to which you will connect
*--------------------------------------------------------------------*
* Define the message decks included in this path                     *
*--------------------------------------------------------------------*
HOST1    PATH  HOST1                * Execute HOST1 msgtxt
HOST2    PATH  HOST2                * Execute HOST2 msgtxt
*--------------------------------------------------------------------*
* Define the network resources.                                      *
*                                                                    *
* This is a Telnet connection with 2 simulated Telnet 3270E terminals  *
* and 2 simulated Telnet LU3 printers.  You may add additional       *
* operands on the devices if desired.  See the TPNS Language Reference *
* manual for details on valid operands.                              *
*--------------------------------------------------------------------*
TCONN1   TCPIP   TNPORT=23
DEV11    DEV     TYPE=TN3270E,RESOURCE=TPNS01,PATH=(HOST1)
DEV12    DEV     TYPE=TN3270E,FUNCTS=(0,1,2),PATH=(HOST1)
DEV13    DEV     TYPE=TN3270P,RESOURCE=TPNS02,ASSOC=YES,PATH=(HOST2)
DEV14    DEV     TYPE=TN3270P,PRTSPD=1000,PATH=(HOST2)


                     MESSAGE GENERATION DECK

HOST1    MSGTXT
*--------------------------------------------------------------------*
* The Message Generation deck for the Telnet 3270E terminal.         *
*                                                                    *
* This deck calls WAITSCRN to wait for the application logon screen  *
* and issues a Write To Operator message acknowledging that the device *
* has successfully connected.  A USERID and password are selected    *
* from user tables defined below that attempt to logon to the host.  *
* The device then calls WAITREDY to wait for a "ready prompt" from   *
* the host indicating a successful logon.  After receiving the       *
* appropriate ready message, the device logs off.  After a device    *
* logs off, TPNS is closed down.                                     *
*--------------------------------------------------------------------*
         CALL   NAME=WAITSCRN
         WTO    ($DEVID$ ESTABLISHED TELNET SESSION, LOGGING ON)
         SET    DC1=NSEQ
         SET    NSEQ=+1
         TEXT   ($UTBL,IDS,DC1$),MORE=YES
         TAB
         TEXT   ($UTBL,PWS,DC1$)
         ENTER
         CALL   NAME=WAITREDY
         WTO    (GOT READY PROMPT)
         TEXT   (LOGOFF)
         ENTER
         CALL   NAME=WAITLOGF
         WTO    (GOT LOGOFF MESSAGE)
         OPCMND (ZEND)
         ENDTXT
*---------------------------------------------------------------------*
WAITSCRN MSGTXT
*---------------------------------------------------------------------*
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate logon screen.    *
*---------------------------------------------------------------------*
0        IF     WHEN=IN,LOC=B+0,TEXT=(VM/ESA ONLINE),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*----------------------------------------------------------------------
WAITREDY MSGTXT
*---------------------------------------------------------------------*
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate ready message    *
*---------------------------------------------------------------------*
0        IF     LOC=B+0,TEXT=(Ready),SCAN=YES,THEN=CONT
1        IF     LOC=B+0,TEXT=(Reconnected),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*----------------------------------------------------------------------
WAITLOGF MSGTXT
*----------------------------------------------------------------------
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate logoff message   *
*----------------------------------------------------------------------
0        IF     LOC=B+0,TEXT=(Logoff),SCAN=YES,THEN=CONT
1        IF     LOC=B+0,TEXT=(LOGOFF),SCAN=YES,THEN=CONT
         WAIT
         CLEAR
         ENDTXT
*
HOST2    MSGTXT
*---------------------------------------------------------------------*
* The Message Generation deck for the Telnet 3270E LU3 printer.       *
*                                                                     *
* This deck waits for the NL and EM in the data stream and checks for *
* an unbind sent by the server.                                       *
*                                                                     *
*---------------------------------------------------------------------*
         WTO      (STARTING PRINTER SESSION,$MSGTXTID$)
0        IF       LOC=B+0,SCAN=YES,TEXT=('1519'),
                  THEN=E-MSGAA,WHEN=IN
1        IF       LOC=D+0,TEXT=('00'),
                  COND=GE,THEN=E-INCAL,
                  DATASAVE=(1,D+0,32000)
2        IF       LOC=D+0,SCAN=YES,TEXT=('0800'),
                  THEN=E-MSGBB,WHEN=IN
         WAIT
MSGAA    WTO      (PRINTOUT RECEIVED,$MSGTXTID$)
         RETURN
MSGBB    WTO      (UNBIND RECEIVED,$MSGTXTID$)
         WTO      ($CNTR,DC1$ BYTES RECEIVED)
         SET      DC1=0
         RETURN
INCAL    SET      DC2=LENG(1),DC1=+DC2
         WTO      ($CNTR,DC2$ BYTES RECEIVED)
         WTO      ($CNTR,DC1$ TOTAL BYTES RECEIVED)
         RETURN
         ENDTXT
*-----------------------------------------------------------------------
*                                    * <== The USERIDs and passwords
*                                    *     below must be changed to
*                                    *     valid names
*-----------------------------------------------------------------------
IDS      MSGUTBL  (USER1),(USER2),(USER3),(USER4)
PWS      MSGUTBL  (PASSWORD),(PASSWORD),(PASSWORD),(PASSWORD)
```

STL PROCEDURE

```
@PROGRAM=TN3270E
/*---------------------------------------------------------------------*
* This deck waits for the application logon screen and displays a      *
* message to the operator acknowledging that the device has been       *
* successfully connected.  A USERID and password are selected from     *
* user tables defined below that attempt to logon to the host.  The    *
* device then calls WAITREDY to wait for a "ready prompt" from the     *
* host indicating a successful logon.  After receiving the appropriate *
* ready message, the device logs off.  Once a device logs off, TPNS    *
* is closed down.                                                      *
*---------------------------------------------------------------------*/
allocate nextnum 'NSEQ'
integer  nextid
integer  totaldata

host1: msgtxt
 wait until onin index(screen, 'VM/ESA ONLINE') > 0
 say devid() 'ESTABLISHED TELNET SESSION, LOGGING ON'
 nextid  = nextnum
 nextnum = nextnum + 1
 type utbl(ids,nextid)
 tab
 type utbl(pws,nextid)
 transmit and wait until onin index(screen, 'READY;') > 0
 say 'GOT READY PROMPT'
 type 'LOGOFF'
 transmit and wait until onin index(screen, 'LOGOFF') > 0
 opcmnd 'ZEND'
endtxt

/*---------------------------------------------------------------------*
* The Message deck for the Telnet 3270E LU3 printer.                   *
*                                                                      *
* This deck waits for the NL and EM in the data stream and checks for  *
* an unbind sent by the server.                                        *
*---------------------------------------------------------------------*/
host2: msgtxt
 say 'STARTING PRINTER SESSION, ' msgtxtid()
 onin index(buffer,'1519') > 0 then
      say 'PRINTOUT RECEIVED ' msgtxtid()
 onin then do
      totaldata=totaldata+length(data)
      say char(length(data)) ' BYTES RECEIVED'
      say char(totaldata) ' TOTAL BYTES RECEIVED'
   end
 onin index(buffer,'0800') > 0 then do
      say 'UNBIND RECEIVED ' msgtxtid()
      say char(totaldata) ' BYTES RECEIVED'
      total=0
   end
   wait
endtxt

ids: msgutbl
 'USER1'
 'USER2'
 'USER3'
 'USER4'
endutbl

pws: msgutbl
 'PASSWORD'
 'PASSWORD'
 'PASSWORD'
 'PASSWORD'
endutbl
```

Telnet Line Mode Network Virtual Terminal

TPNS now simulates Telnet Line Mode Network Virtual Terminal clients connecting to a Telnet server. The client will look like a Network Virtual Terminal. The TPNS user must append carriage control and line feed characters at the end of each data stream to be sent. However, the incoming data is translated from ASCII to EBCDIC and the outgoing data is translated from EBCDIC to ASCII. Line mode does not provide screen images. No buffers are maintained in the TPNS line mode support. For logic testing, only the incoming data streams are checked. Also refer to "Appendix C. Simple TCP Sample Script" on page 109 for an example of a Telnet Line Mode NVT client exchanging negotiations with a Telnet server using the TPNS Simple TCP support.

Defining a Telnet Line Mode Network Virtual Terminal Client

To simulate a Telnet Line Mode Network Virtual Terminal client, code the following operand in your network definition. The operand can be coded on the DEV statement in a TCPIP group definition, or on the TCPIP statement to establish the terminal type for any of the DEV statements in the group that do not have a specific type.

```
 TYPE=TNNVT

      Function: Specifies a Telnet Line Mode Network Virtual Terminal client.

      Format:   TNNVT
```

Sample Telnet Line Mode Network Virtual Terminal Message Generation Deck

The following is a sample Telnet Line Mode Network Virtual Terminal message generation deck:

Telnet Line Mode Network Virtual Terminal Simulation Message Generation Deck

```
*---------------------------------------------------------------------*
* The Message Generation deck for the Telnet Line Mode NVT.           *
*                                                                     *
* This deck calls WAITSCRN to wait for the application logon screen   *
* and issues a Write To Operator message acknowledging that the device *
* has successfully connected.  A USERID is selected from the id user  *
* table defined below to attempt to logon.  The device then calls     *
* WAITPWD to wait for "Password" and then send the password from the  *
* password user table below.  After receiving the "$" prompt, the     *
* device logs off.                                                    *
*                                                                     *
*---------------------------------------------------------------------*
HOST1    MSGTXT
         CALL   NAME=WAITSCRN
         WTO    ($DEVID$ ESTABLISHED TELNET SESSION, LOGGING ON)
         SET    DC1=NSEQ
         SET    NSEQ=+1
         TEXT   ($UTBL,IDS,DC1$'0D25')
         ENTER
         CALL   NAME=WAITPWD
         WTO    (GOT PASSWORD)
         TEXT   ($UTBL,PWS,DC1$'0D25')
         ENTER
         CALL   NAME=WAIT$
         TEXT   (LOGOUT)
         WTO    (GOT $$ PROMPT)
         ENDTXT
*---------------------------------------------------------------------*
WAITSCRN MSGTXT
*---------------------------------------------------------------------*
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate logon screen.    *
*---------------------------------------------------------------------*
0        IF     WHEN=IN,LOC=B+0,TEXT=(login:),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*----------------------------------------------------------------------
WAITPWD  MSGTXT
*---------------------------------------------------------------------*
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate password message *
*---------------------------------------------------------------------*
0        IF     LOC=B+0,TEXT=(Password),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*----------------------------------------------------------------------
WAIT$    MSGTXT
*----------------------------------------------------------------------
*                                    * <== The TEXT operand below must  *
*                                    *     be changed to reflect the    *
*                                    *     appropriate prompt message   *
*----------------------------------------------------------------------
0        IF     LOC=B+0,TEXT=($$),SCAN=YES,THEN=CONT
         WAIT
         CLEAR
         ENDTXT
*----------------------------------------------------------------------
*                                    * <== The USERIDs and passwords
*                                    *     below must be changed to
*                                    *     valid names
*----------------------------------------------------------------------
IDS      MSGUTBL  (USER1),(USER2),(USER3)
PWS      MSGUTBL  (PASSWD1),(PASSWD2),(PASSWD3)
```

Sample Telnet Line Mode Network Virtual Terminal STL Procedure

The following is a sample Telnet Line Mode Network Virtual Terminal STL procedure:

```
allocate nextnum 'NSEQ'
integer  nextid
constant crlf '0D25'x
host1: msgtxt
 data_received=''
 onout then data_received=''
 onin then data_received=data_received||buffer
 wait until onin index(data_received,'login') > 0
 say devid() 'ESTABLISHED TELNET SESSION, LOGGING ON'
 nextid =nextnum
 nextnum=nextnum+1
 if nextnum=utblmax(ids) then
  nextnum=0
 say devid() 'sending LOGIN'
 type utbl(ids,nextid)||crlf
 transmit and wait until onin index(data_received,'Password') > 0
 say devid() 'got Password PROMPT'
 type utbl(pws,nextid)||crlf
 do nextcmd=0 to utblmax(cmds)
  transmit and wait until onin index(data_received,'$') > 0
  say devid() 'sending COMMAND' utbl(cmds,nextcmd)
  type utbl(cmds,nextcmd)||crlf
 end
endtxt

ids: msgutbl
 'user1'
 'user2'
 'user3'
endutbl

pws: msgutbl
 'passwd1'
 'passwd2'
 'passwd3'
endutbl

cmds: msgutbl
 'cd /usr/lpp'
 'ls'
 'logout'
endutbl
```

Telnet 5250 Support

TPNS now simulates Telnet 5250 clients connecting to a Telnet 5250 server. To simulate a Telnet 5250 client, code the following operand in your network definition:

```
 TYPE=TN5250

      Function: Specifies a Telnet 5250 terminal.

      Format:   TN5250
```

The operand can be coded on the DEV statement in a TCPIP group definition, or on the TCPIP statement to establish the terminal type for any of the DEV statements in that group which do not have a specific type. For details on 5250 simulation statements, refer to *TPNS Language Reference*.

Simple UDP Terminal Support

TPNS now supports simulation of Simple UDP terminals. The terminal type SUDP has been added to provide simple client UDP support in TPNS. To simulate a Simple UDP terminal, code the following operand in your network definition:

```
TYPE=SUDP
```

SUDP terminal simulation is the same as simple TCP terminal simulation other than supporting UDP instead of TCP protocols. The SUDPPORT=port_number operand has also been added to define a default simple UDP port number other than the default of 256.

Limited Server Enhancements

For STCP devices, you can now specify whether the device is to act as a client or server. If specified as a server, the device listens for a connection before any messages can be generated. In most cases, the script is also set up to wait for a received data message before generating a response, however, the only requirement is to wait for the connection. In order for STCP devices to perform the server role, a local port number must be specified. To specify whether a STCP device is to act as a server or client, code the following operand in your network definition:

```
STCPROLE=CLIENT|SERVER
```

where the value specified indicates the role to be performed. **CLIENT** is the default.

Note: This operand can only be specified on a DEV statement associated with a TCPIP statement.

The exit routine which can be used to glean the address information from received data and set the address information for data to be transmitted is ITPGSIPA. This exit can be used as an input user exit to retrieve the full address of the source of the last data received for Simple UDP or Simple TCP simulated devices or as a message generation exit to set the full address to be used for the next message to be transmitted for SUDP devices or for the next connection established for STCP devices. When called as an input exit (specified by INEXIT on the NTWRK statement), ITPGSIPA saves the full INET address for the message being received by Simple TCP or Simple UDP terminals in network save area 13. The address is in the form used by the sockets interface, which is as follows:

Table 1. INET Address Format

Offset	Length	Description
0	2	Address Family
2	2	Port (AF_NET=0002)
4	4	IP Address
8	8	Binary Zeros

When called as a Message Generation exit (USEREXIT STL statement or EXIT statement in TPNS Scripting Language), ITPGSIPA moves the INET address from network save area 13 into the internal TPNS control block so that the new address will be used for the next message transmitted (SUDP) or the next connection (STCP). ITPGSIPA assumes that the INET address exists in the save area in the format described in Table 1 on page 37.

Local Port Number Support

TPNS allows the specification of the local port number to be used by a Simple TCP or Simple UDP device. When a local port number is specified for STCP or SUDP devices, TPNS will obtain the socket and BIND that socket to the specified local port before any data is transmitted or received on that socket. To specify the local port number, code the following operand in your network definition:

```
LOCLPORT=n
```

where *n* is an integer from 1 to 65535 representing the local port number to be used.

Note: This operand can only be specified on a DEV statement associated with a TCPIP statement.

TCP/IP Macro API Utilization

When available for the TCP/IP instance, as specified by the TCPIP operand, TPNS will use the TCP/IP Macro API instead of the IUCV API. This will improve performance and provide compatibility with future releases of the IBM TCP/IP product.

Alter PORT=port_number

The port number can now be altered for TCP/IP resources via PORT=nnnnn on the alter command. As part of this change, the PORT=nnnnn and NEXTPORT=nnnnn values are displayed when a TCP/IP resource is queried, and the port number has been added to informational message ITP478I. The port number is changed just before the next connection attempt. The NEXTPORT=nnnnn value is only shown when such a change is pending. An example of the query output follows:

```
A STCP001,PORT=2345
ITP033I ALTER COMPLETE
Q STCP001

ITP143I  NAME=STCP001 -ACTIVE  TYPE=STCP  QUIESCED=NO
ITP143I  MSG DELAY=F(1)                  BLK DELAY=NONE
ITP143I  WAIT=ON
ITP143I  SERVADDR=9.9.9.9  TCPSTATE=CLOSED
ITP143I  PORT=1234  NEXTPORT=2345 LOCLPORT=9999 STCPROLE=CLIENT
ITP143I  WAIT EVENTS = *NONE*
ITP143I  ON   EVENTS = *NONE*
ITP143I  INSERT PATH=NONE  PATHS=STCPPATH
ITP143I  CURRENT PATH=STCPPATH PATH ENTRY=1
ITP143I  CURRENT DECK=TCPDECK       CURRENT STATEMENT:  TPNS=00002
ITP143I  MSGTRACE=NO   STLTRACE=YES  INTERMESSAGE DELAY=NOT ACTIVE
ITP143I  ACTIVE UTI IS  NTWRKUTI=0
ITP143I  DSEQ            1             NUMBER INDEX COUNTERS=3
ITP143I  DC01-03         0          0          0
ITP143I  DEV    SWITCHES       01-04=0000  05-08=0000  09-12=0000  13-16=0000
ITP143I  NUMBER SWITCHES=32    17-20=0000  21-24=0000  25-28=0000  29-32=0000
```

```
ITP143I  NO MESSAGE TRANSMITTED
ITP143I  NO MESSAGE RECEIVED
ITP143I  DEV LOCATION=014310F8
```

Simulated Resource Type Codes

New TPNS code values have been added for the new simulated resource types. The following codes identify resource types when reading a log data set listing or processing in a user exit routine:

Device	Type
TN3270E	95
TN3270P	96
SUDP	97
TNNVT	98
TN5250	99

User Exit Control Blocks

The following new constant values have been added to the DEV (device control block) and the LOG (log record header format) sections:

Device Control Block

Len	Type	Value	Name	Description
DEVICE TYPES				
1	HEX	95	DEVTNE	TELNET 3270E DEVICE
1	HEX	96	DEVTNEP	TELNET 3270E PRINTER
1	HEX	97	DEVSUDP	SIMPLE UDP
1	HEX	98	DEVLNMD	TELNET LINE MODE NVT DEVICE
1	HEX	99	DEV5250	TELNET 5250 DEVICE

Log Record Header Format

Len	Type	Value	Name	Description
LOG TERMINAL TYPES				
1	HEX	95	LOG32TE	TELNET 3270E DEVICE
1	HEX	96	LOG32TP	TELNET 3270E PRINTER
1	HEX	97	LOGSUDP	SIMPLE UDP
1	HEX	98	LOGLNMD	TELNET LINE MODE NVT DEVICE
1	HEX	99	LOG5250	TELNET 5250 DEVICE

Chapter 4. TPNS Scripting Enhancements

This chapter describes the many enhancements related to the Structured Translator Language (STL) and to the TPNS scripting language as well as other miscellaneous enhancements.

STL Data Manipulation Functions

STL, the high-level structured programming language used to create TPNS message generation decks, has been expanded to include new scripting statements and functions. With this support, STL becomes more REXX-like giving the TPNS scripter more capabilities for performing advanced data manipulation. Each new statement and function listed below is detailed in the pages that follow including a section which lists the equivalent TPNS scripting language for that specific STL statement or function.

- A BITAND function that performs a logical AND'ing of two strings.
- A BITOR function that performs a logical OR'ing of two strings.
- A BITXOR function that performs a logical XOR'ing of two strings.
- A B2X function which converts binary strings of data to their hexadecimal equivalent.
- A CENTER function that centers a string of data within an area of a particular length.
- A COPIES function that produces a string of data which represents some number of concatenated copies of the original string.
- A DELWORD function that deletes some number of consecutive words within a string of words.
- A D2C function which converts a decimal number into its equivalent hexadecimal string value.
- A LASTPOS function which locates the last occurrence of a specified string within another string.
- An OVERLAY function that replaces or overlays existing text strings with new text.
- A POS function which locates the first occurrence of a specified string within another string.
- A REVERSE function which creates a string of characters in reverse order of another string.
- A SPACE function which creates a string of uniformly delimited words from a string of words containing various amounts of spacing.
- A STRIP function which removes leading and/or trailing characters from a string.
- A SUBWORD function which creates a string of words which represents a consecutive set of words from another string.
- A WORD function which creates a single word string by specifying a particular word position in another string.
- A WORDINDEX function which identifies where in a string a particular word is located on a character boundary.
- A WORDPOS function which identifies where in a string a particular word is located on a word boundary.
- A WORDS function which determines how many words there are in a particular string.

- An X2B function which converts a string of hexadecimal characters to its binary character equivalent.
- An X2C function which converts a string of hexadecimal characters to its character equivalent.

BITAND

```
BITAND(string1[,[string2][,pad]])
```

Where
string1 is a string expression.

string2 is a string expression. This is optional.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns
String

Function
The BITAND function returns a string composed of the *string1* and *string2* input strings logically AND'ed together, bit by bit. The length of the result is the length of the longer of the two strings. The shorter of the two strings is extended with the pad character on the right before carrying out the logical operation. The default for *string2* is the null string and the default for *pad* is X'FF'.

Examples
```
a = BITAND('55AA'x,'FF88'x)          /* Assigns '5588'x to "a"      */
b = BITAND('COLORADO','FF'x,'BF'x)   /* Assigns 'Colorado' to "b"   */
```

Note: The BITAND function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support BITAND
The DATASAVE statement has been updated to allow BITAND as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=BITAND,
         TEXT=(string1),
         TEXT2=(string2),
         PAD=pad,
         AREA=area
```

BITOR

```
BITOR(string1[,[string2][,pad]])
```

Where

string1 is a string expression.

string2 is a string expression. This is optional.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns

String

Function

The BITOR function returns a string composed of the *string1* and *string2* input strings logically inclusive-OR'ed together, bit by bit. The length of the result is the length of the longer of the two strings. The shorter of the two strings is extended with the pad character on the right before carrying out the logical operation. The default for *string2* is the null string and the default for *pad* is X'00'.

Examples

```
a = BITOR('152535'x,'22'x)          /* Assigns '372535'x to "a"       */
b = BITOR('Barney',,'40'x)          /* Assigns 'BARNEY' to "b"        */
c = BITOR('112233'x,'66'x,'88'x)    /* Assigns '77AABB'x to "c"       */
```

Note: The BITOR function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support BITOR

The DATASAVE statement has been updated to allow BITOR as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=BITOR,
         TEXT=(string1),
         TEXT2=(string2),
         PAD=pad,
         AREA=area
```

BITXOR

```
BITXOR(string1[,[string2][,pad]])
```

Where

string1 is a string expression.

string2 is a string expression. This is optional.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns

String

Function

The BITXOR function returns a string composed of the *string1* and *string2* input strings logically eXclusive-ORed together, bit by bit. The length of the result is the length of the longer of the two strings. The shorter of the two strings is extended with the pad character on the right before carrying out the logical operation. The default for *string2* is the null string and the default for *pad* is X'00'.

Examples

```
a = BITXOR('1211'x,'22'x)            /* Assigns '3011'x to "a"     */
b = BITXOR('1111'x,'444444'x,'40'x)  /* Assigns '555504'x to "b"   */
c = BITXOR('AAAA'x,,'FF'x)           /* Assigns '5555'x to "c"     */
```

Note: The BITXOR function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support BITXOR

The DATASAVE statement has been updated to allow BITXOR as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to "Operands for Datasave Statement" on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=BITXOR,
         TEXT=(string1),
         TEXT2=(string2),
         PAD=pad,
         AREA=area
```

B2X

```
B2X(binary_string)
```

Where

binary_string is a string expression containing only ’0’s or ’1’s.

Returns

String

Function

The B2X function returns a string in character format, that represents *binary_string* converted to hexadecimal. It can be any length. You can optionally include blanks in *binary_string* (at four-digit boundaries only, not leading or trailing) to aid readability; they are ignored. The returned string uses uppercase alphabetics for the values A-F, and does not include blanks.

If *binary_string* is the null string or a string formatted other than as described above, B2X returns a null string. If the number of binary digits in *binary_string* is not a multiple of four, then up to three 0 digits are added on the left before the conversion to make a total that is a multiple of four.

Examples

```
a = B2X('10101010')                    /* Assigns 'AA' to "a"      */
b = B2X('1   0011     1100')           /* Assigns '13C' to "b"     */
c = B2X('10001110    0111')            /* Assigns '8E7' to "c"     */
```

Note: The B2X function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support B2X

The DATASAVE statement has been updated to allow B2X as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=B2X,
         TEXT=(binary_string),
         AREA=area
```

CENTER

```
CENTER(string,length[,pad])
```

Where

string is a string expression.

length is an integer expression with a value from 1 to 32767.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns

String

Function

The CENTER function returns a string of length *length* with *string* centered in it, with *pad* characters added as necessary on both ends. If *string* is longer than *length*, it is truncated at both ends to fit. If an odd number of characters are truncated or added, the right-hand end loses or gains one more character than the left-hand end. The default value for *pad* is a blank (X’40’).

Examples

```
a = CENTER('abc',5)              /* Assigns ' abc ' to "a"      */
b = CENTER('abcdef',10,'C1'x)    /* Assigns 'AAabcdefAA' to "b" */
c = CENTER('abcdefg',4)          /* Assigns 'bcde' to "c"       */
```

Note: The CENTER function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support CENTER

The DATASAVE statement has been updated to allow CENTER as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=CENTER,
         TEXT=(string),
         PLENG=length,
         PAD=pad,
         AREA=area
```

COPIES

```
COPIES(string,n)
```

Where

string is a string expression.

n is an integer expression with a value from 1 to 32767.

Returns

String

Function

The COPIES function returns *n* concatenated copies of *string*.

Examples

```
a = COPIES('abc',3)              /* Assigns 'abcabcabc' to "a"       */
b = COPIES(' ab',3)              /* Assigns ' ab ab ab' to "b"       */
```

Note: The COPIES function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support COPIES

The DATASAVE statement has been updated to allow COPIES as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to "Operands for Datasave Statement" on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=COPIES,
         TEXT=(string),
         COUNT=n,
         AREA=area
```

DELWORD

```
DELWORD(string,n[,length])
```

Where

string is a string expression.

n is an integer expression with a value from 1 to 32767.

length is an integer expression with a value from 1 to 32767. This is optional.

Returns

String

Function

The DELWORD function returns *string* after deleting the substring that starts at the *n*th word and is of *length* blank-delimited words. If you omit *length*, or if *length* is greater than the number of words from *n* to the end of *string*, the function deletes the remaining words in *string* (including the *n*th word). If *n* is greater than the number of words in *string*, the function returns *string* unchanged. The string deleted includes any blanks following the final word involved but none of the blanks preceding the first word involved.

Examples

```
a = DELWORD('Now is the time',2,2)    /* Assigns 'Now time' to "a"   */
b = DELWORD('Now is the time',3)      /* Assigns 'Now is' to "b"     */
c = DELWORD('Now is the time',1,2)    /* Assigns 'the time' to "c"   */
```

Note: The DELWORD function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support DELWORD

The DATASAVE statement has been updated to allow DELWORD as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=DELWORD,
         TEXT=(string),
         POS=n,
         PLENG=length,
         AREA=area
```

D2C

```
D2C(number[,n])
```

Where

number is an integer expression.

n is an integer constant expression with a value of 1 to 4. This is optional.

Returns

String

Function

The D2C function converts a decimal integer value into its equivalent hexadecimal string value. *n* specifies the number of characters to be returned. If *n* is specified as 1 and the hexadecimal string has a length greater than 1, then the rightmost byte is returned. If *n* is not specified, all significant bytes of the number will be returned; a leading zero will not be returned.

Examples

```
a =  D2C(15)              /* Assigns '0F'x to "a"            */
b =  D2C(32767)           /* Assigns '7FFF'x to "b"          */
c =  D2C(32767,1)         /* Assigns 'FF'x to "c"            */
```

Notes:

1. The D2C function can not be used in asynchronous conditions.
2. The D2C function is equivalent to the HEX function and was added solely to accommodate those familiar with the REXX programming language.

TPNS Scripting Language Changes to Support D2C

None.

LASTPOS

```
LASTPOS(needle,haystack[,start])
```

Where

needle is a string expression.

haystack is a string expression.

start is an integer expression with a value from 1 to 32767. This is optional.

Returns

Integer

Function

The LASTPOS function returns the position of the last occurrence of one string, *needle*, in another, *haystack*. (See also the POS function.) If *needle* is the null string or is not found then the function returns a 0. By default the search starts at the last character of *haystack* and scans backward. You can override this by specifying *start*, the point at which the backward scan starts. The default for *start* is set equal to LENGTH(*haystack*) if the value specified is larger than LENGTH(*haystack*) or if it is omitted.

Examples

```
a =  LASTPOS(' ','ab cd ef gh')           /* Assigns 9 to "a"        */
b =  LASTPOS(' ','ab cd ef gh',7)         /* Assigns 6 to "b"        */
c =  LASTPOS('25','12352425352522',10)    /* Assigns 7 to "c"        */
```

Note: The LASTPOS function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support LASTPOS

The SET statement has been updated to allow LASTPOS as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to “SET - Set Counters Statement” on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=LASTPOS(needle_area,haystack_area[,start])
```

OVERLAY

```
OVERLAY(new,target[,[n][,[length][,pad]]])
```

Where

new is a string expression.

target is a string expression.

n is an integer expression. This is optional.

length is an integer expression. This is optional.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns

String

Function

The OVERLAY function returns the string *target*, which, starting at the *n*th character, is overlaid with the string *new*, padded or truncated to the value of *length*. Overlays may also extend beyond the end of the optional target string. If *n* is greater than the length of the target string, padding is added before the new string. The default for *n* is 1. If *length* is not specified, it defaults to a value equal to the length of the string expression *new*. The default value for *pad* is a blank (X'40').

Examples

```
a = OVERLAY(' ','abcdef',3)          /* Assigns 'ab def' to  "a"      */
b = OVERLAY('.','abcdef',3,2)        /* Assigns 'ab. ef'  to "b"      */
c = OVERLAY('qq','abcd')             /* Assigns 'qqcd'  to "c"        */
d = OVERLAY('qq','abcd',4)           /* Assigns 'abcqq'  to "d"       */
e = OVERLAY('123','abc',5,6,'+')     /* Assigns 'abc+123+++'  to "e"  */
```

Note: The OVERLAY function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support OVERLAY

The DATASAVE statement has been updated to allow OVERLAY as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=OVERLAY,
         TEXT=(target),
         INSERT=(new),
         POS=n,
         PLENG=length,
         PAD=pad,
         AREA=area
```

PATHID

```
PATHID()
```

Returns
String

Function
The PATHID function returns the name of the PATH statement currently being executed. This is the 1- to 8-character name coded on the PATH statement.

Examples
```
say 'The current path is' pathid()
```

Note: The PATHID function can be used in asynchronous conditions.

TPNS Scripting Language Changes to Support PATHID
The Data Field Options have been expanded to include a PATHID option. Refer to the *TPNS Language Reference* manual for information on other available data field options.

```
$PATHID$
```

Note: The name field from the PATH statement is inserted in the data. The blanks used to pad the name to eight characters are deleted.

POS

```
POS(needle,haystack[,start])
```

Where

needle is a string expression.

haystack is a string expression.

start is an integer expression with a value from 1 to 32767. This is optional.

Returns

Integer

Function

The POS function returns the position of one string, *needle*, in another, *haystack*. (See also the LASTPOS function.) If *needle* is the null string or is not found or if *start* is greater than the length of *haystack* then the function returns a 0. By default the search starts at the first character of *haystack*.

Examples

```
a =  POS(' ','ab cd ef gh')             /* Assigns 3 to "a"         */
b =  POS(' ','ab cd ef gh',7)           /* Assigns 9 to "b"         */
c =  POS('25','12352425352522',10)      /* Assigns 11 to "c"        */
```

Note: The POS function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support POS

The SET statement has been updated to allow POS as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to “SET - Set Counters Statement” on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=POS(needle_area,haystack_area,[start])
```

REVERSE

```
REVERSE(string)
```

Where
string is a string expression.

Returns
String

Function
The REVERSE function returns a string with the order of the characters reversed.

Examples
```
a =  REVERSE('abcdefghi')          /* Assigns 'ihgfedcba' to "a"     */
b =  REVERSE('517C'x)              /* Assigns '7C51'x to "b"         */
```

Note: The REVERSE function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support REVERSE
The DATASAVE statement has been updated to allow REVERSE as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=REVERSE,
         TEXT=(string),
         AREA=area
```

SAY

```
SAY message [TYPE 'ABRHD']
```

Where

message is a string expression.

Function

The SAY statement writes *message* to the TPNS operator console. The string that you specify can be any length; however, a maximum of 100 characters will be displayed.

If TYPE 'ABRHD' is specified, *message* will be written with an abbreviated header containing only a TPNS message number preceding the data. If omitted, *message* is written with network and device or LU names included in the header and using message number ITP113I or ITP137I.

Examples

```
say 'Beginning to execute procedure' msgtxtid()
say 'Beginning to execute procedure' msgtxtid() type 'ABRHD'
```

Note: When using the SAY statement, make sure that the system console is not overloaded with these messages.

TPNS Scripting Language Changes to Support the Enhanced SAY

The WTOABRHD statement has been added which writes user-specified data to the TPNS operator console using only an abbreviated header containing only a TPNS message number preceding the data. The format is as follows:

[*name***]** **WTOABRHD (***data...***)**

name

Function:	Specifies a name to be used when branching during message generation.
Format:	From one to eight alphanumeric characters.
Default:	None. This field is optional.

(*data...*)

Function:
Defines the data to be written to the TPNS operator.

Format:
You can enter any amount of data for this operand, but a maximum of 100 characters of user data will actually be written to the operator console. The data is enclosed by the text delimiting character specified on the MSGTXT statement, defaulting to left and right parentheses.

You can use the data field options (see Appendix A. ″Data Field Options″ in the *TPNS Language Reference* manual). Enter hexadecimal data by enclosing the digits within single quotes. To enter a single quote, the special control character (CONCHAR), or a text delimiting character (TXTDLM) as data, enter two of the characters. If two text delimiting characters are entered, they must be on the same statement (no continuation between the characters).

You can continue the data on the next line. However, if a single text delimiting character is detected in column 71, it indicates the end of the operand, and any data after column 71 is ignored.

Default:
None. If no data is entered, no user data will be included in the console message.

SNACMND

```
SNACMND(type,[arg1...arg5])
```

Where

type is STSN (set and test sequence numbers).

arg1,...,arg5 are the arguments for STSN.

Note: For further information on STL and other valid type values on the SNACMND statement, refer to the *Using the TPNS Structured Translator Language (STL) and the STL Translator* manual.

Type	1st Argument	2nd Argument	3rd Argument	4th Argument	5th Argument
STSN	*pseqact*	*pseqval*	*sseqact*	*sseqval*	*log_byte*

Argument	Format
pseqact	Is a string constant that can have the following values: **IGNORE** Specifies that this STSN command is to be ignored. **SET** Specifies that the primary-to-secondary sequence number of the secondary end user is to be set to *pseqval* operand value. **TEST** Specifies that the secondary end user must return its primary-to-secondary sequence number in the response RU. **TESTSET** Specifies that the primary-to-secondary sequence number of the control program (CP) manager is to be set to the *pseqval* operand value, and the secondary end user is to compare that value against its own and respond accordingly.
pseqval	Is an integer constant with a value from 0 to 65535.
sseqact	Is a string constant that can have the following values: **IGNORE** Specifies that this STSN command is to be ignored. **SET** Specifies that the primary-to-secondary sequence number of the secondary end user is to be set to *sseqval* operand value. **TEST** Specifies that the secondary end user must return its primary-to-secondary sequence number in the response RU. **TESTSET** Specifies that the primary-to-secondary sequence number of the control program (CP) manager is to be set to the *sseqval* operand value, and the secondary end user is to compare that value against its own and respond accordingly.
sseqval	Is an integer constant with a value from 0 to 65535.
log_byte	Is a 1-byte string constant or a string expression.

Function

The function of *type* for the SNA command STSN is set and test sequence numbers.

The arguments have a unique function for STSN. The function of the arguments are as follows:

pseqact	Specifies the action to be executed by the STSN receiver for the primary-to-secondary sequence number. The default *pseqact* value is SET.
pseqval	Specifies the primary-to-secondary sequence number value to be sent with the STSN. The default *pseqval* is 0.
sseqval	Specifies the secondary-to-primary sequence number value to be sent with STSN. The default *sseqval* is 0.
sseqact	Specifies the action to be executed by the STSN receiver for the secondary-to-primary sequence number. The default *sseqact* value is SET.
log_byte	Specifies the byte of user data to be associated with all data transmitted and received. The *log_byte* remains active until data is ″typed″ and transmitted with a TRANSMIT statement or until an INITSELF, TERMSELF, or another SNACMND statement is issued. This byte gives users of the Response Time Utility a way to identify transactions when gathering statistics by the various ″log_byte″ categories. The default *log_byte* is X'00'.

SPACE

```
SPACE(string,n[,pad])
```

Where

string is a string expression.

n is an integer expression with a value from 1 to 32767. This is optional.

pad is a 1-character string constant or 2-digit hexadecimal constant used for padding. This is optional.

Returns

String

Function

The SPACE function returns the blank-delimited words in *string* with *n* *pad* characters between each word. If *n* is 0, all blanks are removed. Leading and trailing blanks are always removed. The default for *n* is 1, and the default value for *pad* is a blank (X’40’).

Examples

```
a = SPACE('aa   bb    cc',1,'+')     /* Assigns 'aa+bb+cc' to "a"      */
b = SPACE(' ab  cd  ',2)             /* Assigns 'ab  cd' to "b"        */
c = SPACE('  a  a  ',9,'#')          /* Assigns 'a#########a' to "c"   */
```

Note: The SPACE function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support SPACE

The DATASAVE statement has been updated to allow SPACE as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=SPACE,
         TEXT=(string),
         PAD=pad,
         PLENG=n,
         AREA=area
```

STRIP

```
STRIP(string[,[option][,char]])
```

Where

string is a string expression.

option is a single character constant with one of the following values:

B or b	removes both leading and trailing characters from *string*
L or l	removes leading characters from *string*
T or t	removes trailing characters from *string*

char is a single character constant. This is optional.

Returns

String

Function

The STRIP function returns *string* with leading or trailing characters or both removed, based on the *option* you specify. The third argument, *char*, specifies the character to be removed. The default for *option* is B and the default for *char* is a blank (X’40’).

Examples

```
a = STRIP('   abc da   ')            /* Assigns 'abc da' to "a"       */
b = STRIP('aaaabcdefaaa','T','a' )    /* Assigns 'aaaabcdef '  to "b"  */
c = STRIP('aaaabcdefaaa','b','a' )    /* Assigns 'bcdef' to "c"        */
```

Note: The STRIP function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support STRIP

The DATASAVE statement has been updated to allow STRIP, STRIPL, and STRIPT as values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=STRIP|STRIPL|STRIPT,
         TEXT=(string),
         PAD=char,
         AREA=area
```

SUBWORD

```
SUBWORD(string,n[,length])
```

Where
string is a string expression.

n is an integer expression with a value from 1 to 32767.

length is an integer expression with a value from 1 to 32767. This is optional.

Returns
String

Function
The SUBWORD function returns the substring of *string* that starts at the *n*th word and is up to *length* blank-delimited words. If you omit *length*, it defaults to the number of remaining words in *string*. The returned string never has leading or trailing blanks, but includes all blanks between the selected words.

Examples
```
a = SUBWORD('Now is the time',2,2)     /* Assigns 'is the' to "a"      */
b = SUBWORD('Now is the time',3)       /* Assigns 'the time' to "b"    */
c = SUBWORD('Now is the time',5)       /* Assigns ''  to "c"           */
```

Note: The SUBWORD function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support SUBWORD
The DATASAVE statement has been updated to allow SUBWORD as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to "Operands for Datasave Statement" on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=SUBWORD,
         TEXT=(string),
         POS=n,
         PLENG=length,
         AREA=area
```

WORD

```
WORD(string,n)
```

Where

string is a string expression.

n is an integer expression with a value from 1 to 32767.

Returns

String

Function

The WORD function returns the *n*th blank-delimited word in *string* or returns the null string if fewer than *n* words are in *string*. This function is exactly equivalent to SUBWORD(*string*,*n*,1).

Examples

```
a = WORD('Now is       the time',3)     /* Assigns 'the' to "a"     */
b = WORD('Now is the time',5)           /* Assigns '' to "b"        */
```

Note: The WORD function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support WORD

The DATASAVE statement has been updated to allow SUBWORD as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to "Operands for Datasave Statement" on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=SUBWORD,
         TEXT=(string),
         POS=n,
         PLENG=1,
         AREA=area
```

WORDINDEX

```
WORDINDEX(string,n)
```

Where

string is a string expression.

n is an integer expression with a value from 1 to 32767.

Returns

Integer

Function

The WORDINDEX function returns the position of the first character in the *n*th blank-delimited word in *string* or returns 0 if fewer than *n* words are in *string*.

Examples

```
a = WORDINDEX('Now is the time',2)        /* Assigns 5 to "a"       */
b = WORDINDEX('Now is the time',3)        /* Assigns 8 to "b"       */
c = WORDINDEX('Now is the time',5)        /* Assigns 0 to "c"       */
```

Note: The WORDINDEX function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support WORDINDEX

The SET statement has been updated to allow WORDINDEX as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to “SET - Set Counters Statement” on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=WORDINDEX(string_area,n)
```

WORDPOS

WORDPOS(*phrase***,***string***[,***start***])**

Where

phrase is a string expression.

string is a string expression.

start is an integer expression with a value from 1 to 32767. This is optional.

Returns

Integer

Function

The WORDPOS function returns the word number of the first word of *phrase* found in *string* or returns 0 if *phrase* contains no words or if *phrase* is not found in *string*. Multiple blanks between words in either *phrase* or *string* are treated as a single blank for the comparison, but otherwise the words must match exactly. By default the search starts at the first word in *string*. You can override this by specifying *start*, the word at which to start the search.

Examples

```
a = WORDPOS(' is what','It is      what it is')  /* Assigns 2 to "a"    */
b = WORDPOS('it     is','It is what it is')      /* Assigns 4 to "b"    */
c = WORDPOS(' is ','It is what it is',3)        /* Assigns 5 to "c"    */
d = WORDPOS(' Is ','It is what it is')          /* Assigns 0 to "d"    */
```

Note: The WORDPOS function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support WORDPOS

The SET statement has been updated to allow WORDPOS as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to “SET - Set Counters Statement” on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=WORDPOS(phrase_area,string_area[,start])
```

WORDS

```
WORDS(string)
```

Where

string is a string expression.

Returns

Integer

Function

The WORDS function returns the number of blank-delimited words in *string*.

Examples

```
a = WORDS('Now is the time ')              /* Assigns 4 to "a"      */
b = WORDS('a b d  cd  e    f')             /* Assigns 6 to "b"      */
```

Note: The WORDS function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support WORDS

The SET statement has been updated to allow WORDS as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to "SET - Set Counters Statement" on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=WORDS(string_area)
```

X2B

```
X2B(hexstring)
```

Where

hexstring is a string expression containing hexadecimal characters.

Returns

String

Function

The X2B function returns a string, in character format, that represents *hexstring* converted to a binary string. The *hexstring* is a string of hexadecimal characters. It can be of any length. Each hexadecimal character is converted to a string of four binary digits. You can optionally include blanks in *hexstring* (at byte boundaries only, not leading or trailing) to aid readability; they are ignored. The returned string has a length that is a multiple of four, and does not include any blanks. If *hexstring* is null, the function returns a null string.

Examples

```
a = X2B('F1C2')                  /* Assigns '1111000111000010' to "a"  */
b = X2B('5A  A5')                /* Assigns '0101101010100101' to "b"  */
```

Note: The X2B function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support X2B

The DATASAVE statement has been updated to allow X2B as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=X2B,
         TEXT=(hexstring),
         AREA=area
```

X2C

```
X2C(hexstring)
```

Where
hexstring is a string expression containing hexadecimal characters.

Returns
String

Function
The X2C function returns a string, in character format, that represents *hexstring* converted to a character string. The returned string is half as many bytes as the original *hexstring*. The string *hexstring* can be any length. If necessary, it is padded with a leading 0 to make an even number of hexadecimal digits. You can optionally include blanks in *hexstring* (at byte boundaries only, not leading or trailing) to aid readability; they are ignored. If *hexstring* is null, the function returns a null string.

Examples
```
a = X2C('F1F2  F3')                    /* Assigns '123' to "a"      */
b = X2C('C1 C2 C3')                    /* Assigns 'ABC' to "b"      */
c = X2C('8140')                        /* Assigns 'a '  to "c"      */
```

Note: The X2C function can not be used in asynchronous conditions.

TPNS Scripting Language Changes to Support X2C
The DATASAVE statement has been updated to allow X2C as one of the values that can be coded for the FUNCTION operand (refer to page 330 of the *TPNS Language Reference* manual for information on the existing DATASAVE statement). Refer to “Operands for Datasave Statement” on page 69 for specific details on coding the DATASAVE statement as follows:

```
DATASAVE FUNCTION=X2C,
         TEXT=(hexstring),
         AREA=area
```

Operands for Datasave Statement

The following table indicates the valid operands on the DATASAVE statement for the specific function.

STL Function	Datasave Function	Text	Text2	Area	Count	Insert	Pad	Pleng	Pos
BITAND	BITAND	string1	string2	area			char		
BITOR	BITOR	string1	string2	area			char		
BITXOR	BITXOR	string1	string2	area			char		
B2X	B2X	binary_string		area					
CENTER	CENTER	string		area			char	length	
COPIES	COPIES	string		area	n				
DELWORD	DELWORD	string		area				length	n
D2C		$CNTRX, number_cntr [,n_constant]$		area					
OVERLAY	OVERLAY	target		area		new	char	length	n
REVERSE	REVERSE	string		area					
SPACE	SPACE	string		area			char	n	
STRIP	STRIP	string		area			char		
	STRIPL	string		area			char		
	STRIPT	string		area			char		
SUBWORD	SUBWORD	string		area				length	n
WORD	SUBWORD	string		area				1	n
X2B	X2B	hexstring		area					
X2C	X2C	hexstring		area					

Note: For the D2C function, refer to Appendix A, ″Data Field Options″ in the *TPNS Language Reference* manual for coding the TEXT operand on the DATASAVE statement.

The following definitions add to and revise the information on the DATASAVE statement included in the *TPNS Language Reference* manual.

```
FUNCTION={BITAND}
         {BITOR}
         {BITXOR}
         {B2X}
         {CENTER}
         {COPIES}
         {DELWORD}
         {OVERLAY}
         {REVERSE}
         {SPACE}
         {STRIP}
         {STRIPL}
         {STRIPT}
         {SUBWORD}
         {X2B}
         {X2C}
```

Function: Specifies that a string manipulation function be performed on the TEXT operand.

Format: You can code one of the following values for the FUNCTION operand.

BITAND	Specifies that the data coded for the TEXT and TEXT2 operands will be logically AND'ed together, bit by bit. If the PAD operand is specified, the shorter text data of TEXT and TEXT2 will be extended with the PAD character on the right before carrying out the logical operation.
BITOR	Specifies that the data coded for the TEXT and TEXT2 operands will be logically inclusive-OR'ed together, bit by bit. If the PAD operand is specified, the shorter text data of TEXT and TEXT2 will be extended with the PAD character on the right before carrying out the logical operation.
BITXOR	Specifies that the data coded for the TEXT and TEXT2 operands will be logically eXclusive-OR'ed together, bit by bit. If the PAD operand is specified, the shorter text data of TEXT and TEXT2 will be extended with the PAD character on the right before carrying out the logical operation.
B2X	Specifies that the data coded for the TEXT operand will be converted to hexadecimal.
CENTER	Specifies that the data coded for the TEXT operand will be centered with the PAD character around both ends until a length of PLENG is reached.
COPIES	Specifies that the data coded for the TEXT operand will be copied and concatenated by the number specified on the COUNT operand.
DELWORD	Specifies that the data coded for the TEXT operand will be deleted starting at the word corresponding to the value coded on the POS operand for a length of PLENG words.
OVERLAY	Specifies that the data coded for the INSERT operand is to be overlaid on the data specified by the TEXT operand. If the PLENG operand is also coded, the data specified by the INSERT operand is padded or truncated to that length before it is overlaid on the TEXT data. If the POS operand is also specified, the INSERT data is overlaid starting at that position in the TEXT data. If the PAD operand is specified, the specified PAD character is used, if necessary, to extend the TEXT or INSERT data to meet the POS or PLENG specifications respectively.
REVERSE	Specifies that the data coded for the TEXT operand will be reversed.
SPACE	Specifies that the data coded for the TEXT operand will have a space of length PLENG between each word. If the PAD operand is coded, the PAD value is used instead of a space.
STRIP	Specifies that the data coded for the TEXT operand will have both leading and trailing blanks removed. If the PAD operand is coded, the PAD value will be used as the character to remove instead of blanks.
STRIPL	Specifies that the data coded for the TEXT operand will have leading blanks removed. If the PAD operand is coded, the PAD value will be used as the character to remove instead of blanks.

STRIPT Specifies that the data coded for the TEXT operand will have trailing blanks removed. If the PAD operand is coded, the PAD value will be used as the character to remove instead of blanks.

SUBWORD Specifies that a substring of the data coded for the TEXT operand will start at the *n*th word coded by the *n* value of the POS operand and up to the PLENG value coded for the number of words. If the PLENG value is not coded, the default will be the number of remaining words in the data coded for the TEXT operand.

X2B Specifies that the data coded for the TEXT operand will be converted to binary. Each hexadecimal character is converted to a string of four binary digits. Blanks are ignored.

X2C Specifies that the data coded for the TEXT operand will be converted to character. A leading 0 will be added if necessary to make an even number of hexadecimal digits.

AREA={s|Ns|N±value|U±value|1}

Function: Specifies whether one of the save areas or user areas is to be used to save the data.

Format: For the AREA operand, you can enter one of the following options. The value for *value* can be any integer from 0 to 32766 or a counter specification whose value is within this range. Zero is the offset to the first byte of the field for positive offsets (+*value*) and the offset to the last byte of the field for negative offsets (-*value*).

s Specifies a device save area to be used to save the data, where *s* is an integer from 1 to 4095.

N*s* Specifies a network save area to be used to save the data, where *s* is an integer from 1 to 4095.

N±*value* Specifies a network user area to be used to save the data, where +*value* is the offset from the start of the user area and -*value* is the offset back from the end of the user area.

U±*value* Specifies a device user area to be used to save the data, where +*value* is the offset from the start of the user area and -*value* is the offset back from the end of the user area.

Note: If *value* specifies an offset that is outside the user area, no data is saved and an informational message is written to the log data set.

Default: 1, indicating device save area number 1.

COUNT=*value*

Function: When FUNCTION=COPIES is coded, *value* specifies the number of copies to create.

Note: This operand is only valid when the TEXT operand is coded and when FUNCTION=COPIES is coded.

Format: *value* can be an integer from 1 to 32767 or a counter specification whose value is within this range.

Default: None. This operand is required when FUNCTION=COPIES.

INSERT=(*data***)**

Function: Specifies the string data to be inserted into, or overlaid on, the data coded in the TEXT operand.

Note: This operand is only valid when the TEXT operand is coded and when FUNCTION=INSERT or FUNCTION=OVERLAY is coded.

Format: You can code any amount of data for this operand. If this resulting string is longer than the space available in the save area, the data is truncated and a message is written to the log data set.

The data is enclosed by the text delimiting character specified on the MSGTXT statement. (The defaults are left and right parentheses.) You can also continue the data.

You can use the data field options (see Appendix A, ″Data Field Options″ in the *TPNS Language Reference* manual). To enter hexadecimal data, enclose the digits within single quotes. To enter a single quote, a special control character (CONCHAR), or a text delimiter (TXTDLM) as data, enter two of the characters. If you enter two text delimiting characters, they must be on the same statement; you cannot continue the statement between the characters.

Default: None.

PAD={*char***|blank}**

Function: Specifies the character to be used as padding when BITAND, BITOR, BITXOR, CENTER, INSERT, LEFT, OVERLAY, RIGHT, SPACE, STRIP, STRIPL, STRIPT or TRANSLATE is coded for the FUNCTION operand.

Note: This operand is only valid when the TEXT operand is coded along with the specified functions.

Format: *char* is a 1-character string constant or a 2-character hexadecimal constant. When you use a special character (such as a quote or parenthesis) as padding, do not double it.

Default: When FUNCTION=BITAND is coded, the default is X’FF’. When BITOR or BITXOR is coded for the FUNCTION operand, the default is X’00’. Otherwise, the default is blank (X’40’).

PLENG=*value*

Function: When INSERT or OVERLAY is coded for the FUNCTION operand, *value* specifies the number of characters to which the data specified by the INSERT operand is truncated or padded when data is inserted into, or overlaid on, the data specified by the TEXT operand. When FUNCTION=DELETE is coded, *value* specifies the number of characters to be deleted. When FUNCTION=SPACE is coded, *value* specifies the number of blanks or characters specified by the PAD operand, to place between the words. If *value* is 0, all blanks are removed. Leading and trailing blanks are always removed. When CENTER, LEFT or RIGHT is coded for the FUNCTION operand, *value* specifies the number of characters to which the data coded for the TEXT operand is truncated or padded. When DELWORD or SUBWORD is coded for the FUNCTION operand, *value* specifies the number of words to delete or return.

Note: This operand is only valid when the TEXT and FUNCTION operands are coded.

Format: *value* can be an integer from 0 to 32767 or a counter specification whose value is within the range.

Default: When INSERT or OVERLAY is coded for the FUNCTION operand, the default is the length of the data specified by the INSERT operand. When FUNCTION=DELETE is coded, the default is the length of the TEXT data after the position specified by the POS operand. When FUNCTION=SPACE is coded, the default is 1. When CENTER, LEFT or RIGHT is coded for the FUNCTION operand, there is no default, and a value is required. When DELWORD or SUBWORD is coded for the FUNCTION operand, the default is the number of words left in the TEXT data after the word in the position specified by the POS operand.

POS=*value*

Function: When FUNCTION=INSERT is coded, *value* specifies the position in the TEXT data after which the data coded for the INSERT operand is inserted. When FUNCTION=OVERLAY is coded, *value* specifies the first position in the TEXT data to be overlaid with the data coded for the INSERT operand. When FUNCTION=DELETE is coded, *value* specifies the first position in the TEXT data to be deleted. When DELWORD or SUBWORD is coded for the FUNCTION operand, *value* specifies the word position in the TEXT data.

Note: This operand is only valid when the TEXT operand is coded and when DELETE, DELWORD, INSERT, OVERLAY or SUBWORD is coded on the FUNCTION operand.

Format: When FUNCTION=INSERT is coded, *value* can be an integer from 0 to 32766 or a counter specification whose value is within that range. If *value* is greater than the length of the data specified in the TEXT operand, pad characters are inserted after the TEXT data, with the INSERT data following. If *value* is zero, the data coded in the INSERT operand is inserted before the beginning of the data coded in the TEXT operand.

When FUNCTION=OVERLAY is coded, *value* can be an integer from 1 to 32767 or a counter specification whose value is within that range. If *value* is greater than the length of the data specified in the TEXT operand, pad characters are added to the TEXT data.

When DELETE, DELWORD or SUBWORD is coded on the FUNCTION operand, *value* can be an integer from 1 to 32767 or a counter specification whose value is within that range.

Default: When FUNCTION=INSERT is coded, the default is 0. When FUNCTION=OVERLAY is coded, the default is 1. When DELETE, DELWORD or SUBWORD is coded for the FUNCTION operand, there is no default and a value must be specified.

TEXT=([*data***])**

Function: Specifies the text data to be saved in the save area or user area based on the function specified for the FUNCTION operand.

Note: If this operand is coded, the LENG and LOC operands are not valid.

Format: You can code any amount of data for this operand. If the data is longer than the space available in the save area or user area, it is truncated and an informational message is written to the log data set. The data is enclosed by the text delimiting characters specified on the MSGTXT statement. (The default is the left and right parentheses.) You can also continue the data. You can use the data field options (see Appendix A, ″Data Field Options″ in the *TPNS Language Reference* manual). Enter hexadecimal data by enclosing the digits within single quotes. To enter a single quote, a special control character (CONCHAR), or a text delimiting character (TXTDLM) as data, enter two of the characters. If two text delimiting characters are entered, they must be on the same statement (no continuation between the characters).

If the TEXT operand specifies only two text delimiting characters with no intervening data (for example, TEXT=(),), a clear function is executed. Specifying TEXT=() also frees a dynamically allocated save area. If the AREA operand specifies a save area number, the length of data saved in that save area is set to zero. If the AREA operand specifies an offset into a user area, the user area is usually cleared to binary zeros from the specified offset to the end. This will not be the case if the FUNCTION operand or CONVERT=YES has also been specified. In those cases, the user area is unchanged.

Default: None. This operand is optional.

TEXT2=([*data*])
Function: Specifies the text data to be used when BITAND, BITOR or BITXOR is coded for the FUNCTION operand.

Note: If this operand is coded, the LENG and LOC operands are not valid.

Format: See the TEXT description above.

Default: None. This operand is optional.

SET - Set Counters Statement

[*name*] SET *cntr*=*option*[**,...**]

The following definitions add to the information on the SET statement included in the *TPNS Language Reference* manual.

The SET statement performs the following new functions:

- Sets the counter to the position of the last occurrence of specified data in a save area
- Sets the counter to the position of the first occurrence of specified data in a save area
- Sets the counter to the position of the first character in the *n*th blank-delimited word in a save area
- Sets the counter to the word number of the first word of specified data found in a save area
- Sets the counter to the number of blank-delimited words in a save area or a user area
- Sets the counter to the number of text data items on the queue.

Note: To use the SET statement, you must code at least one operand.

name
Function: Specifies a name to be used when branching during message generation.

Format: From one to eight alphanumeric characters.

Default: None. This field is optional.

cntr

Function: Specifies which counter is to be set.

Note: You can set the same counter multiple times with the same SET statement. If you code multiple operands, the counters will be set in the order specified.

Format: The valid values for *cntr* are NSEQ, LSEQ, TSEQ, DSEQ, NC*n*, LC*n*, TC*n*, and DC*n*, where n is an integer from 1 to 4095. These values represent the sequence counters and the index counters for the network, line, terminal, and device levels.

Default: None. You must code a value for *cntr*.

Notes:

1. For terminal types that do not allow devices, TSEQ and DSEQ are the same sequence counter, and TC*n* and DC*n* are the same index counter.
2. For domain simulation, TSEQ, LSEQ, TC*n*, and LC*n* will reference a single set of counters allocated to each simulated subarea. The terminal and line counters will be the same.
3. For VTAMAPPL LU simulation, TSEQ, LSEQ, TC*n*, and LC*n* will reference a single set of counters allocated to each VTAM application (VTAMAPPL).
4. For CPI-C transaction program simulation:
   - LSEQ and LC*n* will reference a single set of counters allocated to each APPC LU.
   - TSEQ and TC*n* will reference a single set of counters allocated to each transaction program.
   - DSEQ and DC*n* will reference a single set of counters allocated to each transaction program instance.
5. For Type 2.1 node simulation, TSEQ, LSEQ, TC*n*, and LC*n* will reference a single set of counters allocated to each Type 2.1 node (PU21).
6. See Appendix D, ″Counters and Switches″ in the *TPNS Language Reference* manual, for valid counter and switch specifications.

option

Function: Specifies how the counter is to be set.

Format: For the *option*, you can code one of the following values:

LASTPOS(*needle_area***,***haystack_area***[,***start***])**

Set the counter to the position of the last occurrence of the data specified in the *needle_area* save area within the data specified in the *haystack_area* save area. Returns 0 if data in *needle_area* is the null string or is not found. By default the search starts at the last character in *haystack_area* and scans backward. You can override the default by specifying *start*, the position to start the backward scan. *start* defaults to the length of *haystack_area* if it is not specified or it is larger than the length of *haystack_area*. *start* can be an integer from 1 to 32767 or a counter specification whose value is within this range.

POS(*needle_area***,***haystack_area***[,***start***])**

Set the counter to the position of the first occurrence of the data specified in the *needle_area* save area within the data specified in the *haystack_area* save area. Returns 0 if data in *needle_area* is the null string or is not found, or if *start* is greater than the length of *haystack_area*. By default the search starts at the first character in *haystack_area*. You can override the default by specifying *start*, the position to start the search. *start* can be an integer from 1 to 32767 or a counter specification whose value is within this range.

QUEUED([*queue_name***])**

Set the counter to the number of text data items on the queue, *queue_name*. *queue_name* can be specified as either a static one to eight character alphameric queue name or a save or user area + offset definition. The default for *queue_name* is a unique value for each simulated device.

WORDINDEX(*string_area***,***n***)**

Set the counter to the position of the first character in the *n*th blank-delimited word in the *string_area* save area. Returns 0 if fewer than *n* words are in *string_area*. *n* can be an integer from 1 to 32767 or a counter specification whose value is within this range. *n* is required.

WORDPOS(*phrase_area***,***string_area***[,***start***])**

Set the counter to the word number of the first word in the *phrase_area* save area, found in the *string_area* save area. Returns 0 if *phrase_area* contains no words or if the data in *phrase_area* is not found in *string_area*. Multiple blanks between words in either *phrase_area* or *string_area* are treated as a single blank for comparison, otherwise the words must match exactly. By default, the search starts at the first word in *string_area*. You can override the default by specifying *start*, the word at which to start the search. *start* can be an integer from 1 to 32767 or a counter specification whose value is within this range.

WORDS(*string_area***)**

Set the counter to the number of blank-delimited words in the *string_area* save area or user area.

Default: None. You must code a value for *option*.

Named Queue Support

Named queue support has been added to provide an easier method for passing data between simulated resources by using named queues. The following statements and functions have been added as part of this new support. Each new statement and function listed below is detailed in the pages that follow including a section which lists the equivalent TPNS scripting language for that specific STL statement or function.

- A PULL function which returns the next text/string data item from the queue.
- A PUSH statement which adds a text/string data item to the queue on a last in, first out (LIFO) basis.
- A QUEUE statement which adds a text/string data item to the queue on a first in, first out (FIFO) basis.
- A QUEUED function which returns the number of text/string data items on the queue.

PULL

```
PULL([queue_name])
```

Where

queue_name is a string expression consisting of 1 to 8 alphanumeric characters. This is optional.

Returns

String

Function

The PULL function returns the next text/string item from *queue_name*.

If the specified *queue_name* is a nonconstant expression, the first 8 characters of the string are used. If the string is shorter than 8 characters, the available characters are used. When specifying *queue_name* within the PULL function, the specified name must exactly match the name initially used to QUEUE or PUSH the text/string item.

If you specify a string variable, it cannot be the name of one of the STL reserved variables (for example, BUFFER). If not specified, *queue_name* defaults to a unique value assigned to each device.

Examples

```
user_queue  = 'UseridQ'          /* Assigns 'UseridQ' to "user_queue"  */
queue '111111' TO user_queue     /* Places '111111' on queue 'UseridQ' */
queue '222222' TO 'UseridQ'      /* Places '222222' on queue 'UseridQ' */
queue 'ABCD'                     /* Places 'ABCD' on unique device Q   */
a = PULL('UseridQ')              /* Assigns '111111' to "a"            */
b = PULL(user_queue)             /* Assigns '222222' to "b"            */
c = PULL()                       /* Assigns 'ABCD' to "c"              */
```

Notes:

1. The PULL function can be used in asynchronous conditions.
2. The named queue structure and text/string data items are allocated dynamically by TPNS and deleted as the queue is emptied.

TPNS Scripting Language Changes to Support PULL

The Data Field Options have been expanded to include a PULL option. Refer to the *TPNS Language Reference* manual for information on other available data field options.

```
$PULL[,queue_name]$
```

Notes:

1. *queue_name* can be specified as either a static one to eight character alphanumeric queue name or an area+offset definition.
2. The *queue_name* field conforms to the same rules as event names.

PUSH

```
PUSH string [TO queue_name]
```

Where

string is a string expression.

queue_name is a string expression consisting of 1 to 8 alphanumeric characters. This is optional.

Function

The PUSH statement places *string* on *queue_name* on a last in first out (LIFO) basis.

If the specified *queue_name* is a nonconstant expression, the first 8 characters of the string are used. If the string is shorter than 8 characters, the available characters are used. When specifying *queue_name* with the PUSH statement, the test/string item will be placed on a queue exactly matching *queue_name*. If you specify a string variable, it cannot be the name of one of the STL reserved variables (for example, BUFFER). If not specified, *queue_name* defaults to a unique value assigned to each device.

Examples

```
Qname = 'QUEUE1'              /* Assigns 'QUEUE1' to "Qname"       */
push 'ABCD'                   /* Places 'ABCD' on unique device Q  */
push '1234567'  TO Qname      /* Places '1234567' on queue 'QUEUE1' */
push '7654321'  TO 'QUEUE1'   /* Places '7654321' on queue 'QUEUE1' */
a = PULL(Qname)               /* Assigns '7654321' to "a"          */
b = PULL('QUEUE1')            /* Assigns '1234567' to "b"          */
c = PULL()                    /* Assigns 'ABCD' to "c"             */
```

Notes:

1. The named queue structure and text/string data items are allocated dynamically by TPNS and deleted as the queue is emptied.

TPNS Scripting Language Changes to Support PUSH

The PUSH statement has been added to the TPNS Scripting Language. Refer to the *TPNS Language Reference* manual for information on other available data field options.

```
PUSH  TEXT=(text_data)[,Q=queue_name]
```

Notes:

1. *queue_name* can be specified as either a static one to eight character alphanumeric queue name or an area+offset definition.
2. The *queue_name* field conforms to the same rules as event names.

QUEUE

```
QUEUE string [TO queue_name]
```

Where

string is a string expression.

queue_name is a string expression consisting of 1 to 8 alphanumeric characters. This is optional.

Function

The QUEUE statement adds *string* to *queue_name* on a first in, first out (FIFO) basis.

If the specified *queue_name* is a nonconstant expression, the first 8 characters of the string are used. If the string is shorter than 8 characters, the available characters are used. When specifying *queue_name* with the QUEUE statement, the test/string item will be placed in a queue by the exact name as *queue_name*. If you specify a string variable, it cannot be the name of one of the STL reserved variables (for example, BUFFER). If not specified, *queue_name* defaults to a unique value assigned to each device.

Examples

```
queue 'AAAA' TO 'Queue1'        /* Places 'AAAA'  in queue 'Queue1'   */
queue 'BBBB' TO 'QUEUE1'        /* Places 'BBBB'  in queue 'QUEUE1'   */
a = PULL('QUEUE1')              /* Assigns 'BBBB' to "a"              */
```

Notes:

1. The named queue structure and text/string data items are allocated dynamically by TPNS and deleted as the queue is emptied.

TPNS Scripting Language Changes to Support QUEUE

The QUEUE statement has been added to the TPNS Scripting Language. Refer to the *TPNS Language Reference* manual for information on other available data field options.

```
QUEUE TEXT=(text_data)[,Q=queue_name]
```

Notes:

1. *queue_name* can be specified as either a static one to eight character alphanumeric queue name or an area+offset definition.
2. The *queue_name* field conforms to the same rules as event names.

QUEUED

```
QUEUED([queue_name])
```

Where

queue_name is a string expression consisting of 1 to 8 alphanumeric characters. This is optional.

Returns

Integer

Function

The QUEUED function returns the number of text/string items on *queue_name*.

If the specified *queue_name* is a nonconstant expression, the first 8 characters of the string are used. If the string is shorter than 8 characters, the available characters are used. When specifying *queue_name* within the QUEUED function, the specified name must exactly match the name initially used to QUEUE or PUSH the text/string item. If you specify a string variable, it cannot be the name of one of the STL reserved variables (for example, BUFFER). If not specified, *queue_name* defaults to a unique value assigned to each device.

Examples

```
Q_name = 'I'||devid()                /* This example will place   */
Do i = 1 to 5                        /* five entries on a queue.  */
   Queue char(i) to Q_name           /* Then the queue will be    */
End                                  /* read until it is empty.   */
Do while queued(Q_name) > 0
   Say 'Queue item' pull(Q_name)
End
```

Notes:

1. The QUEUED function can not be used in asynchronous conditions.
2. The named queue structure and text/string data items are allocated dynamically by TPNS and deleted as the queue is emptied.

TPNS Scripting Language Changes to Support QUEUED

The SET statement has been updated to allow QUEUED as one of the options that can be coded (refer to page 410 of the *TPNS Language Reference* manual for information on the existing SET statement). Refer to "SET - Set Counters Statement" on page 74 for specific details on coding the SET statement as follows:

```
SET cntr=QUEUED([queue_name])
```

Notes:

1. *queue_name* can be specified as either a static one to eight character alphanumeric queue name or an area+offset definition.
2. The *queue_name* field conforms to the same rules as event names.

Increased Scripting Resources

STL has been updated to expand its support for the number of bits, strings, integers and onin/onouts which can be used when creating STL programs.

BIT

STL has been expanded to assign bit variables to switches up to number 4095 as required for STL program translation. STL ALLOCATE statements can now specify network, terminal, and device switches up to this new increased limit.

Note: NSW32 cannot be used with ALLOCATE because it is used for special purposes in the STL generated code.

The Alter switch operator command has been updated to support switch numbers up to 4095.

The Query operator command has been updated to allow a single switch setting to be queried. The following example illustrates this enhanced support.

```
Q  SLU,SW25
ITP143I  Name=SLU-1  SW25=OFF
```

Note: TPNS will allocate switches (minimum 32) up to the maximum switch referenced in a script associated with the network for each level of switches. For example, if SW4095 and TSW77 are the maximum switches referenced in a script, 4095 device switches will be allocated for each device level resource in the network and 77 terminal switches will be allocated for each terminal level resource in the network. Only those switches allocated can be Altered or Queried.

INTEGER

STL has been expanded to assign integer variables to index counters up to number 4095 as required for STL program translation. STL ALLOCATE statements can now specify network, line, terminal, and device index counters up to this new increased limit. Additionally, TPNS now supports network, line, terminal, and device sequence counters up to 4095.

The Alter counter operator command has been updated to support index counter numbers up to 4095.

The Query operator command has been updated to allow a single counter value to be queried. The following example illustrates this enhanced support.

```
Q  SLU,DC3
ITP143I  Name=SLU-1   DC3=0
```

Note: TPNS will allocate index counters (minimum 3) up to the maximum index counter referenced in a script associated with the network for each level of counters. For example, if DC4095 and TC77 are the maximum numbered index counters referenced in a script, all device counters from DSEQ to DC4095 will be allocated for each device level resource in the network and all terminal counters from TSEQ to TC77 will be allocated for each terminal level resource in the network. Only those counters allocated can be Altered or Queried.

STRING

STL has been updated to assign string variables to save areas up to number 4095 as required for STL program translation. STL ALLOCATE statements can now also specify network and device save areas up to this new increased limit.

The Query and Alter save area operator commands have been updated to support network and device save areas numbers up to 4095.

Note: TPNS dynamically allocates save areas, numbered above the save area operand value, when they are referenced in the script.

ONIN/ONOUT

STL has been updated to assign ONINs/ONOUTs to input/output IFs up to number 4095 as required for STL program translation. Also, the STL @IFNUM statement has been updated to accept a value from 1 to 4095.

Date and Time Stamp

A TPNS scripting language comment specifying the date and time of the execution of the STL program is added to each generated MSGTXT deck. This comment immediately follows the generated MSGTXT statement and reads as follows:

`*GENERATED FROM STL SOURCE -` *mmmmmmmmm dd, yyyy, hh:mm*

where:

mmmmmmmmm is the month, ***dd*** is the day, ***yyyy*** is the year, ***hh*** is the hour, and ***mm*** is the minute of the execution of STL.

The time stamp is in 24 hour format. All decks generated during a single execution of STL will have the same date and time in the generated comment. For example:

```
*GENERATED FROM STL SOURCE - OCTOBER 13, 2001, 13:13
```

There is no option associated with this new comment. Anytime a TPNS Message Deck is generated using ITPSTL, this comment is included.

NOIMPLICIT Option

If NOIMPLICIT is specified as an execution parameter for STL, implicit definitions of BIT, INTEGER, or STRING variables are considered errors and are flagged with a new error message (ITP3206I). Refer to “Appendix B. Messages and Return Codes” on page 105 for a detailed description of this new error message. Implicit definitions are most commonly found in an assignment statement where a previously unused and undefined variable is assigned a value of a particular type. They can also be encountered when previously unused variables are used as output variables on CPI-C STL statements or on the UTBLSCAN function.

If NOIMPLICIT is not specified, implicitly defined variables are assigned a type based on the context of the first usage of the variable.

Variables are explicitly defined by using ALLOCATE statements or BIT, INTEGER, or STRING statements.

The NOIMPLICIT option is particularly useful in avoiding inadvertent resource conflicts when STL programs are developed separately using a common set of variables distributed in a common member which is included by all such programs.

A modification has also been made to the TPNS/ISPF Interface to allow the specification of this option when executing STL from that interface. A new Control Option has been added to the second panel of the STL execution panels. It is the

Allow Implicit Variables option and is specified as **Y** for Yes or **N** for No. The default value is **Y** for compatibility with previous releases.

Delay Cancellation

STL has been updated to enable you to write scripts that account for the possibility that long delays may need to be cancelled based on operator or other input. STL now allows you to specify CANCEL DELAY or CANCEL SUSPEND as an asynchronous subset action following an ON SIGNALED statement. Below is an example of how a delay cancellation statement might be coded:

```
ON SIGNALED(event_name) THEN CANCEL DELAY
```

The STL Translator has also been modified to generate a new DLYCNCL action when CANCEL DELAY or CANCEL SUSPEND is specified on ONIN, ONOUT, or ON SIGNALED statements.

DLYCNCL Action

The TPNS scripting language has been updated with a new DLYCNCL action to support the delay cancellation feature added to STL. DLYCNCL cancels any active or pending intermessage delay. DLYCNCL is permitted on the THEN and ELSE operands of IF statements and the THEN operand of ON statements. The DLYCNCL action is identical to that taken when DELAY=CANCEL is coded on an input or output IF statement and the THEN action is taken. However, the DLYCNCL action can be taken independently of other actions and on other statement types.

MSGTRACE and STLTRACE messages have been updated to document the cancellation of delays as a result of the DLYCNCL action.

Verify Record Reports

The verify record reports have been adjusted to allow for additional counters, switches, and save areas. Because of formatting changes, the expected and actual value fields are now 29 instead of 30 bytes. A sample verify report follows:

```
                             VERIFICATION REPORT
     DESCRIPTION        LOCATION    LENG COND       EXPECTED VALUE           ACTUAL VALUE
====================== =========== ===== == ============================= ===================
SA1 OK                 1+0            11 EQ SAVE AREA 1                   SAVE AREA 1
N4095 OK               N4095+0        18 EQ NET SAVE AREA 4095            NET SAVE AREA 4095
DC1 OK                 DC1               EQ 1                             1
DC4095 OK              DC4095            EQ 4095                          4095
SW1 OK                 SWITCH               SW1                           1
NSW4095 OK             SWITCH               NSW4095                       1
SW1&TSW666&NSW4095 OK  SWITCHES             SW1&TSW666&NSW4095            111
NSW3000|SW99|TSW666 OK SWITCHES             NSW3000|SW99|TSW666           001
SW1|..|SW7 OK          SWITCHES             SW1|SW2|SW3|SW4|SW5|SW6|SW7+ 1000000001000000
SW10|..|SW7 OK         SWITCHES             SW10|SW2|SW3|SW4|SW5|SW6|SW7+ 0000000001000000
```

New Exit Interface Routine Requests

The following new exit interface routine requests have been added to support the additional counters, save areas, and switches:

X'5A' This request returns the address of a specific network save area, the address of the length of the data in the network save area, and the address of the length of the network save area. The lengths are two-byte binary numbers.

Note: The user exit must place the network save area number (1–4095) in the first two bytes of the return area before calling the exit interface routine.

X'5B' This request allocates a new network save area for the size requested and frees the existing network save area if currently allocated. The user exit must place the network save area number (1–4095) in the first two bytes of the return area and the desired network save area size (1–32767) in the third and fourth bytes of the return area. If the network save area allocation is successful, the X'5A' request information is returned.

X'5C' This request frees a currently allocated network save area. The user exit must place the network save area number (1–4095) in the first two bytes of the return area.

X'5D' This request increases the size of a currently allocated network save area while retaining any saved data in the increased network save area. The user exit must place the network save area number (1–4095) in the first two bytes of the return area and the desired network save area size (1–32767) in the third and fourth bytes of the return area. If the network save area increase is successful, the X'5A' request information is returned.

X'5E' This request returns the address of the number of network index counters. This is a two-byte field.

X'5F' This request returns the address of the number of network switches. This is a two-byte field.

X'74' This request returns the address of the number of line index counters. This is a two-byte field.

X'94' This request returns the address of the number of terminal index counters. This is a two-byte field.

X'95' This request returns the address of the number of terminal switches. This is a two-byte field.

X'C2' This request returns the address of a specific device save area, the address of the length of the data in the device save area, and the address of the length of the device save area. The lengths are two-byte binary numbers.

Note: The user exit must place the device save area number (1–4095) in the first two bytes of the return area before calling the exit interface routine.

X'C3' This request allocates a new device save area for the size requested and frees the existing device save area if currently allocated. The user exit must place the device save area number (1–4095) in the first two bytes of the return area and the desired device save area size (1–32767) in the third and fourth bytes of the return area. If the device save area allocation is successful, the X'C2' request information is returned.

X'C4' This request frees a currently allocated device save area. The user exit must place the device save area number (1–4095) in the first two bytes of the return area.

X'C5' This request increases the size of a currently allocated device save area while retaining any saved data in the increased device save area. The user exit must place the device save area number (1–4095) in the first two bytes of the return area and the desired device save area size (1–32767) in the third and fourth bytes of the return area. If the device save area increase is successful, the X'C2' request information is returned.

X'C6' This request returns the address of the number of device index counters. This is a two-byte field.

X'C7' This request returns the address of the number of device switches. This is a two-byte field.

Note: Return code 44 is set when the save area number is greater than 4095.

Variable Parameter Data for Script User Exit

The *parm_list* argument on the USEREXIT STL statement can now be any type of string expression rather than just a string constant.

The PARM operand on the EXIT statement in the TPNS scripting language can now specify either a save area number or a constant data string. The syntax of this operand is as follows:

`PARM=`*s*|`N`*s*|(*data...*)

where **s** is a save area number, **N** indicates a network save area, and **(***data...***)** is a constant string of 100 bytes or less.

The only change in the interface to the exit routines is that with a save area, the data can be of any length up to the maximum length of the save area and is not limited to 100 bytes.

Chapter 5. Miscellaneous Enhancements

This section outlines a wide range of updates and improvements included in the TPNS Version 3 Release 5 function and service enhancements.

Enhancements to CPI-C Protocol Support

Support has been added for CPI-C single sessions when the control mode is not defined on the partner LU. Under normal circumstances, the TPNS CPI-C function uses parallel sessions when establishing conversations with partner LUs. Parallel sessions consist of a user session and a control session. The control session is used by the LU 6.2 control mode SNASVCMG to accomplish session control tasks. On rare occasions, a TPNS user might want to establish a CPI-C session that has a user session, but no control session. This type of session is referred to as a single session.

TPNS 3.5 or TPNS 3.5 Service Level 9711

Using TPNS 3.5 or TPNS 3.5 Service Level 9711, single sessions can be used by the TPNS CPI-C function if the LU 6.2 control mode (SNASVCMG) is defined on the partner LU and the TPNS network specifies the following setting for each mode that is to use a single session:

```
CNOS=((LUNAME=lu,MODENAME=mode,SESSIONS=1,CWL=1,CWP=0))
```

If the TPNS network specifies a single session using the CNOS setting in the previous example, and the SNASVCMG mode is not defined on the partner LU, TPNS message ITP4014I is issued indicating the CNOS command has failed.

TPNS 3.5 Service Level 0110

With the new support added in TPNS 3.5 Service Level 0110, single sessions can be used even if the SNASVCMG mode is not defined on the partner LU. To use single sessions, specify the CNOS setting as shown in the previous example for each mode that is to use a single session. TPNS 3.5 Service Level 0110 will recognize from the CNOS definition that a single session is desired, and communicates this to VTAM. VTAM then establishes a user session between the local and partner LUs, but no control session is established.

ITPFIOX File I/O User Exit

ITPFIOX is a TPNS message generation exit (EXIT MODULE=ITPFIOX) and network control exit (NCTLEXIT=ITPFIOX) providing sequential file I/O support to TPNS scripts.

QSAM is used to perform the actual file I/O operations. Storage for the DCB is allocated below the 16 MB line. The file handle, DCBE, and other control blocks are allocated above the 16 MB line. The DCBE is coded with RMODE31=BUFF which allows QSAM to allocate the block buffers above the 16 MB line. GET locate and PUT locate modes are used to access the record data.

Data sets must be preallocated and partitioned data sets are supported through member reference. The exit dynamically allocates the DD statement required for the data set unless the data set name is specified as DDNAME=*ddname* using an existing DD statement. Existing DCB attributes are used when data sets are opened for input. DCB attributes are set when data sets are opened for output.

Once a file handle is available after an OPEN, any simulated device in any active network can issue file I/O requests by passing the file handle value to the user exit.

ITPFIOX Syntax and Use

The syntax for ITPFIOX is as follows:

```
EXIT MODULE=ITPFIOX,
     PARM=(file_request rc_counter handle_sa# [data_sa#] [recfm] [blksize] [lrecl])
```

Where:

- *file_request* is one of the following:

OPENI	Open the data set for input
OPENO	Open the data set for output
OPENA	Open the data set for output append
READ	Read record
WRITE	Write record
CLOSE	Close the data set

- *rc_counter*, the return code counter name, is one of the following:

  DC1-DC4095

  NC1-NC4095

  The counter contains the return code after each file I/O request completes.

  The following codes are set:

0	OK
1	Record truncated, READ or WRITE
2	READ EOF
8	Parameter error
12	Request error
16	Handle error
20	Data error
24	DD allocation error
28	GETMAIN error
32	OPEN error
36	READ error
40	WRITE error
44	CLOSE error
48	Allocate data save area error
52	DD clear error
56	Find data save area error
60	Interface exit routine error
64	DCB values (*recfm*, *blksize*, or *lrecl*) error

68	Invalid member name
72	Data set not found
76	Data set in use
80	Invalid data set name
84	DD allocation, unexpected error
88	DSORG error, PO without member or PS with member
92	OPENA (append) for member of PDS
96	WRITE null record
100	DD name error

- *handle_sa#*, the file handle save area name, is one of the following:

  1-4095

  NI-N4095

  A four-byte address is saved in the handle save area when the file is opened. This value must be returned in the specified save area for all the other file I/O requests. This value is validated to avoid errors.
- *data_sa#*, the data save area name, is one of the following:

  1-4095

  N1-N4095

  For OPENI, OPENO, and OPENA, this save area contains the MVS or VM data set name or DDNAME=*ddname* keyword.

  For WRITE, this save area contains the record data to be written.

  For READ, this save area contains the record read from the data set.

  For CLOSE, this save area is not required.
- *recfm* is either **VB** or **FB**
- *blksize* is **1-32760**
- *lrecl* is **1-32760**

Notes:

1. When data sets are opened for output, the following DCB default values are set if *recfm*, *blksize*, and *lrecl* are not specified.
   ```
   RECFM=VB     BLKSIZE=23476     LRECL=23472
   ```
2. For VB, *lrecl* can be a maximum of four bytes less than *blksize*.
3. For FB, *blksize* must be a multiple of *lrecl*.
4. Code `NCTLEXIT=ITPFIOX` and the open files in a network will be closed when the network is cancelled or reset.
5. OPENA (append) is not accepted for a partitioned data set.
6. When DDNAME=*ddname* is specified as the data set name, the file is OPENed against the DD name specified without allocating a DD statement.

   OPENO and OPENA are equal because the disposition of the data set is controlled by the DISP= value on the DD statement.
7. DISP=SHR is set on the DD statement for data sets OPENed for input (DDNAME=*ddname* not specified).

8. DISP=OLD is set on the DD statement for data sets OPENed for output (OPENO and DDNAME=*ddname* not specified).

The following are examples of the ITPFIOX syntax:

```
DATASAVE AREA=254,TEXT=(MYMVS.FILE)
SET DC55=999
EXIT MODULE=ITPFIOX,PARM=(OPENO DC55 252 254)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR

DATASAVE AREA=254,TEXT=(RECORD 1 DATA)
EXIT MODULE=ITPFIOX,PARM=(WRITE DC55 252 254)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR

EXIT MODULE=ITPFIOX,PARM=(CLOSE DC55 252)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR

DATASAVE AREA=254,TEXT=(MYMVS.FILE)
SET DC55=999
EXIT MODULE=ITPFIOX,PARM=(OPENI DC55 252 254)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR

EXIT MODULE=ITPFIOX,PARM=(READ DC55 252 254)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR
WTO (FIRST RECORD READ = $RECALL,254$)

EXIT MODULE=ITPFIOX,PARM=(READ DC55 252 254)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=2,THEN=B-ERROR

EXIT MODULE=ITPFIOX,PARM=(CLOSE DC55 252)
IF WHEN=IMMED,LOC=DC55,COND=NE,TEXT=0,THEN=B-ERROR
```

STL Language Example

The following is an example of ITPFIOX in STL:

```
constant handle_sa#   '253'
constant data_sa#     '254'
constant rc_counter#  'DC55'

constant rc_eof        2

allocate file_handle handle_sa#
allocate file_data   data_sa#
allocate file_rc     rc_counter#

file_data = 'MYMVS.FILE'
file_rc = 999
userexit('ITPFIOX', 'OPENO' rc_counter# handle_sa# data_sa#)
if file_rc <> 0 then call error

file_data = 'Record 1'

userexit('ITPFIOX', 'WRITE' rc_counter# handle_sa# data_sa#)
if file_rc <> 0 then call error

userexit('ITPFIOX', 'CLOSE' rc_counter# handle_sa#)
if file_rc <> 0 then call error

file_data = 'MYMVS.FILE'
file_rc = 999
userexit('ITPFIOX','OPENI' rc_counter# handle_sa# data_sa#)
if file_rc <> 0 then call error

userexit('ITPFIOX','READ' rc_counter# handle_sa# data_sa#)
if file_rc <> 0 then call error
say 'Record 1 =' file_data
```

```
userexit('ITPFIOX','READ' rc_counter# handle_sa# data_sa#)
if file_rc <> rc_eof then call error

userexit('ITPFIOX','CLOSE' rc_counter# handle_sa#)
if file_rc <> 0 then call error
```

Data Compression

ITPSGEN now supports the two forms of data compression used in SNA. By default, run-length encoding (RLE) is supported in all traced RU data. When a BIND response is included in the traced RU data, Lempel-Ziv (LZ) encoding is also supported if indicated in the BIND response. To decompress both types of RU data, all RU data must be included in the traced data in order to recreate the original uncompressed RU data.

Decompression errors are reported in the summary and detail reports. The specific type of decompression error is indicated for the transmitted and received records in the detail report.

The following sections include examples of detail and summary reports, and provide possible code values for decompression errors.

Summary Report

An example of a summary report containing *terminals with errors* field is shown below:

```
                      GENERATION REPORT - SUMMARY

          TERMINALS   TRACE RECORDS     MSGTXTS       LIMIT      PATHS     TERMINALS
NETWORK   ELIGIBLE    ELIGIBLE         GENERATED     REACHED     ADDED     WITH ERRORS
VASGEN         15       10,669             0            0          0        4
```

Detail Report

An example of a detail report containing the *error codes* field is shown below:

```
                     GENERATION REPORT - SUMMARY

               TERMINALS        TRACE RECORDS        MSGTXTS          LIMIT          PATHS        TERMINALS
 NETWORK       ELIGIBLE           ELIGIBLE          GENERATED        REACHED         ADDED        WITH ERRORS
 VASGEN              15             10,669                0                0              0                4


                     GENERATION REPORT - DETAIL

                       TRACE RECORDS     MESSAGES        START            STOP
TERMINAL      NETWORK     ELIGIBLE       GENERATED       TIME             TIME          PATH         ERROR CODES
DGA7060L      VASGEN             0               0
F258TSG       VASGEN            86               0                                                   X-DCE10 R-DCE10
F294TSG       VASGEN         2,390               0                                                   X-DCE10 R-DCE10
F295TSG       VASGEN         3,554               0                                                   X-DCE10 R-DCE10
F296TSG       VASGEN         4,639               0                                                   X-DCE10 R-DCE10
IDCSLU        VASGEN             0               0
MVSAPL10      VASGEN             0               0
TP1           VASGEN             0               0
NR5E02A2      VASGEN             0               0
NR5E02A4      VASGEN             0               0
NR509A63      VASGEN             0               0
PLU           VASGEN             0               0
SLU           VASGEN             0               0
TP2           VASGEN             0               0
TP3           VASGEN             0               0
```

The detail report error codes field is blank if no decompression errors occur during script generation. If decompression errors occur, the error codes field will be formatted as follows to indicate the type of decompression error associated with transmitted and received RU data.

X-Code R-Code

The possible code values are listed below.

OK	No errors occurred decompressing the RU data.
DCE1	The three byte compression header is missing from the RU data.
DCE2	The compression header compression type field value is invalid.
DCE3	An RLE string control byte (SCB) count field has a zero value.
DCE4	An RLE SCB indicates raw data is present. However, the complete raw data is not present in the RU data.
DCE5	An RLE SCB indicates a duplicated character is present. However, the duplicated character is not present in the RU data.
DCE6	An RLE SCB type field value is invalid.
DCE7	During RLE decompression, the output buffer was overrun.
DCE8	After RLE decompression of the complete RU data, the length of the decompressed RU data did not equal the compression header length field value.
DCE9	The compression header indicates LZ compression. However, the BIND response did not indicate support of LZ compression.

DCE10 The compression header indicates LZ compression. However, no BIND response was available in the traced data to indicate the type of LZ compression supported.

DCE11 The compression header indicates LZ compression. However, only one byte of additional RU data is present.

DCE12 An LZ control sequence command field value is invalid.

DCE13 An LZ compressed RU contains one extra byte of data.

DCE14 After LZ decompression of the complete RU data, the length of the decompressed RU data did not equal the compression header length field value.

DCE15 During LZ decompression, the output buffer was overrun.

A generated script is typically usable if only the receive records cannot be decompressed (R-DCEnn). This is because the messages created by a TPNS script are generated from the transmitted RU data records in the sorted trace input file.

New DSPLY Loglist Control Command Operand

A new operand, LOG, has been added to the DSPLY data type selection command in the loglist utility. The LOG operand causes only the log display records created using the 'LOG DISPLAY' scripting statements to be printed.

Loglist Data Output and Display

The TPNS loglist data output for Simple TCP and Simple UDP is formatted as ASCII and EBCDIC data with 24 bytes of hex data with ASCII and EBCDIC interpretations. The X.25 TWX PAD packets will be interpreted as ASCII only. ASCII data will be framed with '<' and '>' characters. Below is an example of the loglist output:

```
00000000   53616D70 .. 726F6D20   <Sample Transaction from >   *./_.%..../>./...?>...?_.*
00000018   53544350 .. 78787878   <STCP001 ..xxxxxxxxxxxxxx>   *...&....................*
00000030   78787878 .. 78787878   <xxxxxxxxxxxxxxxxxxxxxxxx>   *........................*
        TO NEXT LINE SAME AS ABOVE
00000060   78787878 .. 6D706C65   <xxxxxxxxxxxxxxx..@Sample>   *................. ./_.%.*
00000078   20547261 .. 0A0D0A     < Transaction line 2.... >   *.../>./...?>.%.>....... *
```

The TPNS Display Monitor Facility has been updated to allow the monitored data to be displayed in either EBCDIC or ASCII. The M operator command will now accept CODE=EBCDIC|ASCII when the monitoring is started. EBCDIC is the default. The example below shows the updated TPNS Display Monitor Facility logon panel:

```
             TPNS Version 3 Release 5.0.1 Display Monitor Facility

Name   =                         TPNS name of simulated device or 3270 display
View   = SCREEN         DATA or SCREEN - show data stream or 3270 screen image

Screen image display only:
Update = XMITRECV       Monitoring display updated when:
                          MONITOR  - MONITOR statement is executed from script,
                          TIMER    - the specified time value expires, or
                          XMITRECV - data is transmitted/received by display.
Source = BLOCKS         Data stream sent to the monitoring display built from:
                          BLOCKS - TPNS internal control blocks
                          DATA   - data transmitted/received by display.
Timer  = 10             1-600 Seconds when Update = TIMER
Aid    = ON             ON, OFF, or (row,column) location of AID display field

Data stream display only:
Lines  = 2              Maximum number of displayed data lines
Code   = EBCDIC         ASCII or EBCDIC - interpret data as ASCII or EBCDIC


ENTER    - Submits parameters to start monitoring of simulated display.
PA1/ATTN - Stops monitoring of simulated display.
PF3/PF15 - Ends Display Monitor Facility session.
```

ITPECHO Generic Resource Support

VTAM Generic Resource support has been added to ITPECHO. The GNAME=generic_resource_name execution parameter allows ITPECHO to associate itself with the generic resource name specified. The following default prompt is displayed by ITPECHO.

```
ITP900I ITPECHO: APPLID=ITPECHO,BUFSIZE=2048,GNAME=,PASSWD=,SMSG,NOTRACE,WTOR
```

New ITPVTBRF Execution Parameters

The following execution parameters have been added to ITPVTBRF:

INVERT
INVERT allows scripts to be generated in situations where the VTAM buffer trace was active for an LU that did not capture ″inbound″ data to VTAM. When INVERT is specified, ″outbound″ data from VTAM will be reformatted to allow for script generation by ITPSGEN.

SSCPNAME=*sscp_name*
SSCPNAME allows an SSCP name other than the default (″VTAM″) to be specified.

ITPSGEN Initial Delay Calculation

When the initial time record is present as the first record of the trace input to ITPSGEN, the time stamp in that record will provide a default ″start time″ for calculating initial delays in generated scripts where delays have been requested. The initial time record is a special record written to the reformatted trace data set by the ITPVTBRF utility. This record contains the earliest time stamp from the reformatted trace data set and a dummy name of all hex 00's so that it will always sort to the beginning of the input to ITPSGEN. The provision of this record was part of the updates to ITPVTBRF for Service Level 9711.

The default ″start time″ established through the new initial delay calculation function is used if all of the following conditions are true:

- No start time has been specified via a TIME control statement.
- A DELAY control statement has been specified.
- Either ISTART is specified on the DELAY control statement or no other start time (ACTLU, BIND, SDT) is available for the script being generated.

Except for obtaining the start time from a different source, the actual initial delay is calculated in the same manner as it was prior to Service Level 9711. Refer to the *TPNS Script Generating Utilities* manual for a description of the existing DELAY control statement.

Storage Allocation Below 16 Megabytes Minimized

For VTAMAPPL and CPI-C simulations, the amount of storage allocated below the 16 megabyte line has been lowered by allocating only the ACB, APPLID, and PASSWD areas below the 16 megabyte line. This change allows additional VTAMAPPL and/or CPI-C APPCLU resources to be simulated within a TPNS region.

Boundary Channel-Attached Type 2.1 Node CP to CP Capability

Boundary channel-attached Type 2.1 nodes now provide CP to CP session capability.

VM/ESA REAL I/O Support Without RIO370 Pages

TPNS now supports Real I/O on VM/ESA without RIO370 pages. To support Real I/O when running on VM/ESA, TPNS required the system to be generated with RIO370 pages. This was a problem for many users since the space available below 16MB is limited. The need for RIO370 pages comes from the way the channel programs are built and the ending status is interpreted. This enhancement allows TPNS users running on VM to use Real I/O without reserving pages for RIO370.

TPNS/ISPF Interface PF3 Key Changes

PF3 in the TPNS/ISPF Interface now takes you to the previous panel, unless pressed on the main panel, in which case it exits. Prior to Service Level 9711, PF3 would exit the TPNS/ISPF Interface regardless of where it was pressed.

Service Level Indicator

A TPNS service level indicator has been added by appending a digit after the modification level. For example, 3.5.0.2 will be used for the second refresh (Service Level 0110). For the first refresh (Service Level 9711) 3.5.0.1 was used. This change has been made in the following areas:

Interval Report:

```
  TPNS 3.5.0.2 PRINTER OUTPUT
```

Display Monitor:

```
  TPNS Version 3 Release 5.0.2 Display Monitor Facility
```

ITPIDC:

```
  ITP1501I TPNS VERSION 3, RELEASE 5.0.2, INTERACTIVE DATA CAPTURE UTILITY

  TPNS Version 3 Release 5.0.2  Program Number 5688-121
```

ITPLL:

```
  RUN  TIME 15.05.23,   JANUARY 30, 2001  VERSION  3  RELEASE  5.0.2

  15052337 0099030  35020000  CNSL
```

ITPRESP:

```
  RUN  TIME 15.05.23,   JANUARY 30, 2001  VERSION  3  RELEASE  5.0.2
```

ITPCOMP:

```
  MASTER RUN  TIME 11.09.48,   JANUARY 29, 1997  VERSION  3  RELEASE  5.0.0
  TEST   RUN  TIME 10.45.39,   JANUARY 30, 2001  VERSION  3  RELEASE  5.0.2
```

ITP0BRW2

The BROWSE command in the ITP0BRW2 exec of the TPNS/ISPF Interface has been changed to VIEW. This enables the VIEW mode in ISPF where the edit functions, including scrolling, overtyping data, entering line commands and primary commands to modify the data can be used. If you have a level of ISPF earlier than ISPF 4.2 or prefer BROWSE over VIEW, change line 56 of the ITP0BRW2 exec to “BROWSE DATASET(″dsname″)”.

NOWTOR TPNS Execution Parameter

The NOWTOR TPNS execution parameter has been added to allow TPNS to be executed under MVS as a batch job without issuing a WTOR for operator control. When NOWTOR is specified, no WTOR will be issued and TPNS must either be ZENDed via an operator command from an active TPNS network or canceled by the operator.

Chapter 6. Migration Considerations

This chapter describes issues you should consider when migrating to the TPNS Version 3 Release 5 function and service enhancements that are being made available in TPNS Service Levels 9711 and 0110.

Loglist Utility ITPLL and CPI-C Trace Records

The loglist utility program ITPLL shipped with the TPNS Version 3 Release 5 function and service enhancements *must* be used to format the CPI-C trace records created using Service Level 9711 or 0110. The loglist utility program shipped with the original program product release of TPNS Version 3 Release 5 will loop processing the CPI-C trace records created by Service Level 9711 or 0110.

The loglist utility program shipped with Service Level 9711 or 0110 will format the CPI-C trace records created by these service levels of TPNS and those created by the original program product release of TPNS Version 3 Release 5 without any problems.

Loglist Utility ITPLL and Verify Records

The loglist utility program ITPLL shipped with Service Level 9711 or 0110 *must* be used to format verify records created by these service levels of TPNS. The loglist utility program shipped with the original program product release of TPNS Version 3 Release 5 will not format the verify records created by Service Level 9711 or 0110 correctly. The loglist utility program shipped with Service Level 9711 or 0110 will format the verify records created by these service levels of TPNS and those created by the original program product release of TPNS Version 3 Release 5 without any problems.

STL

The following are new STL reserved words that cannot be used as variable or constant names in STL programs:

- BITAND
- BITOR
- BITXOR
- B2X
- CENTER
- COPIES
- DELWORD
- D2C
- LASTPOS
- OVERLAY
- PATHID
- POS
- PULL
- PUSH
- QUEUE
- QUEUED

- REVERSE
- SPACE
- STRIP
- SUBWORD
- WORD
- WORDINDEX
- WORDPOS
- WORDS
- X2B
- X2C

User Exits

Counter Reference

The counter control block structure has not changed. When index counters above number 63 must be referenced by a user exit, the new exit interface routine requests (refer to “New Exit Interface Routine Requests” on page 84), should be used to determine the number of index counters allocated at the desired resource level before referencing and/or changing the counter value.

Switch Reference

User exits which reference switches using the exit interface routine will not require any changes to reference switch numbers up to 32. When switch numbers above 32 must be referenced, the new exit interface routine requests (refer to “New Exit Interface Routine Requests” on page 84), should be used to determine the number of switches allocated at the desired resource level before referencing and/or changing the switch setting.

User exits which reference switches using the TPNS internal control block structure will require changes when more than the default 32 switches are allocated at the desired resource level. These exits should be recoded to use the documented interface to locate switches through the user exit interface routine.

Save Area Reference

User exits which reference save areas using the exit interface routine will not require any changes to reference save area numbers up to 255. When save area numbers above 255 must be referenced, the new exit interface routine requests (refer to “New Exit Interface Routine Requests” on page 84), must be used to reference the additional save areas. The SAV DSECT should be used to reference other fields in the save area control block since offsets into the control block have changed.

Save Area Control Block

The save area control block has been updated to support save area numbers above 255. The SAVNUM field has been changed from a one-byte to a two-byte field and the SAVBUFSZ, SAVDATLN, SAVFLAGS, and SAVDATAX fields now have different offsets into the control block. The SAV DSECT has been updated to reflect this change.

Log Record Header

The LOGLEVEL field has been modified to include a service level indicator. This two-byte field now contains two one-byte components named LOGVEREL and LOGMODRF. For Service Level 9711 of TPNS Version 3 Release 5, LOGLEVEL will contain X'3501'. For Service Level 0110 of TPNS Version 3 Release 5, LOGLEVEL will contain X'3502'. The LOG DSECT has been updated to reflect this change.

TCP/IP Subsystem Detection

TPNS will not detect the TCP/IP instance, as specified by the TCPIP operand, being inactive until the TPNS network is started. Previously, this detection was performed when the TPNS network was initialized.

TPNS/ISPF Interface PF3 Key Changes

PF3 in the TPNS/ISPF Interface now takes you to the previous panel, unless pressed on the main panel, in which case it exits. Prior to Service Level 9711, PF3 would exit the TPNS/ISPF Interface regardless of where it was pressed.

TPNS/ISPF Interface ITP0BRW2 Changes

If you have a level of ISPF earlier than ISPF 4.2 or prefer BROWSE over VIEW, change line 56 of the ITP0BRW2 exec to “BROWSE DATASET (″dsname″)”.

Appendix A. Work Station Trace Reformatter Utility

ITPWSTRF is a REXX exec that reformats OS/2 Communications Manager (CM/2) and IBM Communications Server LU 6.2 traces into TIR formatted records for processing by the TPNS Script Generator (ITPSGEN). The steps for using OS/2 Communications Manager (CM/2) and IBM Communications Server traces as script generation source files are as follows:

1. Capture an LU 6.2 trace using either the OS/2 Communications Manager (CM/2) trace facility or the IBM Communications Server trace facility.
2. Upload the trace output file to your host system as an EBCDIC TEXT file.
3. Run the Work Station Trace Reformatter utility (ITPWSTRF) against the uploaded trace file.
4. Sort the ITPWSTRF output file in ascending order by resource name, session number, date, and time fields.
5. Run the Script Generator utility (ITPSGEN) against the sorted file.

Trace Output Format Requirements

ITPWSTRF can only be used to reformat trace output produced by the OS/2 Communications Manager (CM/2) or IBM Communications Server trace facilities. Examples of the format required for the trace output files are provided below.

Note: ITPWSTRF will only process trace records in the formats listed below. The trace record formats are subject to change as the OS/2 Communications Manager (CM/2) and IBM Communications Server products evolve over time.

If the Work Station trace does not match one of the following formats, such that ITPWSTRF cannot be used, you should use a VTAM buffer trace and ITPVTBRF to provide input for CPI-C script generation.

Example: CM/2 Trace Record

```
"TRACE COPIED 06/30/2001 13:09:05.06""
<==SEND=====  IBMTRNET  #00 40003745100204  0C400774E27E37BF          13:08:08.16
      #:009F TH:2F0001020001 RH:6B8100
    31001307 B0B05033 01808686 80010602      <1.....P3..ff....>
    00000000 0000001C 23000010 E4E2C9C2      <........#...USIB>

===RECV====>  IBMTRNET  #00 40003745100204  0C400774E27E37BF          13:08:10.44
      #:0068 TH:2F0002010001 RH:EB8000
    31001307 B0B050B3 00808585 80000602      <1.....P...ee....>
    00000000 00000010 23000000 25000902       <........#...%...>
```

Example: CS Trace Record

This section includes two examples of valid IBM Communications Server Trace Records.

Token Ring Trace Format

```
Á22‖ 12/16 10:59:49.45,(0015) len=195, Connectivity.LAN (LLC2).0001,  00000000:0
Frame Type:           TOKEN_RING
Source Address:       0004ac751777 (canonical: 002035aee8ee)
Source SAP:           04
Destination Address:  0004ac325f7b (canonical: 0020354cfade)
Destination SAP:      04
LPDU Type:            I-Frame
Command Flag:         C              Poll Flag:            0
NR:                   00
NS:                   00
LPDU Data length:     179
  2D000002 80016B81 00310013 07B0B050  <-.....k..1.....P>  <......,a.......&>
  33018186 86810106 02000000 00000000  <3...............>  <..affa..........>
  16230000 0DD4C1D9 D2D5C5E3 4BD4C1D9  <.#..........K...>  <.....MARKNET.MAR>
```

SDLC Trace Format

```
  Á22‖ 12/16 10:59:49.45,(0015) len=195, Connectivity.SDLC.0704, 00000000:0000000
  SDLC TRANSMIT I-FIELD
  2D000002 80016B81 00310013 07B0B050 <-.....K..1.....P>  <......,a.......&>
  33018186 86810106 02000000 00000000 <3...............>  <..affa..........>

  Á23‖ 12/16 10:59:49.45,(0016) len=150, Connectivity.SDLC.0704, 00000000:0000000
  SDLC TRANSMIT I-FIELD
  2D000200 8001EB80 00310013 07B0B050 <-........1.....P>  <...............&>
  33008086 86800006 02000000 00000000 <3...............>  <...ff...........>
```

Using ITPWSTRF

An explanation of the commands to execute ITPWSTRF under VM CMS and MVS are provided below.

Executing ITPWSTRF Under VM CMS

To execute ITPWSTRF under VM CMS, specify the following:

ITPWSTRF *ifn ift ifm ofn oft ofm* **([***INVERT***]**

Where:

- *ifn* is the input file name
- *ift* is the input file type
- *ifm* is the input file mode
- *ofn* is the output file name
- *oft* is the output file type
- *ofm* is the output file mode
- *INVERT* causes the reformatter to switch the origin and destination LU names

Notes:

1. The *ifn*, *ift*, and *ifm* parameters are required.
2. The input names may be duplicated for *ofn*, *oft*, and/or *ofm* by using an equals sign (=) in place of the parameter name.

3. The defaults for *ofn* and *ofm* are the *ifn* and *ifm* values respectively; the default for *oft* is WSTRF.
4. The input and output files must be different physical files.

Executing ITPWSTRF Under MVS

To execute ITPWSTRF under MVS, specify the following:

```
ITPWSTRF input_file  output_file [INVERT]
```

Where:

- *input_file* is the input file name
- *output_file* is the output file name
- *INVERT* causes the reformatter to switch the origin and destination LU names

Notes:

1. The *input_file* and *output_file* parameters are required.
2. The input and output files must be different physical files.
3. If the file name is not specified in quotes, the TSO userid is added as the high-level qualifier.

Appendix B. Messages and Return Codes

The TPNS Version 3 Release 5 function and service enhancements changed a few existing messages and added a few new messages and return codes. The new and revised messages and codes are explained below. Note that Initiator Message ITP1385I has been eliminated. Refer to *TPNS Messages and Codes* for complete information on the existing TPNS messages.

TPNS Script Generator Utilities Messages

The following are TPNS Script Generator Utilities messages added or changed by the TPNS Version 3 Release 5 function and service enhancements.

ITP863I I/O ERROR ON INPUT TAPE - EXECUTION ABORTED

Explanation: This message indicates that an I/O error occurred while reading the input data set.

System Action: ITPSGEN execution stops.

User Response: Check the definition of the input data set. If the data set is a tape file, the problem could be with the tape drive. Try using another drive.

ITP864I I/O ERROR ON DATA SET *ddname* — EXECUTION ABORTED

Explanation: This message indicates that an I/O error occurred while reading or writing the data set named.

System Action: ITPSGEN execution stops.

User Response: Check the definition of the data set.

ITP866I NO TRACE RECORDS WERE FOUND THAT ARE ELIGIBLE FOR SCRIPT GENERATION

Explanation: No trace records were found in which the resource name matched a TERM, DEV, LU, or APPCLU name specified in the model network(s).

System Action: No scripts are generated.

User Response: Make sure the correct model network and input trace data set are being referenced. Make sure at least one device in the model network matches at least one resource found in the trace data set.

ITP867I USER CONTROL DATA ENCOUNTERED FOR TP XXXXXXXX - THE DATA WAS IGNORED

Explanation: The trace data set contains data for the specified TP that is encoded as user control data. Since the NOUCD script generation control command was specified, this data is ignored by the script generator.

System Action: The user control data is ignored and script generation continues.

User Response: If the user control data should be processed as application data, specify the UCD script generation control command and rerun the script generator.

ITP868I USER CONTROL DATA ENCOUNTERED FOR TP XXXXXXXX - THE DATA WAS PROCESSED AS APPLICATION DATA

Explanation: The trace data set contains data for the specified TP that is encoded as user control data. Since the UCD script generation control command was specified or defaulted, this data is processed as application data by the script generator.

System Action: The user control data is processed as application data and script generation continues.

User Response: If the user control data should not be processed as application data, specify the NOUCD script generation control command and rerun the script generator.

ITP869I THE SCRIPT FOR TP XXXXXXXX WAS GENERATED FROM A FULL-DUPLEX SESSION — USE THIS SCRIPT WITH CAUTION

Explanation: The session that was used to generate the specified TP was identified as a full-duplex session.

System Action: Script generation continues. A script is generated for the specified TP.

User Response: When running TPNS simulations that use scripts generated from full-duplex sessions, be aware that the simulations may not accurately reproduce the original traced scenario. However, it is possible that the session was identified as full-duplex but used as if it were half-duplex flip-flop. In this case, the generated scripts should accurately reproduce the original traced scenario.

ITP870I TRUNCATED TRACE DATA ENCOUNTERED FOR TP XXXXXXXX

Explanation: This message indicates that truncated data has been detected in the trace data set for the specified TP.

System Action: Script generation continues. A script is generated for the specified TP.

User Response: Use generated scripts with caution as they might not reflect complete data flows. Make sure you request a full buffer trace when capturing trace files. For VTAM buffer traces, a full buffer trace is requested by specifying the AMOUNT=FULL parameter on the trace command.

ITP871I ERRONEOUS TRACE DATA DETECTED: END-OF-CHAIN RECEIVED BEFORE EXPECTED

Explanation: This message indicates that an end-of-chain indicator was received before all of the chain was received.

System Action: Script generation continues.

User Response: Use generated scripts with caution as they might not reflect complete data flows. The trace file contains corrupted data. Consider recapturing the trace data.

ITP872I ERRONEOUS TRACE DATA DETECTED:END-OF-CHAIN NOT RECEIVED WHEN EXPECTED

Explanation: This message indicates that an end-of-chain indicator was not received when all of the data in the chain was received.

System Action: Script generation continues.

User Response: Use generated scripts with caution as they might not reflect complete data flows. The trace file contains corrupted data. Consider recapturing the trace data.

TPNS STL Translator Messages

The following are TPNS STL messages added by the TPNS Version 3 Release 5 function and service enhancements.

ITP3202I INVALID OPTION ARGUMENT FOR ″STRIP″ FUNCTION

Explanation: This message indicates that an invalid option was specified.

System Action: The current statement is ignored and processing continues.

User Response: The valid options are B, L, and T.

ITP3203I SEQUENCE NUMBER VALUE TOO LARGE

Explanation: This message indicates that the sequence number value specified was too large.

System Action: The current statement is ignored and processing continues.

User Response: Specify a sequence number value between 0 and 65535.

ITP3204I SEQUENCE ACTION VALUE INVALID

Explanation: This message indicates that an invalid action was specified.

System Action: The current statement is ignored and processing continues.

User Response: The valid action values are IGNORE, SET, TEST, or TESTSET.

ITP3205I SAY STATEMENT TYPE OPTION INVALID

Explanation: This message indicates that the TYPE option specified on the SAY statement is invalid.

System Action: The current statement is ignored and processing continues.

User Response: The valid TYPE option is ABRHD.

ITP3206I IMPLICIT VARIABLE DEFINITION DISALLOWED BY NOIMPLICIT OPTION

Explanation: This message indicates that the NOIMPLICIT option is specified and implicit definitions variables are not allowed.

System Action: The current statement and the remainder of the current line is ignored. Processing continues with the next input line.

User Response: Ensure that the flagged variable definition is specified correctly. If correctly specified, then declare it using a BIT, INTEGER, STRING, or ALLOCATE statement.

Informational Log Data Set Messages

The following Informational Log Data Set messages were changed by the TPNS Version 3 Release 5 function and service enhancements.

ITP427I *logic test (dname,snum)* **MET - THEN ACTION TAKEN:** *action*

Explanation: This message trace record indicates that the specified logic test was evaluated, the specified test condition was met, and the action specified by the THEN operand was taken. The specified logic test is found in message generation deck *dname* at statement *snum*. The *action* contains information about the action taken. It can be one of the following:

- BRANCH FROM *snum* OF *dname* TO {*label* AT *snum*|THE BEGINNING} OF *dname*
- CALL FROM *snum* OF *dname* TO {*label* AT *snum*|THE BEGINNING} OF *dname*
- RETURN FROM *snum* OF *dname* TO THE STMT AT *snum* OF *dname*
- EXECUTED *dname* FROM {THE BEGINNING|*label* AT *snum*}
- ABORTED *dname* AT STMT# *snum* AND STARTED *dname*
- CONTINUE (RESET WAIT)
- WAIT INDICATOR SET
- QUIESCE DEVICE
- RELEASE DEVICE
- WAITING ON EVENT
- IGNORE
- SET DEV SWITCH *XX* ON
- SET DEV SWITCH *XX* OFF
- SET DEV SWITCHES ON
- SET DEV SWITCHES OFF
- SET TERM SWITCH *XX* ON
- SET TERM SWITCH *XX* OFF
- SET TERM SWITCHES ON
- SET TERM SWITCHES OFF
- SET NTWRK SWITCH *XX* ON
- SET NTWRK SWITCH *XX* OFF
- SET NTWRK SWITCHES ON
- SET NTWRK SWITCHES OFF
- EVENT *ename* POSTED
- EVENT *ename* QSIGNALED
- EVENT *ename* SIGNALED
- EVENT *ename* RESET
- MAX CALL LEVEL EXCEEDED
- INVALID RETURN ISSUED
- VERIFY
- ABORTED *dname* AT STMT# *snum* AND TERMINATED THE TP
- OUTSTANDING DELAY CANCELED

System Action: The THEN action is taken.

User Response: None.

ITP477I **socket_call_type SOCKET CALL FAILED WITH ERRNO nnnn**

Explanation: A socket call used to control the connection completed with a return code of -1. The ERRNO value can be found in the TCPERRNO H header file. Typically, this indicates network problems.

System Action: TPNS issues this message, queues a timer delay, and continues operation.

User Response: If the problem continues, stop the TCP/IP connection.

ITP478I **TCPERRNO nnnn — CANNOT CONNECT TO xxx.xxx.xxx.xxx PORT nnnnn**

Explanation: The CONNECT to the destination xxx.xxx.xxx.xxx PORT nnnnn failed. The ERRNO value is defined in the C language TCPERRNO header file or in the IBM TCP/IP Product Publications. This indicates that the destination address and port number are unavailable or unreachable.

System Action: TPNS issues this message, queues a timer delay, and continues operation.

User Response: Ensure that the destination IP address and port number are correct and alter the SERVADDR and/or PORT values as needed. In MVS, the TCPERRNO header file is a member of the C header files and Pascal include files data set. This data set is called *hlq*.SEZACMAC, where *hlq* is the installation-defined high-level qualifier. In VM, the TCPERRNO header file is a separate file named TCPERRNO H, installed with the TCP/IP product.

Changed Return Codes

The following are Script Generator Utility return codes changed by the TPNS Version 3 Release 5 function and service enhancements:

72	An error occurred while reading from an input data set
76	The STLTXT data set failed to open

80 The NTWRK data set failed to open.

Appendix C. Simple TCP Sample Script

The following Simple TCP script provides Telnet Line Mode NVT negotiations exchange. Simple TCP provides an alternative means of simulating the various Telnet protocols. Simple TCP gives the user more control in the negotiations but requires more TPNS scripting.

Simple TCP Client Connecting to a Server Using Telnet Line Mode Network Virtual Terminal

```
@NET
************************************************************************
* Network Configuration:  Simple TCP Client simulation                 *
*                                                                      *
* Description:  This TPNS script will simulate one Simple TCP Client   *
*               connecting to a server, issuing a request to that      *
*               server, receiving data until the server closes the     *
*               connection, and then repeating the process.            *
*                                                                      *
*               The server to which this Simple TCP Client connects    *
*               is assumed to have the following characteristics:      *
*                1) requests to it must use ASCII code;                *
*                2) the end of a request is marked by the              *
*                   carriage return/line feed (CR/LF) sequence;        *
*                3) the server closes the connection when all response *
*                   data has been sent.                                *
*                                                                      *
*               Some values may need to be changed in this data set in *
*               order to operate in your environment.  They are        *
*               indicated by the "<== " string.                        *
*                                                                      *
* TPNS CP Gen:  None                                                   *
* NCP Gen:      None                                                   *
*                                                                      *
* Publications Cross Reference:                                        *
*  1) Defining TPNS Networks       - Information on this particular   *
*                                    network configuration             *
*  2) Using STL and the STL        - Help on writing STL scripts      *
*     Translator                                                       *
*  3) TPNS Language Reference      - Details on how to code TPNS      *
*                                    statements                        *
************************************************************************

*----------------------------------------------------------------------*
* Network statement operands.                                          *
*----------------------------------------------------------------------*
STCPLNMD NTWRK HEAD='Telnet Line Mode NVT', * Set the title line
               CONRATE=YES,         * Print message rates on console
               OPTIONS=(MONCMND,debug),   * Network Options
               ITIME=1,             * Interval report every 1 minute
               BUFSIZE=32000,       * Specify buffer size
               THKTIME=UNLOCK,      * Wait for keyboard unlock
               UTI=100,             * User time interval is 1 second
               MSGTRACE=YES,        * Trace messages
               STLTRACE=YES,        * Trace messages
               TCPNAME=TCPIP        * <== Default name of the local
*                                   *     TCPIP virtual machine
*----------------------------------------------------------------------*
STCPDECK PATH  STCPDECK
*----------------------------------------------------------------------*
* Define the network resources.                                        *
*                                                                      *
* This is a TCP/IP connection with 1 simulated device.  You may        *
* add additional operands on the device if desired.  See the TPNS      *
* Language Reference manual for details on valid operands.             *
*----------------------------------------------------------------------*
TCONN1   TCPIP
DEV010   DEV  TYPE=STCP,            * Simple TCP Client
              PORT=23,              * Server Port for connection
              SERVADDR=9.67.127.216,* Server IP Address for connect
              PATH=(STCPDECK)       * Path Sequence for this DEV
@ENDNET

@program=stcpprog
integer  shared nextnum
integer  nextid
integer  startpos
constant crlf '0D0A'x
constant ff 'FF'x

stcpdeck:  msgtxt
/*---------------------------------------------------------------------*
* The Message Generation deck.                                         *
*                                                                      *
* Generates requests for the server hypothesized in the network        *
*          description above, waits for the connection to be          *
*          closed, and then generates another request.                 *
*---------------------------------------------------------------------*/
/*  Initialize table for translation to EBCDIC                        */
  asc2ebc = '00010203372D2E2F1605250B0C0D0E0F'X||, /* 00-0F */
            '101112133C3D322618193F27221D351F'X||, /* 10-1F */
            '405A7F7B5B6C507D4D5D5C4E6B604B61'X||, /* 20-2F */
            'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'X||, /* 30-3F */
            '7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'X||, /* 40-4F */
            'D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D'X||, /* 50-5F */
            '79818283848586878889919293949596'X||, /* 60-6F */
            '979899A2A3A4A5A6A7A8A9C04FD0A107'X||, /* 70-7F */
            '00010203372D2E2F1605250B0C0D0E0F'X||, /* 80-8F */
            '101112133C3D322618193F27221D351F'X||, /* 90-9F */
            '405A7F7B5B6C507D4D5D5C4E6B604B61'X||, /* A0-AF */
            'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'X||, /* B0-BF */
            '7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'X||, /* C0-CF */
            'D7D8D9E2E3E4E5E6E7E8E9ADE0BD5F6D'X||, /* D0-DF */
            '79818283848586878889919293949596'X||, /* E0-EF */
            '979899A2A3A4A5A6A7A8A9C04FD0A107';   /* F0-FF */
/*  Initialize table for translation to ASCII             */
  ebc2asc = '000102031A091A7F1A1A1A0B0C0D0E0F'X||, /* 00-0F */
            '101112131A1A081A18191A1A1C1D1E1F'X||, /* 10-1F */
            '1A1A1C1A1A0A171B1A1A1A1A1A050607'X||, /* 20-2F */
            '1A1A161A1A1E1A041A1A1A1A14151A1A'X||, /* 30-3F */
            '20A6E180EB909FE2AB8B9B2E3C282B7C'X||, /* 40-4F */
            '26A9AA9CDBA599E3A89E21242A293B5E'X||, /* 50-5F */
            '2D2FDFDC9ADDDE989DACBA2C255F3E3F'X||, /* 60-6F */
            'D78894B0B1B2FCD6FB603A2340273D22'X||, /* 70-7F */
            'F861626364656667686996A4F3AFAEC5'X||, /* 80-8F */
            '8C6A6B6C6D6E6F7071729787CE93F1FE'X||, /* 90-9F */
            'C87E737475767778797AEFC0DA5BF2F9'X||, /* A0-AF */
            'B5B6FDB7B8B9E6BBBCBD8DD9BF5DD8C4'X||, /* B0-BF */
            '7B414243444546474849CBCABEE8ECED'X||, /* C0-CF */
            '7D4A4B4C4D4E4F505152A1ADF5F4A38F'X||, /* D0-DF */
            '5CE7535455565758595AA0858EE9E4D1'X||, /* E0-EF */
            '30313233343536373839B3F7F0FAA7FF'X;  /* F0-FF */
   /* clear data received each time data is transmitted */
 onout then
  data_in=''
   /* if no data is received, the connection is closed */
 onin buffer='' then abort
   /* store data received until data transmitted */
 onin then
  data_in=data_in||buffer
```

```
type '0D0A'x

/* this script does not account for 'FF'x at the end of data    */
/* look at each received stream of data and transmit a response */
/* WILL and DO will be responded back with WONT and DONT        */
/* the client will look like a Network Virtual Terminal         */
/* the script assumes that there will be a prompt to check      */
/* for to indicate the when the client can send data            */

 /* set look_for to what you expect to receive */
look_for='login'
call wait4it

say devid() ' received login'
nextid =nextnum                   /* index for user tables */
nextnum=nextnum+1
if nextnum=utblmax(ids) then
 nextnum=0
type translate(utbl(ids,nextid),ebc2asc)||crlf

 /* change 'Password' to what you expect to receive */
look_for='Password'
call wait4it

say devid() ' received Password'
type translate((utbl(pws,nextid)),ebc2asc)||crlf

do nextcmd=0 to utblmax(cmds)

     /* change '$' to what prompt you expect to receive */
 look_for='$'

 call wait4it
 say devid() ' sending COMMAND' utbl(cmds,nextcmd)
 type translate((utbl(cmds,nextcmd)),ebc2asc)||crlf
end
suspend()
quiesce
endtxt


wait4it: msgtxt

 /* wait for specific data deck -                             */
 /* set look_for to the data expected                         */
 /* check for FF in the data stream in case more negotiations */
 /* need to take place                                        */

look_for = translate(look_for,ebc2asc)
notfound = on
do while notfound=on                /* wait until data found   */
 delay(0)
 transmit and wait until onin
 if index(data_in,ff) > 0 then      /* look for commands first */
  call negotiat                     /* negotiate commands      */
 else
  if index(data_in,look_for) > 0 then  /* look for data next    */
   notfound = off
end
endtxt


negotiat: msgtxt

 /* negotiation deck -                                         */
```

```
  /* parse through data for FF, look at next bytes for the      */
  /* commands and options                                       */
  /* looks for DO, DONT, WILL, WONT and DATA MARK commands      */
  /* looks for the Suppress Go Ahead option                     */

 data_out=''                         /* clear output buffer      */
 data_ck=''                          /* clear data parsing field */
 datalen=length(data_in)             /* get data length          */
 if datalen>1 then                   /* >1 implies not just FF   */
  do
   startpos=index(data_in,ff)        /* find FF in data          */
        /* start at command past ff */
   data_ck=substr(data_in,startpos+1,length(data_in)-startpos)
   fffound=on
   do while fffound=on
        /* get first byte of parsed data */
    data_byte=substr(data_ck,1,1)
        /* get second byte of parsed data */
    data_byte2=substr(data_ck,2,1)
    if data_byte='FD'x | data_byte='FE'x then  /* DO or DONT    */
     if data_byte2='03'x then              /* Suppress GO Ahead */
      data_out=data_out||'FFFB'x||data_byte2  /* WILL */
     else                                  /* any other option  */
      data_out=data_out||'FFFC'x||data_byte2  /* WONT */
    else
     if data_byte='FC'x | data_byte='FB'x then /* WILL or WONT  */
      if data_byte2='03'x then             /* Suppress GO Ahead */
       data_out=data_out||'FFFD'x||data_byte2 /* DO */
      else                                 /* any other option  */
       data_out=data_out||'FFFE'x||data_byte2 /* DONT */
     else
      if data_byte='F2'x then            /* data mark, synch    */
       do                                /* assume synch signal */
        data_in=buffer                   /* clear previous data */
        data_out=''                      /* no response sent    */
       end
    startpos=index(data_ck,ff)           /* look for ff         */
    if startpos=0 then
     do  /* no ff */
      fffound=off
      startpos=index(data_in,look_for)   /* search for specific data */
      if startpos>0 then                 /* found specific data      */
       notfound=off                      /* get out of loop          */
     end   /* no ff */
    else
       /* found ff, parse through commands */
     data_ck=substr(data_ck,startpos+1,length(data_ck)-startpos)
   end  /* fffound on */
   if data_out¬='' then                  /* check if data to transmit */
    type data_out                        /* transmit data             */
  end
 endtxt

ids: msgutbl
 'userid1'
 'userid2'
 'userid3'
 'userid4'
 'userid5'
endutbl

pws: msgutbl
 'pswd1'
 'pswd2'
 'pswd3'
 'pswd4'
 'pswd5'
```

```
endutbl

cmds: msgutbl
 'cd /usr/lpp'
 'ls'
 'logout'
endutbl
```

Appendix D. Miscellaneous

This appendix provides information about the miscellaneous enhancements made to TPNS Version 3 Release 5.

TPNS Sample Network Models

When using the TPNS Version 3 Release 5 Service Level 9711 CPIC or TN3270 network models which are part of the TPNS/ISPF Interface, you must make the following changes to the model definitions after they are selected.

CPIC

- Change the line after ’@PROGRAM=CPIC’ that starts with ’*----’ to ’/*---’.
- Change the end of the fifth line after the @PROGRAM line from ’----*’ to ’---*/’.

TN3270

Change ’HEAD=Telnet 3270 and 3270E Model Network’ to ’HEAD=Telnet 3270 Network’.

TPNS Sample Data Set

Three new members have been added to the TPNS Sample data set.

WEBLOAD	TPNS network definition scripts to simulate Web clients.
TAPING	TPNS network definition and scripts that define a CPI-C client application that attaches the APINGD server.
TAPINGD	TPNS network definition and scripts that define a CPI-C server application that can be attached by the APING client.

Appendix E. Notices

conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, photographs and color illustrations may not appear.

Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

IBM

VTAM

SNA

IBM Communications Server OS/2

Communications Manager (CM/2)

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Bibliography

TPNS Library

TPNS General Information, GH20–2487

TPNS Primer, SC31–6043

TPNS Planning and Installation, SH20–2488

Defining TPNS Networks, SC31–6008

Creating TPNS Message Generation Decks, SC31–6009

TPNS Language Reference, SH20–2489

TPNS Script Generating Utilities, SC30–3453

TPNS Samples, SC30–3454

TPNS STL Reference Card, SX75–0065

Using the TPNS Structured Translator Language (STL) and the STL Translator, SC31–6013

TPNS Operation, SC30–3289

TPNS Messages and Codes, SC30–3310

TPNS General Utilities, SC30–3290

TPNS User Exits, SC31–6009

TPNS Test Manager User's Guide and Reference, SC31–8719

TPNS Master Index, GC31–6059

TPNS Library (all manuals except TPNS General Information, and TPNS Test Manager User's Guide and Reference), SB0F-1426

IBM Online Library: IBM Networking Systems Softcopy Collection Kit (CD-ROM containing softcopy of all manuals except for TPNS Master Index and TPNS STL Reference Card), SK2T-6012

Related Publications

SAA Common Programming Interface Communications Reference, SC26–4399

VTAM Programming for LU 6.2, SC31–6437

Index

We'd Like to Hear from You

TPNS
Teleprocessing Network Simulator
Function and Service Enhancements-2001
Version 3 Release 5

Publication No. SC31-8654-02

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

Name	Address
Company or Organization	
Phone No.	

Readers' Comments — We'd Like to Hear from You
SC31-8654-02

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

TPNS Development and Support
Department A30A/B503
IBM Corporation
P.O. Box 12195
Research Triangle Park, NC 27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8654-02

Cut or Fold
Along Line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC31-8654-02