Workload Simulator

Creating WSim Scripts

Version 1 Release 1

SC31-8945-01

Workload Simulator

Creating WSim Scripts

Version 1 Release 1

SC31-8945-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xiii.

Second Edition (October 2015)

This document applies to the Workload Simulator Version 1 Release 1 (program number 5655-I39), 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

Contents

Figures

Tables

Notices

References in this publication to IBM® products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594
United States of America

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Trademarks and service marks

The following terms are trademarks of IBM Corporation in the United States or other countries or both:

ACF/VTAM	CICS®	IBM
IMS™	InfoWindow	MVS™
MVS/ESA	MVS/SP	GDDM
OS/390®	S/370	S/390®
SAA	Series/1	SP
System/36	System/38	System/370
Systems Application Architecture®	VTAM®	

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About this book

This book presents a discussion of how to define networks to be simulated by the Workload Simulator (WSim). It discusses how to code and evaluate a simulated network to meet the goals established by the system planner. An overview of the network definition process is provided along with examples and guidelines for coding the network resources that WSim can simulate. This book explains the various options that can be coded in a network definition for controlling how and when messages are generated for specific devices.

This book also describes how to create message generation decks for WSim network simulations using message generation statements. It also describes how to integrate message generation decks with a network definition to create a script. It provides an overview of the message generation process along with examples and guidelines for controlling message generation for the various terminal types WSim can simulate.

The final section of this book provides users of WSim examples of the scripts that they will be required to write. The examples not only show the user how to write scripts, but also can serve as prototypes for the user's message generation deck. This book also presents techniques to facilitate the creation of WSim message generation decks.

Who should read this book

Before you read this book, you should be familiar with the information presented in the *WSim User's Guide*. You will also find information related to network definition in the *WSim Script Guide and Reference* and *WSim Messages and Codes*.

Read this book if you are responsible for coding WSim networks. This book explains how various real network situations can be simulated by WSim, but you should be familiar with the network concepts that are specific to the simulation you are defining.

Read this book if you are responsible for creating message generation decks with message generation statements. If you plan to use the Structured Translator Language (STL) to create message generation decks, you should also see *WSim Script Guide and Reference*. See Chapter 10, “Getting started with message generation decks,” on page 95 to decide which method to use to create message generation decks.

How to use this book

This book contains following five parts:

Part 1, “Defining WSim networks,” on page 1, starts at an introductory level and progresses to the advanced techniques used to define different types of network resources. While this book presents complete discussions of how to code specific networks and resources, network definition is not always a self-contained process. It is often helpful to create your message generation decks with your network definition so that you can code and test them concurrently. Use this book with *WSim Script Guide and Reference* as you test and refine your network definition and message generation decks to ensure that they function as you intend.

- Chapter 1, "Introducing network definition," on page 3, provides a brief description of WSim and an overview of the network definition process.
- Chapter 2, "Understanding the network definition process," on page 11, discusses how to define a simulated network based on a test plan. It discusses the relationship between the logical configuration being simulated and the physical configuration used to run the simulation. It also introduces the basic network definition statements used to define a simple network.
- Chapter 3, "Simulating logical units using the VTAM Application Program Interface," on page 19, discusses using WSim with the VTAM Application Program Interface to simulate LU half-sessions and provides instructions for coding them.
- Chapter 4, "Simulating CPI-C transaction programs," on page 35, discusses using WSim with the Systems Application Architecture (SAA) Common Programming Interface Communications (CPI-C) to simulate client and server transaction programs and provides instructions for coding them.
- Chapter 5, "Simulating TCP/IP devices," on page 49, discusses Transmission Control Protocol/Internet Protocol (TCP/IP) resources and provides instructions for coding them.
- Chapter 6, "Simulating SNA resources and subareas," on page 61, discusses SNA resources and provides instructions for coding them.
- Chapter 7, "Simulating specific devices," on page 65, discusses the devices that require special considerations in simulations and provides instructions for coding them.
- Chapter 8, "Coding network options," on page 73, discusses the various options that can be coded in the network definition to control how the network operates and how messages are generated.
- Chapter 9, "Generating rate tables," on page 91, explains how to generate rate tables to be used by WSim in calculating terminal delays.

Part 2, "Introducing message generation decks," on page 93, introduces you to the process of coding message generation statements to create message generation decks for simulations.

- Chapter 10, "Getting started with message generation decks," on page 95, explains how WSim uses message generation decks to generate messages that are sent by simulated terminals, how message generation decks relate to network definitions, and how you create a message generation deck.
- Chapter 11, "Planning for message generation," on page 99, discusses the factors that you must consider when you plan the types of messages your simulated resources send to the system under test.

Part 3, "Coding message generation statements," on page 105, discusses the different types of message generation statements you can use to create a message generation deck and presents each statement's coding and syntax requirements.

- Chapter 12, "Basic concepts," on page 107, explains how to read and understand the syntax conventions and coding requirements for each type of message generation statement. This chapter also introduces the basic message generation statements.
- Chapter 13, "Generating messages with the TEXT statement," on page 117, explains how to use the TEXT statement to create messages that WSim sends to the system under test. With the TEXT statement, you can code messages manually or generate messages dynamically with random numbers, user table entries, data referenced with counters, and data retrieved from save and user areas.

- Chapter 14, “Understanding delimiters,” on page 137, describes delimiters, which are message generation statements that interrupt the message generation process. In this chapter, you learn how to code delimiters in a message generation deck. In addition, this chapter describes coding conditional delimiters to send messages to the system under test and interrupting message generation with unconditional delimiters.
- Chapter 15, “Understanding intermessage delays,” on page 153, explains how to use intermessage delays to control message traffic between WSim and the system under test. As discussed in this chapter, you can define intermessage delays for a network or a specific resource, and you can define multiple intermessage delays on the network definition. This chapter also discusses delaying the start of an intermessage delay and altering intermessage delays with the A (Alter) operator command.
- Chapter 16, “Defining logic tests,” on page 165, explains how to code the IF message generation statement to define message generation logic tests. With an IF statement logic test, you can test messages that are transmitted or received by simulated terminals and alter the message generation process based on the results of the test.
- Chapter 17, “Understanding control statements,” on page 195, discusses the message generation statements that control how WSim generates messages. This chapter provides information about coding control statements for specific types of devices, altering sequential processing, and controlling switches, counters, and events.
- Chapter 18, “Generating messages for specific types of devices,” on page 219, provides guidelines for generating messages for specific types of devices. This chapter includes information about generating messages for display devices and SNA terminals and the data field options provided for specific devices.

Part 4, “Using message generation decks,” on page 245, describes how to integrate decks with a network definition to create a script and how to analyze and monitor a simulation.

- Chapter 19, “Integrating decks with network definitions,” on page 247, explains how to combine your message generation decks with a network definition to create a script thatWSim uses to run the simulation. In this chapter, you learn how to define the selection of message generation decks and how to store scripts.
- Chapter 20, “Analyzing simulation results,” on page 259, explains how to use the output that is produced by WSim to analyze simulation results. In addition, this chapter provides information about running the test.
- Chapter 21, “3270 extended character set,” on page 271, lists the WSim hexadecimal values used to represent the 3270 Data Analysis/APL Character Set.

Part 5, “Samples,” on page 277, provides examples of WSim scripts. It also gives an explanation of the purpose of the script and any assumptions for the script.

- Chapter 22, “Introduction,” on page 279, gives an overview of the samples that are provided in this book.
- Chapter 23, “Sample installation networks,” on page 281, describes a group of sample WSim network definitions and message generation decks that are present on the WSim install tape in the WSim sample data set.
- Chapter 24, “Message scripting examples,” on page 289, is composed of sample scripts geared toward simulating certain applications (for example, TSO, CICS).
- Chapter 25, “AVMON example,” on page 319, provides information on the AVMON (Availability Monitor) sample network and how you can use it.

- Chapter 26, "Loglist examples," on page 375, provides loglists for selected samples.
- Chapter 27, "Network models," on page 389, provides the network models that are used by the WSim/ISPF Interface.

The Glossary lists the terms that are used in this book.

The Bibliography lists the related publications that you can use to find more information on networks.

Where to find more information

The following list shows the books in the WSim library. For more information about related publications, see the "Bibliography" on page 423.

Planning, Installation, and Operation	
WSim User's Guide	SC31-8948
WSim Messages and Codes	SC31-8951
WSim Test Manager User's Guide and Reference	SC31-8949
Resource and Message Traffic Definition	
Creating WSim Scripts	SC31-8945
WSim Script Guide and Reference	SC31-8946
WSim Utilities Guide	SC31-8947
Customization	
WSim User Exits	SC31-8950

Part 1. Defining WSim networks

Chapter 1. Introducing network definition

This book is designed to assist both the new and the experienced Workload Simulator (WSim) user in coding network simulations by presenting methods for coding both basic and advanced network simulations.

Network definition is part of the process of creating scripts. A script contains the network definition statements that describe the simulated network and the message generation decks that specify the data sent by the various resources in the network. This chapter and the next provide the new user with a basic understanding of network definition. They can also be used by the experienced user to review the basic concepts and steps involved in network definition.

In this chapter, you will find a discussion of what WSim is, where network definition fits into the overall use of WSim, and how WSim uses the network definition and message generation decks to simulate message traffic over a network. At the end of the chapter is a checklist that summarizes the steps you follow in creating scripts. After you become familiar with the process, you can use this checklist as a quick reference when you create scripts for your own simulations.

With WSim, you can code the following general types of network simulations:

- SNA logical units running as VTAM application programs
- CPI-C client or server transaction programs
- TCP/IP client applications, for Telnet 3270, File Transfer Protocol (FTP), Simple TCP clients, or Simple UDP clients.
- Specific devices within single domain network configurations.

Refer to *WSim User's Guide* for a complete list of the network configurations, terminals, and devices that you can simulate using WSim.

Creating scripts for tests

To perform tests, you must create the script that represents simulated network resources and the data that is sent by these resources. You specify the resources of your simulated network in the network definition portion of the script and the data that is sent by the resources in the message generation decks. WSim uses these message generation decks to send data from the simulated resources to the system under test (the real components of your network), and to respond to data received from the system under test as a real network would.

When you create scripts, you may be working with testing requirements that are established in a test plan. The test plan specifies the objectives of the test, the methods and resources to be used, and the schedule to be followed. The plan may indicate what type of network you must define and what types of lines, terminals, and devices the network contains. It may also indicate how many devices are attached to the network and how these devices must interact with the system under test. Refer to *WSim User's Guide* for information about test plans.

Defining the network

To define a simulated network, use network definition statements to describe the characteristics of each of the resources you want to simulate. With network definition statements, you can define devices, terminals, lines, and links and to describe the relationships between each of these resources within the simulated network and with the system under test.

In your network definition, you specify the sequence of message generation decks that are used by WSim. You can also specify logic tests that compare messages that are sent or received by WSim to expected responses. You can control the messages that are generated by WSim based on the results of these logic tests.

Each type of network simulation has specific requirements concerning how you simulate the resources and how you establish message generation. The type of network you define determines the types of terminal actions and messages you must code in your message generation decks to simulate data traffic appropriate to that network. The requirements for describing these specific networks and terminals in your network definition are discussed in this book. Refer to Part 2, "Introducing message generation decks," on page 93 for information about message generation requirements for specific types of networks.

Creating message generation decks

To create message generation decks, use message generation statements to specify the data that is sent by your simulated resources. Information about message generation statements is provided in Part 2, "Introducing message generation decks," on page 93. WSim provides two sets of tools that can help you create message generation decks: the Structured Translator Language (STL) and the script generating utilities. For information about these tools, refer to *WSim Script Guide and Reference* and *WSim Utilities Guide*.

A method for creating scripts

Creating the network definition and message generation decks that form a script can be an interdependent process. Because the features you code in the network definition statements directly affect how and when messages are sent, you may want to closely coordinate the coding of these two parts of your scripts to ensure that WSim sends messages in ways that accurately reflect the actions of specific terminals and terminal operators.

An effective approach to creating a script is to begin with a simple network with only one terminal and a basic message generation deck. Test this network and message generation deck and add more message generation decks one by one. Add network components in small groups and continue to test the script until you complete all the coding that you need for the test.

If you create your network definition and message generation decks using this method, you find that the process proceeds more smoothly than if you attempt to code your entire network first and then try to test message generation decks with this large network. By testing with a small network consisting of only a few terminals, you can easily determine where errors occur and make corrections as you add message generation decks.

With this approach to creating a script, you can also specify options easily in both your network definition and your message generation decks that affect how and when individual message generation decks are used. The definitions for some features of network simulations, such as time delays and logic tests, can be coded in both the network definition and the message generation decks. Moving back and forth between your network definition and your message generation decks enables you to effectively coordinate the coding of such features.

The Structured Translator Language is a good tool for coding a simulation because you can create the network definition and messages together in one input data set with the tool. Refer to *WSim Script Guide and Reference* for more information.

Chapter 2, “Understanding the network definition process,” on page 11 introduces the basic definition statements you can use to define networks and the lines, devices, and terminals within them. Subsequent chapters provide more detailed instruction for coding specific networks and their resources.

Example of a script

Below shows an example of a script that contains a network definition and several message generation decks. This script defines a simple network in which WSim simulates a single logical unit (LU) representing a VTAM application. WSim uses basic message generation decks to log the LU onto TSO and log it off again.

Figure 1 shows how to write the script in the WSim scripting language and Figure 2 on page 7 shows how to write the same script in STL. For more information on coding scripts in STL, refer to *WSim Script Guide and Reference*.

Later chapters in this book explain the statements used to define this network. Part 2, “Introducing message generation decks,” on page 93 explains the statements used in the message generation decks and describes how WSim processes the decks and how they interact with the network definition. For now, note the relationship between the network definition, which starts with the NTWRK statement, and the message generation decks, which start with MSGTXT statements and end with ENDTXT statements. The network definition is always placed at the beginning of a script. It contains statements that specify characteristics for the entire network and definitions of individual devices within the network. The statements immediately following the NTWRK statement specify the characteristics for the entire network. Within the definitions for individual devices, more characteristics can be specified to further define the devices or to override values specified for the network. The message generation decks follow the network definition. A script can contain any number of message generation decks. Through network definition statements and message generation statements, you control the sequence in which the message generation decks are used by the simulated devices.

In this example, the order of message generation decks that WSim follows is specified by the PATH statement. The first message generation deck, INITSESS, logs the LU on to TSO, which is using the application ID TSO01. This message generation deck checks to see if the LU enters ISPF after successfully logging on. If it does, WSim proceeds to the next message generation deck, LOGOFF, specified by the PATH statement. If the LU does not enter ISPF, WSim calls another message generation deck, ISPFDECK, to invoke ISPF. WSim then uses the message generation deck LOGOFF to log the LU off TSO. WSim uses the message generation deck CLEAR to clear the screen any time the characters “X'1DC8'***” appear in the request unit (RU+0).

Figure 1. Sample script in WSim scripting language

```
SAMP1    NTWRK     HEAD='Sample Network 1',  Heading for interval reports.
                   UTI=100,                  Network user time interval.
                   MSGTRACE=YES,             Logs MTRC records.
                   BUFSIZE=5000              5000-byte buffer for LU.
```

```
***************************************************************************
*                       Sample WSim Script                                *
*   The following script defines a simple network in which WSim           *
*   simulates a single LU (SLU) representing a locally attached 3270      *
*   terminal.  WSim uses three message generation decks to log the LU onto *
*   TSO and then off again.                                               *
***************************************************************************
*   Coding for network named SAMP1:
*
NETIF    IF        LOC=RU+0,
                   TEXT=('1DC8'***),         Defines logic test for messages
                   THEN=CCLEAR,WHEN=IN,      received from the system under
                   SCAN=YES                  test.
*
SIMPLE   PATH      INITSESS,LOGOFF           Specifies the sequence in which
*                                            WSim processes message decks
*                                            named INITSESS and LOGOFF.
*
WSIMAPPL VTAMAPPL                            Defines VTAM application named
*                                            WSIMAPPL.
SLU      LU        MAXSESS=(0,001),          Defines one secondary half
                   INIT=SEC,                 session for SLU.  Specifies
                   LUTYPE=LU2,               LU type and the name of a VTAM
                   DLOGMOD=D4A32782,         logon mode table entry, delays
                   THKTIME=UNLOCK,           start of intermessage delays,
                   LOGDSPLY=BOTH             and specifies that display
*                                            buffers are written to the
*                                            log data set.
*
INITSESS MSGTXT
***************************************************************************
*   INITSESS deck simulates SLU logging on to TSO.  In this deck, the TSO  *
*   resource name is TSO01, the user ID is ID02 and the password is PW02.  *
***************************************************************************
*
*                                            Beginning of INITSESS.
WTO1     WTO       (STARTING $MSGTXTID$)     Message to operator console.
CMND1    CMND      COMMAND=INIT,             Initiates session.
                   RESOURCE=TSO01
0        IF        LOC=RU+0,                 Tests for characters ENTER
                   TEXT=(ENTER USERID),      USERID.
                   SCAN=YES,THEN=B-MSG1
WAIT1    WAIT
         BRANCH    LABEL=WAIT1               In case bind resets the wait.
*                                            Interrupts message generation.
MSG1     TEXT      (ID02)                    Enters user ID ID02.
WTO2     WTO       (Logging on TSO as ),     Message to operator console.
                   (ID02)
ENTER1   ENTER                               Sets ENTER AID byte.
0        IF        LOC=RU+0,                 Tests for characters ENTER
                   TEXT=(ENTER LOGON),       LOGON.
                   SCAN=YES,THEN=CONT
WAIT2    WAIT
*                                            Interrupts message generation.
MSG2     TEXT      (PW02)                    Enters user password PW02.
ENTER2   ENTER                               Sets ENTER AID byte.
0        IF        LOC=RU+0,                 Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),         PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
1        IF        LOC=RU+0,TEXT=(READY),    Tests for characters READY
                   SCAN=YES,THEN=CISPFDECK   and calls ISPFDECK to invoke
*                                            ISPF.
WAIT3    WAIT
*                                            Interrupts message generation.
WTO3     WTO       (Logged on and ),         Message to operator console.
                   (received ISPF Primary ),
                   (Menu)
         ENDTXT                              End of INITSESS.
*
ISPFDECK MSGTXT
****************************************************************************
*   ISPFDECK simulates SLU accessing ISPF and waiting for the ISPF         *
*   primary menu to be returned before continuing message generation.      *
****************************************************************************
*
*                                            Beginning of ISPFDECK.
WTO1     WTO       (STARTING $MSGTXTID$)     Message to operator console.
MSG1     TEXT      (ISPF)                    Enters ISPF.
ENTER1   ENTER                               Sets ENTER AID byte.
0        IF        LOC=RU+0,                 Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),         PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
WAIT1    WAIT
*                                            Interrupts message generation.
WTO1     WTO       (Logged on and ),         Message to operator console
                   (received ),
                   (ISPF Primary Menu)
         ENDTXT                              End of ISPFDECK.
*
LOGOFF  MSGTXT
****************************************************************************
*   The following message generation deck exits ISPF and logs the LU      *
*   off TSO.                                                               *
****************************************************************************
*
*                                           Beginning of LOGOFF.
WTO1    WTO       (Starting $MSGTXTID$)     Message to operator console.
MSG1    TEXT      (X)                       Type X to exit ISPF.
0       IF        LOC=RU+0,TEXT=(READY),    Tests for characters READY.
                  SCAN=YES,THEN=CONT
WAIT1   WAIT
*                                           Interrupts message generation.
MSG2    TEXT      (LOGOFF)                  Enter text LOGOFF.
WTO2    WTO       (Logged off TSO)          Message to operator console.
ENTER1  ENTER
*                                           Interrupts message generation.
OPCMND1 OPCMND    (ZEND)                    Specifies the operator
WAIT2   WAIT                                command ZEND to end WSim execution.
        ENDTXT                              End of LOGOFF.
*
CLEAR   MSGTXT
****************************************************************************
*   WSim calls the following message generation deck to clear the screen   *
*   when *** appears in the request unit (RU+0).                           *
****************************************************************************
*
*                                           Beginning of CLEAR.
CLEAR1  CLEAR                               Clears the screen.
        ENDTXT                              End of CLEAR.
```

Figure 2. Sample script in STL

```
@network
************************************************************************
*                        Sample WSim Script                            *
*   The following script defines a simple network in which WSim        *
*   simulates a single LU (SLU) representing a locally attached 3270   *
*   terminal.  The three procedures are used by WSim to log the LU     *
*   onto TSO and then off again.                                       *
```

```
**********************************************************************
*   Coding for network named SAMP1:
samp1    ntwrk    head='Sample Network 1',  Heading for interval reports
                  uti=100,                  Network user time interval.
                  msgtrace=yes,             Logs MTRC records.
                  bufsize=5000              5000-byte buffer for LU.
netif    if       loc=ru+0,
                  text=('1dc8'***),         Defines logic test for
                  then=cclearkey,when=in,   messages received from the
                  scan=yes                  system under test.
*
simple   path     initsess,logoff           Specifies the sequence in
*                                           which WSim processes message
*                                           decks named INITSESS and
*                                           LOGOFF.
*
cmm1     vtamappl
*                                           Defines VTAM application
*                                           named WSIMAPPL.
*
slu      lu       maxsess=(0,001),          Defines one secondary half
                  init=sec,                 session for SLU. Specifies
                  lutype=lu2,               LU type and the name of a
                  dlogmod=d4a32782,         VTAM logon mode table
                  thktime=unlock,           entry, delays start of
                  logdsply=both             intermessage delays, and
*                                           specifies that display
*                                           buffers are written to the
*                                           log data set.
@endnetwork
/********************************************************************/
/*   INITSESS simulates SLU logging on to TSO.  In this procedure,   */
/*   the TSO resource name is TSO01, the user ID is ID02 and         */
/*   the password is PW02.                                           */
/********************************************************************/
initsess:   msgtxt
            say 'STARTING' msgtxtid()
onin1:      onin index(RU,'ENTER USERID') &gt; 0 then found = on
            found = off
            initself('TSO01')
            do while found = off
             wait until onin
            end
            deact onin1
            type 'ID02'
            say 'Logging on TSO as ID02'
            transmit using enter,
             and wait until onin index(RU,'ENTER LOGON') &gt; 0
            type 'PW02'
onin2:      onin index(RU,'READY') &gt; 0 then call ispfdeck
            transmit using enter,
             and wait until onin index(RU,'ISPF/PDF PRIMARY') &gt; 0
            deact onin2
            say 'Logged on and received ISPF Primary Menu'
            endtxt
/********************************************************************/
/*   ISPFDECK simulates SLU accessing ISPF and waiting for the ISPF  */
/*   primary menu to be returned before continuing message           */
/*   generation.                                                     */
/********************************************************************/
ispfdeck:   msgtxt
            say 'STARTING 'msgtxtid()
            type 'ISPF'
```

```
           transmit using enter,
            and wait until onin index(RU,'ISPF/PDF PRIMARY') &gt; 0
           say 'Logged on and received ISPF Primary Menu'
           endtxt
/*******************************************************************/
/*   LOGOFF exits ISPF and logs the LU off TSO.                    */
/*******************************************************************/
logoff:    msgtxt
           say 'Starting 'msgtxtid()
           type 'X'
           transmit using enter,
            and wait until onin index(RU,'READY') &gt; 0
           type 'LOGOFF'
           transmit using enter
           say 'Logged off TSO'
           opcmnd 'zend'
           wait
           endtxt
/*******************************************************************/
/*   CLEARKEY simulates a user hitting the clear key.              */
/*******************************************************************/
clearkey:  msgtxt
           transmit using clear
           endtxt
```

Checklist for creating scripts

The following checklist summarizes the steps that you follow to create a script and indicates where you can find instructions for performing each step. The first step is preparation that you should make before coding your scripts. The remaining steps provide a method that can help you code and test your scripts efficiently. After you gain a basic understanding of network definition and message generation, you can refer to this checklist each time you create scripts for your simulations.

Preparation:

1. Determine the resources you need to simulate and the configuration of real components you need to perform the simulation, based on the test plan. See Chapter 2, "Understanding the network definition process," on page 11.

   Coding and Testing:
2. Code a small test network based on the test plan. This network must consist of only one terminal. See Chapter 3, "Simulating logical units using the VTAM Application Program Interface," on page 19 through Chapter 5, "Simulating TCP/IP devices," on page 49 for coding instructions for specific networks and Chapter 7, "Simulating specific devices," on page 65 for coding instructions for specific devices. Also, refer to *WSim Script Guide and Reference* for information on how to code the network definition as part of an STL input data set.
3. Code a basic message generation deck based on the test plan. See *WSim Script Guide and Reference*, *WSim Utilities Guide*, and Part 2, "Introducing message generation decks," on page 93 for information about coding message generation decks.
4. Run the network definition and message generation deck through the Preprocessor to determine if the syntax is correct and to store the network definition and message generation deck statements for later use. If you are using the Structured Translator Language, you can run the STL Translator to check the syntax of your message generation decks and store them. The STL Translator also invokes the Preprocessor for you if you include the network definition in your STL input.

If the Preprocessor or STL Translator detects any errors, fix them and run the utility again until there are no errors. See *WSim Utilities Guide* for information about the Preprocessor and *WSim Script Guide and Reference* for information about the STL Translator.

5. Perform a test run to make sure the simulation works as you intended.
6. Use the Loglist Utility to process the results, if wanted. See *WSim Utilities Guide* for information about the Loglist Utility.
7. Continue coding message generation decks one by one and adding network components in small groups. Translate or preprocess your script, whichever is appropriate, and perform test runs until you have all the message generation decks and network components you need for your simulation. Include all the network options you need for your tests. See Chapter 8, “Coding network options,” on page 73 and the *WSim Script Guide and Reference* for information about the various options you can code in your network definition.

Chapter 2. Understanding the network definition process

This chapter discusses how to define a simulated network based on the information in the test plan. It explains how to determine the configuration of network resources that WSim simulates and the configuration of real components that WSim requires to run the simulation. Illustrations show the general configuration types that you can simulate and the configurations of components that you use to perform these simulations.

After you identify the network resources that you want to simulate, you can code your simulated network using network definition statements. You must use specific combinations of network definition statements and code certain operands on these statements to describe the devices and lines that correspond to the type of configuration you are simulating. This chapter introduces the basic network definition statements that are used to define a simple Systems Network Architecture (SNA) network. By understanding how the basic statements are used in this simple network, you will be prepared to code the more complex networks discussed later in this book. Finally, this chapter discusses the Preprocessor, which you can use to check the syntax of your coded statements.

Defining a network using the test plan

The requirements for a simulation are established in a test plan. Depending on the procedures followed by your organization, you might be responsible for writing the test plan or you might be provided with a completed test plan. If you are responsible for writing the test plan, refer to *WSim User's Guide* for details about the type of information to include in the test plan.

A test plan generally contains the following information:

- Objectives for the simulation
- Methods for performing the simulation
- Existing resources to be incorporated into the simulation
- Resources to be simulated by WSim
- Reporting requirements
- Schedule for setting up and running the simulation.

Before defining your network, you can determine from the test plan what types of devices and terminals you need to simulate and how they interact with the system under test.

Determining the logical and physical configuration

The test plan indicates the number and types of resources you must simulate with WSim. When these simulated resources interact with the real components of the system under test, a logical configuration representing a real network is created. The real components can include anything that you want to test with WSim, such as host processors, communication controllers, cluster controllers, terminals, and application programs.

From the logical configuration, you can determine the physical configuration of hardware and software that WSim requires to run the simulation. The physical configuration also represents how WSim is connected to the system under test.

You can perform the following general types of simulations with WSim:

VTAM application simulation
VTAM applications, logical units, or CPI-C transaction programs that access real applications in the system under test

TCP/IP client simulation
Telnet 3270, 3270E, 5250, NVT, FTP, or Simple TCP or UDP clients that access real applications in the system under test

To perform these simulations, you must run WSim in one of the following physical configurations:

- **VTAMAPPL** for VTAM application or CPI-C transaction program simulations
- **TCP/IP** for TCP/IP client simulations

The following figures show the relationship between the logical configurations that include these simulated resources and the physical configurations that are used to perform the simulations. In the illustrations of the logical configurations, the components that WSim simulates are shown within a dotted box.

VTAM application simulation

Figure 3 illustrates the logical configuration of a system in which WSim simulates SNA logical units (LUs) accessing VTAM applications. To VTAM, these LUs appear as other application programs, while to the system under test, they appear as applications or terminals. Although WSim does not simulate lines, links, or other hardware in this configuration, it does simulate the activities of the LUs and can be used to test applications or subsystems through the VTAM Application Program Interface (VTAM API).

Note: WSim also uses the VTAM application simulation type to simulate CPI-C transaction programs. The logical configuration for a CPI-C transaction program simulation is the same as that depicted in Figure 3 except that LUs in that figure become TPs.

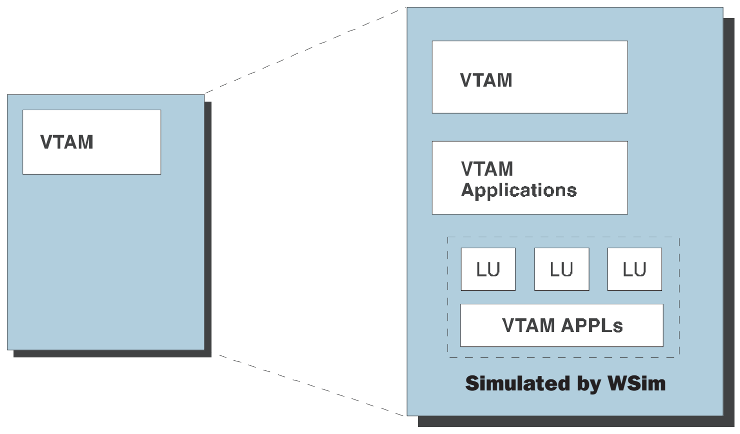


Figure 3. VTAM application simulation—logical configuration

Figure 4 on page 13, Figure 5 on page 13, and Figure 6 on page 13 illustrate the physical configurations you can use to run a VTAM application simulation. In these physical configurations, WSim runs as a VTAM application and uses the VTAM Application Program Interface (VTAM API).

WSim can run in the same host processor as VTAM, as shown in Figure 4, or in a separate host processor, as shown in Figure 5 and Figure 6. If you are conducting tests in which it is important that WSim execution does not affect the system under test, you might want to run WSim in a separate host processor. Refer to *WSim User's Guide* for more information about the types of tests that require running WSim in a host processor separate from the system under test.

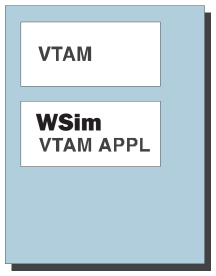


Figure 4. VTAM application simulation—physical configuration (one host processor)

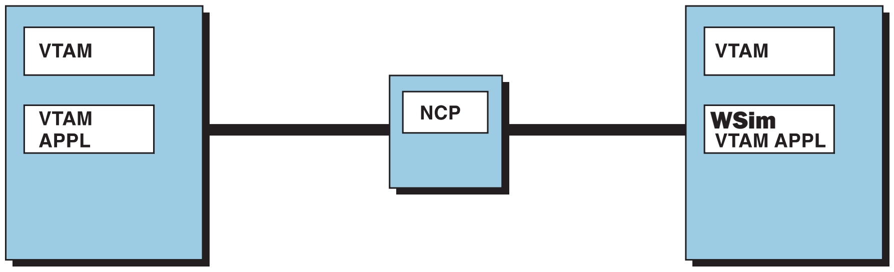


Figure 5. VTAM application simulation—physical configuration (two host processors)

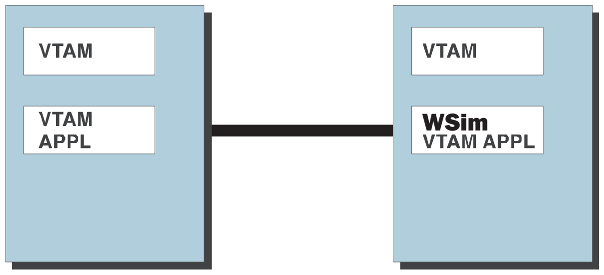


Figure 6. VTAM application simulation—physical configuration (channel attached)

The VTAM subarea can be part of a large network with many other nodes. The real applications with which the simulated applications communicate might be in some other node in the network. WSim and VTAM can be in one host processor and test applications in a different host processor that is attached through an NCP or a channel-to-channel adapter.

TCP/IP client simulation

Figure 7 on page 14 illustrates the logical configuration of a system in which WSim simulates the TCP/IP client applications of the system under test. The clients simulated by WSim can be Telnet 3270, 3270E, 5250, NVT,File Transfer Protocol (FTP), or Simple TCP or UDP.

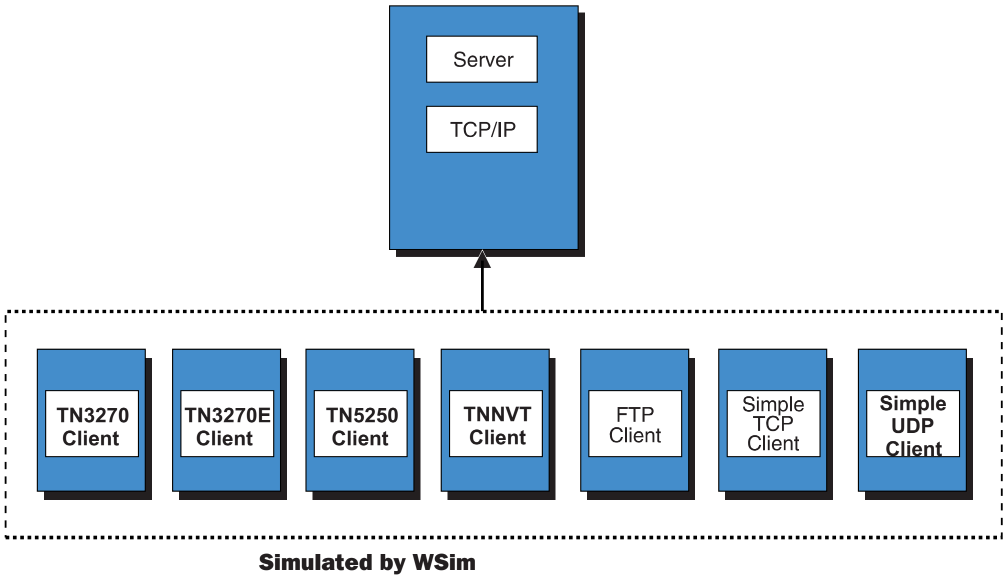


Figure 7. TCP/IP client simulation—logical configuration

Figure 7 illustrates the physical configuration of TCP/IP client network in which WSim simulates TCP/IP client applications within the system under test.

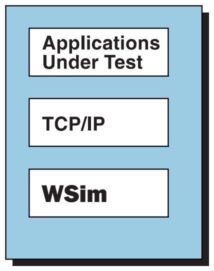


Figure 8. TCP/IP client simulation—physical configuration

Figure 8 illustrates the physical configuration you can use to run the simulations shown in Figure 7.

The basic network definition statements

After you determine the configuration of network resources that you want to simulate, you can define your simulated resources. Use network definition statements to specify the characteristics of all the resources in your network. Enter these network definition statements in MVS data sets (see "Allocating WSim data sets on MVS" on page 16).

Specific network definition statements apply to the different resources found in each type of network that is discussed in this book. Refer to *WSim Script Guide and Reference* for full descriptions and coding requirements for all of the statements used in a network definition.

Each network definition statement consists of the following three parts:

Name This is the first part of each statement and provides a label for the statement. The name begins in column 1 and can be up to 8 alphanumeric characters.

Statement	This is the actual language statement. It is the term used to refer to the statement in text discussions and *WSim Script Guide and Reference*. The statement must be separated from the name by at least one blank and cannot begin in column 1.
Operand	This represents any number of optional operands that are available for the statement. The operands must be separated from the statement by at least 1 blank. The last column available for defining operands is column 71.

Note: All of the examples in this book show network definition statements in uppercase. You can code network definition statement in mixed case.

To understand the process of coding network definition statements, it is helpful to start with the statements used for a basic VTAMAPPL network in which WSim simulates SNA logical units (LUs) accessing VTAM applications. For this basic simulation, you use the following 4 statements to define the resources WSim simulates:

- NTWRK
- PATH
- VTAMAPPL
- LU

The 4 basic network definition statements are explained in the following sections.

NTWRK statement

The NTWRK statement is the first statement in all network definitions. It provides a name for the network and specifies characteristics that apply to the network as a whole. You can code operands on the NTWRK statement to establish default values for the VTAMAPPL and LU statements that are coded after the NTWRK statement. The operands for the NTWRK statement that apply to specific networks are discussed in the chapters concerning those networks. Chapter 8, “Coding network options,” on page 73 discusses operands that specify options you can use for all network simulations.

The following example shows the beginning of a network definition for a network named TESTNET. This NTWRK statement specifies message tracing for all LUs defined in the network.

```
TESTNET  NTWRK  MSGTRACE=YES
```

Note: If you are using STL, you would code STLTRACE=YES in place of MSGTRACE=YES to specify message tracing of the STL program.

PATH statement

The PATH statement identifies the sequences of message generation decks. WSim uses these sequences to generate messages for terminals.

In the following example, if you specify PATH1 for a terminal, that terminal runs DECK1. When the terminal finishes running DECK1, it then runs DECK2. After it finishes running DECK2, it starts over with DECK1. The terminal repeats the execution of the message generation decks on the paths indefinitely until the operator stops the terminal.

```
PATH1  PATH  DECK1,DECK2
```

VTAMAPPL statement

The VTAMAPPL statement defines the VTAM application program symbolic name and the password associated with the VTAM application symbolic name.

LU statement

The LU statement defines one or more logical unit half-sessions to be simulated using the WSim/VTAM application program interface, and it defines the type of logical unit simulation to be performed for the SNA half-sessions.

Hierarchy of the basic network definition statements

You must code the NTWRK, PATH, VTAMAPPL, and LU statements that define the simulated resources in a specific order. The PATH statement must precede all VTAMAPPL or LU statements. The NTWRK statement must be the first statement in the network definition. The LU statements must follow the VTAMAPPL statements with which they are associated. If you define multiple VTAMAPPLs and LUs, the statements must be ordered in a hierarchical fashion as shown in the following example.

```
NTWRK
PATH
  VTAMAPPL
    LU
  VTAMAPPL
    LU
    LU
```

For other types of network simulations, such as CPI-C and TCP/IP, refer to *WSim Script Guide and Reference* for information about the required hierarchy of statements used to define the resources.

Allocating WSim data sets on MVS

If you are running your simulations on MVS, you must allocate data sets to contain your network definition statements and message generation decks. The following example shows an example of a data set allocation job.

```
//ALLOCATE  JOB
//STEP1     EXEC PGM=IEFBR14
//INITDD    DD  DSN=WSIM.TESTFILE,UNIT=3380,VOL=SER=WSIMPK,
//              SPACE=(CYL,(10,,10)),DISP=(NEW,CATLG),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//MSGDD     DD  DSN=WSIM.MSGFILE,UNIT=3380,VOL=SER=WSIMPK,
//              SPACE=(CYL,(10,,10)),DISP=(NEW,CATLG),
//              DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
```

INITDD is the data set that contains your network definition statements. MSGDD is the data set that contains your message generation statements. As described in "Preprocessing your script" on page 17, you might need to increase the block size of these data sets as you preprocess your script.

Note: The BLKSIZE for these data sets must be a multiple of 80 and the LRECL must be 80. Data sets can also be allocated from ISPF under TSO.

Preprocessing your script

WSim provides a Preprocessor you can use to check the syntax of the network definition and message generation statements you code. The Preprocessor places the statements into data sets that you use when running tests. If you use the Preprocessor to check your statements, you can ensure that your network can initialize without syntax errors. However, the Preprocessor only checks the correctness of the coding. Once the coding is correct, you should determine whether the statements produce the expected outcome by running tests.

If you are using the Structured Translator Language to create your message generation decks, you can use the STL Translator to store your statements and check their syntax, like the Preprocessor. See *WSim Script Guide and Reference* for information about using the STL Translator. You can also include your network definition in the STL input data set. The STL Translator invokes the Preprocessor for you to check the syntax of your network definition and store it.

The input for the Preprocessor consists of the data sets that contain your network definition statements and message generation decks. The output consists of a listing of the statements you coded and error message for lines that contain errors. It also stores your network definition and message generation decks in the INITDD and MSGDD data sets, if they are syntactically correct.

When you use the Preprocessor, it builds most of the control blocks necessary to run the network. Therefore, when you preprocess a large network, you must have a region size large enough for the network control blocks. Statements at the end of the Preprocessor output indicate the number of bytes required to store the network control blocks and message generation decks. Check these values each time you obtain Preprocessor output to determine when you might need to increase your region size to continue preprocessing your network definition and message generation decks. Refer to *WSim Utilities Guide* for specific instructions for using the Preprocessor and analyzing the results.

Note: The Preprocessor indicates only the amount of storage needed to store the network definition and message generation decks. It does not indicate the amount of storage needed to run the network simulation. Refer to *WSim User's Guide* for a description of the various methods you can use to calculate storage requirements.

If you want to store your network definition and message generation decks without preprocessing them, you can use the utility program ITPSYSIN. ITPSYSIN stores your statements without checking the syntax. If you want to store previously processed scripts, scripts with minor changes, or automatically generated scripts, you can store them quickly using ITPSYSIN. For more information about ITPSYSIN, see *WSim Utilities Guide*.

Chapter 3. Simulating logical units using the VTAM Application Program Interface

This chapter discusses how to use WSim to simulate logical units (LUs) using the VTAM Application Program Interface (VTAM API). You can use WSim as a VTAM application program to test existing VTAM applications or applications that you are developing. You can test application programs with WSim in the following ways:

- Simulate secondary LU terminals that log on to and test VTAM applications.
- Simulate primary LU application prototypes that can be logged on to and tested by real terminals or simulated terminals.

The first part of this chapter briefly explains how WSim supports VTAM application simulations. It then explains how to define VTAM application resources. An illustration and complete network definition provide an example of a simulated VTAM application configuration. Finally, this chapter discusses how VTAM application sessions are initiated and terminated by both WSim and VTAM.

WSim support for VTAM application simulations

WSim uses the VTAM API to simulate single, multiple, and parallel logical unit half-sessions that can be primary or secondary. Through the API, WSim appears to VTAM as one or more VTAM application programs. Because VTAM treats an application program as a logical unit, WSim can simulate logical unit half-sessions that look like SNA terminals to other VTAM application programs in the system under test.

You can simulate secondary LUs to test application programs and network resources—for example, simulated 3270 LU2s exercising real TSO, IMS/VS, CICS, or CMS applications. Or, you can simulate primary LUs to test application prototypes—for example, simulated application programs that real terminals can log on to. By using WSim to simulate application prototypes, you can create simulated panels, logic, and error messages, and scripts that can be used later to test the real program code.

WSim uses the authorized path facility in VTAM when it is APF-authorized, thus improving performance. Also, WSim does not drive any hardware directly when running exclusively as a VTAM application. No communication controller or attached NCP is required to simulate logical units. WSim uses the VTAM API to send and receive messages, eliminating the need for any extra hardware resources. To run as a VTAM application program without a communication controller, WSim requires only a currently supported release of VTAM and the appropriate VTAMLST APPL definitions that enable WSim to communicate with VTAM.

WSim only simulates local SNA logical units within a VTAM subarea. However, this does not limit a simulated VTAMAPPL LU to same-domain sessions because VTAM might be able to route data to and from partner LUs in other domains.

Note that a VTAM application simulation does not represent an entirely realistic terminal emulation because WSim performs the simulation entirely within its own software, without a communication controller. Various system components, such as the host processor, operating system, and even VTAM, will “know” that the simulated LUs are not real terminals. However, WSim does present a logically realistic simulation of local SNA primary or secondary LUs, and therefore, can effectively represent the actions of these LUs accessing application programs.

You can use the full capabilities of message generation statements, as they apply to SNA logical units, to simulate local terminals or applications. See *WSim Script Guide and Reference* or Part 2, “Introducing message generation decks,” on page 93 for information about the message generation statements used to simulate SNA resources.

Notes:

- A secondary logical unit in a WSim VTAM application simulation cannot communicate with a Network Routing Facility (NRF) primary logical unit that runs within an NCP because of NRF restrictions.
- When WSim simulates a VTAMAPPL LU and the application program that it communicates with issues a VTAM SIMLOGON OPTCD=Q command, the VTAMAPPL simulated LU behaves differently than a real terminal.

  With a real terminal, VTAM waits until the session with the SLU terminal ends and then schedules the logon exit associated with the SIMLOGON OPTCD=Q command. This is because the real terminal has a static session limit of one.

  With the WSim VTAMAPPL LU, VTAM schedules the logon exit associated with the SIMLOGON OPTCD=Q command immediately. This is because there is no session limit for a VTAM application program. When this occurs, the application program usually sends the BIND for the next session to the WSim VTAMAPPL LU before the first session ends. WSim queues the BIND until a secondary LU under the VTAMAPPL can accept the BIND.

Defining VTAM application resources

This section describes the network definition statements you can use to define VTAM application resources. It also discusses coding considerations you must be aware of when defining these resources.

Network definition statements for VTAM resources

Use the VTAMAPPL and LU network definition statements to define a VTAM application program and each logical unit half-session that WSim simulates. In VTAM, the application program defined by these network definition statements is a logical unit capable of supporting multiple and parallel sessions. A network definition can define any number of VTAM application programs.

Note: “VTAMAPPL” is the network definition statement that defines simulated VTAM applications. “VTAM APPL” refers to the APPL statement in a VTAMLST that defines a single VTAM application in a VTAM domain.

The network definition statements used to simulate VTAM applications are described as follows.

VTAMAPPL statement

Use the WSim VTAMAPPL statement to define a VTAM application program. You can use VTAM-unique operands on the VTAMAPPL statement to build a VTAM access method control block (ACB). The name of the VTAMAPPL statement identifies it to VTAM. The following operands on the VTAMAPPL statement specify various characteristics of the simulated VTAM application program:

- APPLID defines the symbolic name of the application program. This symbolic name is either the name of an active VTAM APPL resource or the ACBNAME operand value coded on a VTAM APPL statement.
- PASSWD specifies the password associated with the application program. This password must be the same as the PRTCT operand value coded on the VTAM APPL statement.

LU statement

Use the LU statement to define one or more logical unit half-sessions and the type of half-session (logical unit type, whether the LU is primary or secondary) to be simulated. Any number of LU statements can follow a VTAMAPPL statement. WSim simulates each logical unit half-session as a separate entity (such as a single display, device, or terminal) for message generation and message logging. The name of the LU statement identifies it to WSim. The following operands on the LU statement specify various characteristics of the simulated logical unit half-session:

- MAXSESS defines the number of primary and secondary half-sessions to be simulated.
- LUTYPE defines the type of logical unit half-session to be simulated. The type of logical unit half-session has meaning only for secondary half-sessions.
- DLOGMOD specifies the name of a logon mode table entry for the LU.
- RESOURCE identifies the name of a VTAM LU with which the WSim LU can initiate a session. When running with VTAM Version 3, the RESOURCE operand must specify the name appearing on the VTAMLST APPL statement that is used to define the LU to VTAM. Do not use the ACBNAME coded on the VTAMLST APPL statement as the RESOURCE operand value. See Figure 10 on page 23 and Figure 11 on page 24 for examples of how to code the RESOURCE operand.

Other operands on the LU statement define logical unit half-session characteristics that WSim uses to interpret and generate unique data streams associated with products such as 3270 and 5250 devices.

Figure 9 shows the logical configuration of a VTAM network in which WSim simulates one secondary LU2.

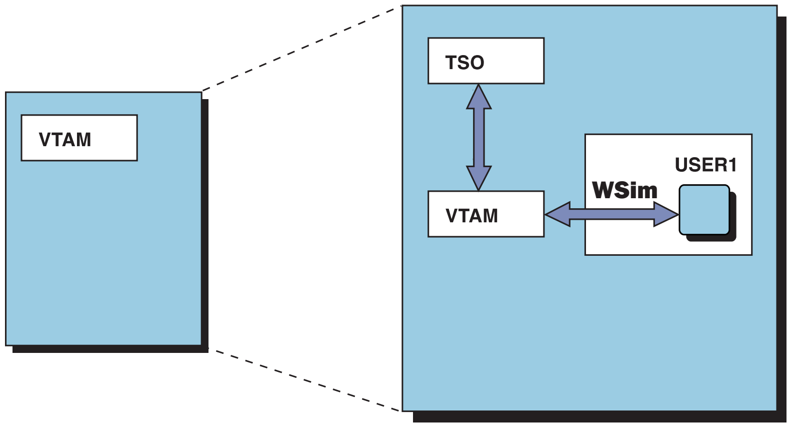


Figure 9. VTAM Network simulating a secondary LU2—logical configuration

The following example shows the network definition for this VTAM network. In this example, the WSim LU2 is known to VTAM as WSIMLU. It is known to WSim as USER1.

```
VTAMNET NTWRK BUFSIZE=2048,THKTIME=UNLOCK,INIT=SEC
0       PATH  LOGON
WSIMLU  VTAMAPPL
USER1   LU    LUTYPE=LU2,
              RESOURCE=TSO
```

WSim VTAMAPPL coding considerations

If the application that WSim is communicating with requires a unique resource name for each logical unit half-session, you must code multiple VTAMAPPL statements, each with a single LU statement, to simulate a single logical unit half-session. If the application that WSim is communicating with does not require a unique resource name for each logical unit half-session and also supports parallel sessions, you can code a single VTAMAPPL statement, followed by one or more LU statements, to simulate multiple logical unit half-sessions.

VTAM APPL coding considerations

Each simulated VTAM application defined in a network requires an active VTAM APPL when the ACB associated with the simulated VTAM application is opened. Use operands on the VTAM APPL definition statement to specify the name of the APPL, the authority of the APPL to use certain VTAM functions, and whether the simulated VTAM application using the APPL supports parallel sessions.

The VTAM APPL can have any valid name. If the VTAM application program that WSim communicates with requires a unique resource name for each SNA session that it supports, the name of the APPL must be one of the valid resource names. For example, when communicating with a CICS application, the APPL name must be defined in the CICS Terminal Control Table (TCT) as the name of a VTAM logical unit, such as an LU2 for a 3270 terminal.

You must specify AUTH=(ACQ) on the VTAM APPL definition when WSim is simulating primary LU half-sessions that initiate the session, rather than letting the secondary LU initiate the session.

You must specify PARSESS=YES on the VTAM APPL definition when the WSim VTAM application has more than one active session, such as parallel sessions, with another VTAM application or logical unit.

You must *not* code SRBEXIT=YES, APPC=YES, or AUTH=TSO on the VTAM APPL definition.

If you are going to use a VTAM APPL for a single session, EAS=1 and SESSLIM=YES must be specified. Specifying EAS=1 holds down the amount of CSA storage VTAM allocates for the APPL. Specifying SESSLIM=YES forces a session limit of one to the application program, which is consistent with a real terminal such as a 3270 LU type 2. SESSLIM=YES is supported by VTAM starting with VTAM V3R4.

For VTAM application simulation, VTAM manages the encryption and decryption of messages. Refer to the cryptographic keys section in *VTAM Network Implementation Guide*, for information on defining SLU cryptographic keys.

Coordinating WSim, VTAM, and subsystem definitions

Before a session can be established between your simulated VTAMAPPL LU and a subsystem within the system under test, you must make sure that your resource definitions are consistent with your VTAM and subsystem definitions. Figure 10 and Figure 11 on page 24 show how you must coordinate resource definitions for two common subsystems, CICS and IMS/VS.

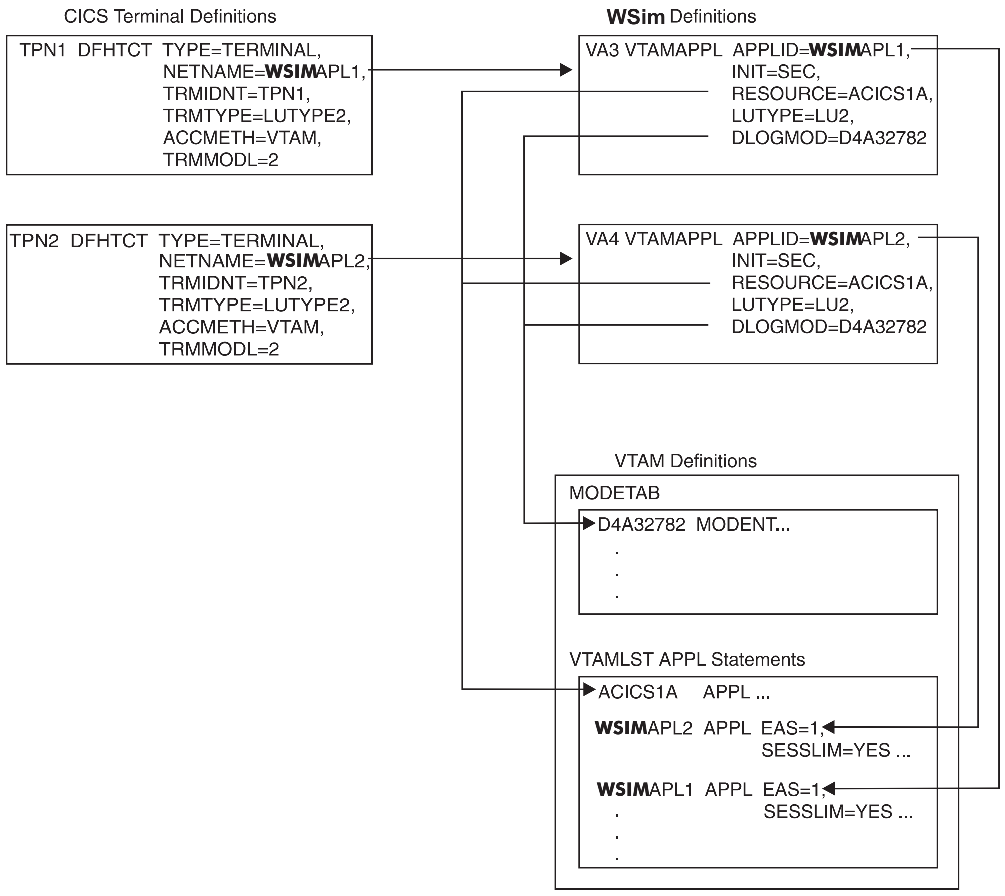

CICS Terminal Definitions
TPN1 DFHTCT TYPE=TERMINAL, NETNAME=WSIMAPL1, TRMIDNT=TPN1, TRMTYPE=LUTYPE2, ACCMETH=VTAM, TRMMODL=2
TPN2 DFHTCT TYPE=TERMINAL, NETNAME=WSIMAPL2, TRMIDNT=TPN2, TRMTYPE=LUTYPE2, ACCMETH=VTAM, TRMMODL=2
WSim Definitions
VA3 VTAMAPPL APPLID=WSIMAPL1, INIT=SEC, RESOURCE=ACICS1A, LUTYPE=LU2, DLOGMOD=D4A32782
VA4 VTAMAPPL APPLID=WSIMAPL2, INIT=SEC, RESOURCE=ACICS1A, LUTYPE=LU2, DLOGMOD=D4A32782
VTAM Definitions
MODETAB
D4A32782 MODENT...
VTAMLST APPL Statements
ACICS1A APPL ...
WSIMAPL2 APPL EAS=1, SESSLIM=YES ...
WSIMAPL1 APPL EAS=1, SESSLIM=YES ...

Figure 10. Coordinating CICS, VTAM, and WSim definitions

Note: If you use the CICS autoinstall feature, a problem can occur caused by the APPL statement in the VTAMLIST not specifying the MODETAB that CICS expects. To remedy the situation, specify the LU2 MODETAB that CICS expects on the APPL statement in the VTAMLIST for the APPL statement.

For example, in the VTAMLIST for WSim, specify the following codes:

```
xxxxyyyy  APPL DLOGMOD=D4A32782,MODETAB=...
```

In the VTAMAPPL statement, specify the following codes:

```
        VTAMAPPL APPLID=xxxxyyyy
```

xxxxyyyy must conform to the naming conventions defined in the CICS autoinstall module. If you use your own autoinstall program, *xxxxyyyy* must conform to the naming conventions your autoinstall program expects.

Also, if you are using the CICS autoinstall feature, see the appropriate CICS resource definition book for your system for information on standard autoinstall models.

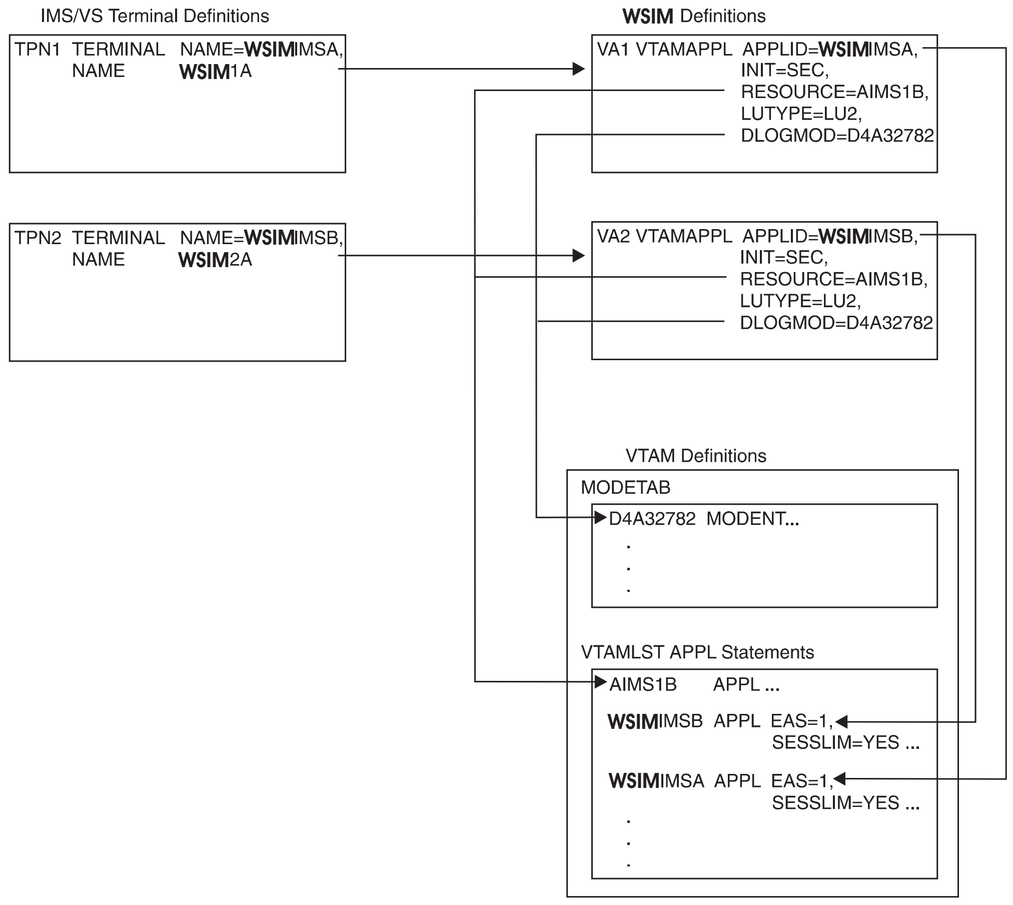


Figure 11. Coordinating IMS/VS, VTAM, and WSim definitions

VTAM application network definition example

The following example shows a complete network definition for four VTAM applications that use all of the session setups and session types available to communicate with each other. This network demonstrates the ability of VTAM applications to communicate with each other as if they were communicating with a logical unit associated with an SNA terminal.

The four VTAM applications in this example are known to VTAM by the resource names APPL1, APPL2, APPL3, and APPL4. These resource names must be used for session initiation. The four VTAM applications are known to WSim as BIG_APPL, VA_APPL2, VA_APPL3, and VA_APPL4. You do not have to use WSim names that are different from those that VTAM knows; you can name the applications APPL1, APPL2, APPL3, and APPL4 for WSim VTAMAPPL definitions as well.

VTAM application BIG_APPL assumes the role of a simple 3270 application program that can communicate with real 3270 LU Type 2 terminals. BIG_APPL also has one secondary LU half-session that simulates a 3270 LU Type 2 to another VTAM application (VA_APPL4).

VTAM applications VA_APPL2 and VA_APPL3 communicate with BIG_APPL and simulate a single 3270 secondary LU Type 2 half-session. This type of VTAM application definition is required to communicate with most VTAM application programs. Like most SNA terminals, these VTAM applications support only one session. Normally, you need to define single-session VTAM applications to communicate with CICS/VS or IMS/VS applications.

VTAM application VA_APPL4 is somewhat like BIG_APPL in reverse. VA_APPL4 simulates two 3270 secondary LU Type 2 half-sessions and communicates with BIG_APPL. VA_APPL4 also looks like a simple 3270 application program to the secondary half-session simulated by BIG_APPL.

Message generation deck SLU_DECK generates standard 3270 LU Type 2 data streams. Message generation deck PLU_DECK processes the 3270 data streams received and generates a response message with the appropriate 3270 command and write control character (WCC) bytes based on the AID received. Refer to Part 2, "Introducing message generation decks," on page 93 for information about the message generation statements included in these message generation decks.

```
VTAM_EX1 NTWRK    MSGTRACE=YES,UTI=100,STLTRACE=YES

PLU_PATH PATH     PLU_DECK           primary message deck
SLU_PATH PATH     SLU_DECK           secondary message deck

*      VTAM Application                          VTAM Application
*    and LU Half-sessions                      and LU Half-sessions
* --------------------------               --------------------------
* BIG_APPL PLUHS_AB_1 LU (pri)  <-session->  VA_APPL2 SLU_HS_A_1 LU (sec)
* BIG_APPL PLUHS_AB_2 LU (pri)  <-session->  VA_APPL3 SLU_HS_B_1 LU (sec)
* BIG_APPL PHUHS_CD_1 LU (pri)  <-session->  VA_APPL4 SLU_HS_C_1 LU (sec)
* BIG_APPL PLUHS_CD_2 LU (pri)  <-session->  VA_APPL4 SLU_HS_D_1 LU (sec)
* BIG_APPL SLUHS_E_1  LU (sec)  <-session->  VA_APPL4 PLUHS_E_1  LU (pri)
*
*     primary sends BIND, secondary receives BIND
*     Parallel sessions exist between BIG_APPL and VA_APPL4.

*---------------------------------------------------------------------
* Define a VTAM application with four primary and one secondary
* LU half-sessions.
*
* APPL1  APPL     AUTH=(ACQ),         minimum VTAM "APPL" required
*                 PARSESS=YES
*

BIG_APPL VTAMAPPL APPLID=APPL1        VTAM APPL = 'APPL1'

PLUHS_AB LU       MAXSESS=(2,0),      two primary half-sessions
                  INIT=SEC,           secondary will initiate session
                  PATH=(PLU_PATH),    primary path
                  DELAY=F0            no delays for primary
PLUHS_CD LU       MAXSESS=(2,0),      two primary half-sessions
                  INIT=PRI,           primary will initiate session
                  PATH=(PLU_PATH),    primary path
                  RESOURCE=APPL4,     generate INIT-SELF to 'APPL4'
                  DLOGMOD=D4A32782,   VTAM logon mode table entry
                  DELAY=F0,           no delays for primary
                  FRSTTXT=HOLDUP      delay before sending INIT-SELF
SLUHS_E  LU       MAXSESS=(0,1),      one secondary half-session
                  LUTYPE=LU2,         LU Type 2 secondary half-session
                  INIT=PRI,           primary will initiate session
                  PATH=(SLU_PATH),    secondary path
                  LOGDSPLY=BOTH,      log display images
                  THKTIME=UNLOCK      start delay after keyboard unlock
*
* Define a VTAM application with one secondary LU half-session.
*
* APPL2  APPL                         minimum VTAM "APPL" required
*
```

```
VA_APPL2 VTAMAPPL APPLID=APPL2         VTAM APPL = 'APPL2'

SLU_HS_A  LU      MAXSESS=(0,1),       one secondary half-session
                  LUTYPE=LU2,          LU Type 2 secondary half-session
                  INIT=SEC,            secondary will initiate session
                  PATH=(SLU_PATH),     secondary path
                  RESOURCE=APPL1,      generate INIT-SELF to 'APPL1'
                  DLOGMOD=D4A32782,    VTAM logon mode table entry
                  LOGDSPLY=BOTH,       log display images
                  THKTIME=UNLOCK       start delay after keyboard unlock
*-----------------------------------------------------------------------
* Define a VTAM application with one secondary LU half-session.
*
* APPL3  APPL                          minimum VTAM "APPL" required
*
VA_APPL3 VTAMAPPL APPLID=APPL3         VTAM APPL = 'APPL3'

SLU_HS_B  LU      MAXSESS=(0,1),       one secondary half-session
                  LUTYPE=LU2,          LU Type 2 secondary half-session
                  INIT=SEC,            secondary will initiate session
                  PATH=(SLU_PATH),     secondary path
                  DLOGMOD=D4A32782,    VTAM logon mode table entry
                  LOGDSPLY=BOTH,       log display images
                  THKTIME=UNLOCK,      start delay after keyboard unlock
                  FRSTTXT=LOGAPPL1     generate INIT-SELF to 'APPL1'
*-----------------------------------------------------------------------
* Define a VTAM application with two secondary and one primary LU
* half-sessions.
*
* APPL4  APPL     AUTH=(ACQ),          minimum VTAM "APPL" required
*                 PARSESS=YES
*

VA_APPL4 VTAMAPPL APPLID=APPL4,        VTAM APPL = 'APPL4'
                  LUTYPE=LU2,          LU Type 2 secondary half-session
                  INIT=PRI,            primary will initiate session
                  PATH=(SLU_PATH),     secondary path
                  LOGDSPLY=BOTH,       log display images
                  THKTIME=UNLOCK       start delay after keyboard unlock

SLU_HS_C  LU      MAXSESS=(0,1)        one secondary half-session
SLU_HS_D  LU      MAXSESS=(0,1)        one secondary half-session
PLUHS_E   LU      MAXSESS=(1,0),       one primary half-session
                  PATH=(PLU_PATH),     primary path
                  RESOURCE=APPL1,      generate INIT-SELF to 'APPL1'
                  DLOGMOD=D4A32782,    VTAM logon mode table entry
                  DELAY=F0             no delays for primary

PLU_DECK MSGTXT
         WTO      (Starting $MSGTXTID$)
         TEXT     ('F5C3'),                    Erase/Write, Unlock Keyboard
                  ($LUID$$SESSNO$ is Active)   startup message

* The following IFs are used by the primary LU half-session to
* 1) ignore SNA responses,
* 2) ignore Data Flow Control (DFC) and Session Control (SC) requests,
* 3) ignore Network Control (NC) requests,
* 4) ignore middle-in-chain and last-in-chain chain elements,
* 5) echo data received when an Enter AID is received,
* 6) generate a time-of-day response message when a PF1 AID is received,
* 7) unlock the keyboard when a Clear AID is received, and
* 8) echo data received when any other AID is received.

1        IF       LOC=RH+0,TEXT='80',THEN=IGNORE,STATUS=HOLD
2        IF       LOC=RH+0,TEXT='40',THEN=IGNORE,STATUS=HOLD
3        IF       LOC=RH+0,TEXT='20',THEN=IGNORE,STATUS=HOLD
```

```
4        IF       LOC=RH+0,TEXT='02',ELSE=IGNORE,STATUS=HOLD
5        IF       LOC=RU+0,TEXT=('7D'),THEN=C-ECHO,STATUS=HOLD,  Enter
                  DATASAVE=(1,B+1,500)                           Save data
6        IF       LOC=RU+0,TEXT=('F1'),THEN=C-TIME,STATUS=HOLD   PF1
7        IF       LOC=RU+0,TEXT=('6D'),THEN=C-UNLOCK,STATUS=HOLD Clear
8        IF       LOC=RU+0,TEXT=('FF'),THEN=C-ECHO,STATUS=HOLD,  Others
                  COND=NE,DATASAVE=(1,B+1,500)                   Save data
TOPLOOP  WAIT
         BRANCH   LABEL=TOPLOOP

UNLOCK   TEXT     ('F1C3')              Write, Unlock Keyboard
         RETURN                         return to caller

TIME     DATASAVE AREA=1,
                  TEXT=($TOD,8$)        save time-of-day
         TEXT     ('F5C3'),             Erase/Write, Unlock Keyboard
                  (The current time-of-day is ),
                  ($RECALL,1+0,2$:$RECALL,1+2,2$:), hh:mm:ss.hh
                  ($RECALL,1+4,2$.$RECALL,1+6,2$)
         RETURN                         return to caller
ECHO     TEXT     ('F1C3'),             Write, Unlock Keyboard
                  ('11'),               Set Buffer Address (SBA) order
                  ($RECALL,1$),         recall cursor address and data
                  ('13')                Insert Cursor (IC) order
         RETURN                         return to caller
         ENDTXT

HOLDUP   MSGTXT
         WTO      (Starting $MSGTXTID$)
         DELAY    TIME=F5               allow VTAMAPPL VA_APPL4 to start
         ENDTXT

LOGAPPL1 MSGTXT
         WTO      (Starting $MSGTXTID$)
         CMND     COMMAND=INIT,         generate INIT-SELF
                  RESOURCE=APPL1,       to 'APPL1'
                  MODE=D4A32782         VTAM logon mode table entry
         DELAY    TIME=F0               start SLU_DECK quickly
         ENDTXT

SLU_DECK MSGTXT
         WTO      (Starting $MSGTXTID$)

* Wait for the primary side of the session to send the first message.

0        IF       LOC=RU+0,TEXT=(PLU_HS),SCAN=YES,
                  THEN=B-STARTNOW
WAITING  WAIT
         BRANCH   LABEL=WAITING         let PLU_HS send first message
STARTNOW DEACT    IFS=(0)

* Loop until WSim is stopped.

TOPLOOP  CLEAR
         TEXT     (Hello There PLU_HS! This is $LUID$$SESSNO$.)
         ENTER
         PF1
         ERIN
         TEXT     (I feel like I am talking to myself. )
         ENTER
         PF24
         WTO      (Loop Completed for $LUID$$SESSNO$ $SEQ,5$)
         BRANCH   LABEL=TOPLOOP
         ENDTXT
```

The following example shows how to code functionally comparable message generation decks that are shown previously using the Structured Translator

Language (STL). This example represents the two separate STL programs that are translated by the STL Translator into the message generation decks. Note that program 1 contains PLU_DECK as well as UNLOCK, TIME, ECHO, and HOLDUP, which are STL procedures that are called by PLU_DECK. For information about STL, see *WSim Script Guide and Reference*.

Program 1

```
@program=trace1x
plu_deck:  msgtxt
/*********************************************************************/
/* PLU_DECK is used by the primary LUs to check messages received    */
/* from the secondary LUs.  The primary LUs take action based on the */
/* attention identifier (AID) byte in the received RUs.              */
/*                                                                   */
/* The device buffer is saved for later use when a message is        */
/* received by the primary LU.                                       */
/*                                                                   */
/* Each time a message is received by the primary LU, WSim checks    */
/* to see if the message is to be ignored.  If it is not to be       */
/* ignored, WSim uses the SELECT group to determine what action      */
/* should be taken based on the first byte in the buffer (the AID    */
/* byte).                                                            */
/*                                                                   */
/* The following messages will be ignored:                           */
/*      First byte of RH='80'x - SNA response                        */
/*      First byte of RH='40'x - Data Flow Control                   */
/*      First byte of RH='20'x - Network Control                     */
/*      First byte of RH='02'x - Chaining indicator                  */
/*********************************************************************/
           say 'Starting' MSGTXTID()
           type 'F5C3'x||LUID()||SESSNO()||' is active'
           onin substr(rh,1,1)='80'x then rsp=on
           onin substr(rh,1,1)='40'x then dfc=on
           onin substr(rh,1,1)='20'x then nc=on
           onin substr(rh,1,1)='02'x then chain=on
           onin then received=on
           onin received=on then data_received=buffer
           do forever
            transmit using enter and wait until onin received=on
            if rsp=off & dfc=off & nc=off & chain=off then
             select
              when substr(data_received,1,1)='F1'X then call time
              when substr(data_received,1,1)='6D'X then call unlock
              otherwise call echo
             end
            rsp=off; dfc=off; nc=off; chain=off; received=off
          end
          endtxt

unlock:   msgtxt
/*********************************************************************/
/* UNLOCK is called by PLU_DECK whenever a '6D'x byte is detected in  */
/* the first byte of the RU received from the secondary LU.  A '6D'x  */
/* byte means that the 'CLEAR' key of the secondary has been pressed  */
/* to clear the screen.  The primary LU sends back a 'F1C3'x, which   */
/* clears the screen and unlocks the keyboard.                        */
/*********************************************************************/
          type 'F1C3'x
          return
          endtxt

time:     msgtxt
/*********************************************************************/
/* TIME is called by PLU_DECK whenever a 'F1'x byte is detected in the*/
/* first byte of the RU returned from the secondary LU.  A 'F1'x byte */
/* means that the PF01 key of the secondary LU has been pressed to    */
/* request the time of day.  The TOD(6) function returns the time     */
/* of day in the format HHMMSS.  TIME puts the time of day into the   */
/* format HH:MM:SS.  The time is inserted in a time                   */
/* message and sent back to the secondary LU.                         */
/**********************************************************************/
          time_of_day=substr(tod(6),1,2)||':'||,
                      substr(tod(6),3,2)||':'||,
                      substr(tod(6),5,2)
          type 'F1C3'x||'The current time-of-day is '||time_of_day
          return
          endtxt

echo:     msgtxt
/**********************************************************************/
/* ECHO is called by PLU_DECK each time the primary LU receives a     */
/* message it is not to ignore (no "ignore" switch is set)            */
/* and the AID byte associated with the message is something other    */
/* than '6d'x (CLEAR) or 'F1'x (PF01).  This procedure extracts       */
/* everything in the device buffer from the second byte on and        */
/* sends it back to the secondary LU.                                 */
/**********************************************************************/
          type 'F1C311'x||substr(data_received,2)||'13'x
          return
          endtxt

holdup:   msgtxt
/**********************************************************************/
/* HOLDUP is used by the primary LUs to ensure that the secondary     */
/* LUs have a chance to get started before WSim begins sending        */
/* messages.  It is a simple 5-second delay.                          */
/**********************************************************************/
          say 'Starting' MSGTXTID()
          suspend(5)
          endtxt
```

Program 2

```
@program=trace2x
logappl1: msgtxt
/**********************************************************************/
/* LOGAPPL1 is used as a FRSTTXT deck by some of the secondary LUs.   */
/* It sends an initiate-self RU to the primary LU to establish a      */
/* session.                                                           */
/**********************************************************************/
          say 'Starting' MSGTXTID()
          initself('APPL1','D4A32782')
          endtxt

slu_deck:  msgtxt
/**********************************************************************/
/* SLU_DECK is used by the secondary LUs.  It waits in a DO WHILE     */
/* loop until 'PLUHS' appears in the buffer, indicating the primary   */
/* LU has contacted the secondary LU.                                 */
/*                                                                    */
/* The DO FOREVER loop that follows sends a CLEAR, types and enters   */
/* a message, sends a PF1, erases to end-of-input, and types and      */
/* sends another message with a PF24.  Since THKTIME=UNLOCK is        */
/* specified in the network definitions, the next message in this     */
/* series is not sent until the primary LU returns a keyboard unlock. */
/*                                                                    */
/* A counter is incremented by one and a message is written to the    */
/* operator console indicating that the loop has been successfully    */
/* completed the number of times indicated by the counter.            */
/**********************************************************************/
           count=0
           say 'Starting' MSGTXTID()
           do while index(buffer,'PLUHS')=0
            suspend(2)
           end
           do forever
```

```
 transmit using clear
 type 'Hello There PLUHS! This is '||LUID()||SESSNO()||'.'
 transmit using enter
 transmit using pf1
 erin
 type 'I feel as if I am talking to myself.'
 transmit using pf24
 count=count+1
 say 'Loop completed for '||LUID()||SESSNO()||'-'||char(count)
end
endtxt
```

How VTAM application sessions are initiated and terminated

This section discusses how sessions are initiated and terminated by the simulated logical unit and by the VTAM logical unit.

Session initiation and termination by WSim

WSim performs session initiation and termination by generating the Formatted System Services (FSS) NS RUs, INIT-SELF, and TERM-SELF.

WSim converts the INIT-SELF RU into a VTAM REQSESS or SIMLOGON request to issue to VTAM. If VTAM can initiate the session, it passes a BIND or CINIT RU to the WSim VTAM application and the SNA session is initiated. If VTAM cannot initiate the session, the REQSESS or SIMLOGON request completes unsuccessfully and VTAM rejects the INIT-SELF RU with an appropriate SNA sense code. If the rejected INIT-SELF RU was generated automatically by WSim, it is generated again after a 30-second delay.

For a primary logical unit half-session, WSim converts the outgoing TERM-SELF RU into a CTERM RU. This CTERM RU causes WSim to generate an UNBIND that terminates the session with the secondary logical unit.

For a secondary logical unit half-session, WSim converts the outgoing TERM-SELF RU into a VTAM TERMSESS request to issue to VTAM. The VTAM TERMSESS requests VTAM to unconditionally UNBIND the session. The WSim VTAM application receives the UNBIND request to terminate the session.

The WSim/VTAM subtask interface layer rejects FSS RUs other than INIT-SELF and TERM-SELF and all unformatted system services (USS) RUs.

Session initiation and termination by VTAM

When a simulated VTAM application receives an unsolicited BIND request from VTAM, the BIND is passed to the first available secondary LU half-session (LU statement) to initiate a session. If no secondary LU half-sessions are available, the BIND is queued for later use. A secondary LU half-session is available under the following conditions:

1. An automatic INIT-SELF is not generated.
2. The RESOURCE operand value, if coded, matches the name of the VTAM resource sending the BIND.
3. A session is not already active.

If a secondary LU half-session receives an UNBIND, the LU is assigned a queued BIND without going through this availability test.

When a simulated VTAM application receives an unsolicited CINIT (logon request) from VTAM, the CINIT is passed to the first available primary LU half-session that sends a BIND to initiate the session. The primary LU half-session is available under the same conditions that are previously described for a secondary LU half-session. If no primary LU half-sessions are available, the CINIT is rejected.

Additional VTAMAPPL considerations

When doing a VTAMAPPL LU 6.2 simulation, you need to take into account a few more considerations. The considerations are listed here in the order that you need to do them.

As for how to tie everything together for VTAMAPPL type of networks, see the rest of this chapter.

1. Get a trace of a real device that you want to simulate. Use this later to help set up your VTAM MODEENT and understand what to expect from the simulation. You also need to know how many parallel session you will be using. Make sure to include the BIND in this trace.
2. Set up a network by using as an example either the network that is shown below or the one in Part 5, “Samples,” on page 277.
3. Generate a script using ITPVTBRF and ITPSGEN from a GTF or NPM trace of the actual device going to the application you want to drive. ITPVTBRF needs the BIND to be included in the trace.
4. Ensure that the BIND image bytes 23 and 24 for the simulated device's sessions have the same settings as the management session's (that is, the CNOS session's) using SNASVCMG.
5. If WSim is simulating the primary LU, then the BIND is controlled with the MODEENT in the mode table associated with the secondary LU (usually the application program or subsystem). Ensure that this MODEENT has all the values specified that you want WSim to send in the BIND. The BIND image bytes 23 and 24 are controlled with the PSERVIC operand in the MODEENT.
6. If you use CICS, make sure that WSim's APPLIDs are defined under CICS's TCT or Auto-install Exit as a valid workstation.
7. You can only do an INIT with VTAMAPPL simulations.
8. To set the BIND USER DATA structured subfields, code them on the DATA=(...) operand on the CMND COMMAND=INIT line. The network name and LU name that you are simulating (which is not necessarily the same one that you captured) need to be in the BIND USER DATA (this network name and LU name is the Network ID of the real network that you are executing in and the APPLID from the VTAMAPPL statement in your network). This is commonly called the LU name in a real network. To avoid confusion, code the LU name in the network with the same value as the APPLID on the VTAMAPPL above it. The names are in USER DATA subfield X'04'.
9. In your generated scripts, make sure to hold off your real application sessions until after your CNOS session is resolved.

Here is a sample VTAMLST definition for the Application Program Minor Node:

```
GM3Z1001  APPL AUTH=(ACQ),PARSESS=YES
```

Here is a sample “model” network. Use this with script generation. Make sure to specify the NETWORK control command in the SYSIN for ITPSGEN. This creates an updated model network to run with the generated scripts.

Your APPLID and LU name do not have to be the same, but it should be. Also, LU name is used by the script generator to identify the sessions for which you are interested in generating a script. Later, when you run your script with this network, it uses the APPLID.

```
YOURNET  NTWRK    INIT=PRI
0        PATH     WAITDECK
VTAMA    VTAMAPPL APPLID=YOURLU
YOURLU   LU       LUTYPE=LU6
```

This is what an updated model network looks like (from ITPSGEN):

```
YOURNET  NTWRK    INIT=PRI
YOURLU   PATH     YOURLU
@DK00000 PATH     @DK00000
@DK00001 PATH     @DK00001
0        PATH     WAITDECK
VTAMA    VTAMAPPL APPLID=YOURLU
YOURLU   LU       PATH=(YOURLU),MAXSESS=(1,0),
                  LUTYPE=LU6
YOURLU   LU       PATH=(@DK00000),MAXSESS=(1,0),
                  LUTYPE=LU6
YOURLU   LU       PATH=(@DK00001),MAXSESS=(0,1),
                  LUTYPE=LU6
```

Here is a sample VTAMAPPL LU6.2 network and generated decks:

```
YOURNET  NTWRK    INIT=PRI
YOURLU   PATH     YOURLU
@DK00000 PATH     @DK00000
@DK00001 PATH     @DK00001
0        PATH     WAITDECK
VTAMA    VTAMAPPL APPLID=GM3Z1001
YOURLU   LU       PATH=(YOURLU),LUTYPE=LU6,MAXSESS=(1,0)    primary CNOS
YOURLU   LU       PATH=(@DK00000),LUTYPE=LU6,MAXSESS=(1,0)  primary
YOURLU   LU       PATH=(@DK00001),LUTYPE=LU6,MAXSESS=(0,1)  secondary

* This deck is for the CNOS - PLU session.
YOURLU   MSGTXT
*       The following INIT is generated with the USER DATA all in hex.
*       I translated some of it so that those part would be easy to find.
         CMND     COMMAND=INIT,MODE=SNASVCMG,RESOURCE=EBAACIDC,
*                 DATA=('000902E2D5C1E2E5C3D4C7090301595B8D60595B8D12'),
                  DATA=('000902'SNASVCMG'090301595B8D60595B8D12'),
                  ('04'EB0ZNET0.GM3Z1001)
         RH       DR1=ON,DR2=OFF,EXC=OFF,CDI=OFF
         DELAY    TIME=F0000008
*       This is the CNOS flow.
         TEXT     ('310502FF0003D000000206'1'001A11'EB0ZNET0.GM3Z1001x),
                  ('FE102B593D00010842EE3F'r))'59'$$'8D00191210020000'),
                  ('000000040004000000008'WSIMLU62)
         RH       FMI=ON,CDI=ON
*       Do not proceed until the CNOS flow has been resolved:
5        IF       LOC=RU+2,TEXT=('1210'),THEN=CONT
         WAIT
*       After the CNOS flow add the following lines:
         DATASAVE AREA=1,TEXT=($LUID$)
         EVENT    POST=1+0
*       The above two lines tells the other sessions for this VTAMAPPL
*       that the CNOS is resolved and that they may continue.
WAIT     WAIT
         BRANCH   LABEL=WAIT
         ENDTXT

* This deck is for the application-data PLU session.
@DK00000 MSGTXT
*       This waits for the CNOS session to be resolved.
```

```
         DATASAVE AREA=1,TEXT=($LUID$)
         WAIT     EVENT=1+0
*       This INIT is similar to the CNOS's but it can use a
*       different MODEENT, as long as the BIND image of this session
*       has the same values in bytes 23-24 as the CNOS's BIND image.
         CMND     COMMAND=INIT,MODE=WSIMLU62,RESOURCE=EBAACIDC,
                  DATA=('000902'WSIMLU62'090301595B8D75595B8D12'),
                  ('04'EB0ZNET0.GM3Z1001)
                  .
                  .
                  .
                  the rest of the generated script
                  .
                  .
                  .
         ENDTXT

* This deck is for the application-data SLU session.
@DK00001 MSGTXT
*       This waits for the CNOS session to be resolved.
         DATASAVE AREA=1,TEXT=($LUID$)
         WAIT     EVENT=1+0
                  .
                  .
                  .
                  the rest of the generated script
                  .
                  .
                  .
         ENDTXT
```

Chapter 4. Simulating CPI-C transaction programs

This chapter discusses how to use WSim to simulate Common Programming Interface Communications (CPI-C) transaction programs. You can use CPI-C transaction program simulations to test applications, or to do stress or performance testing. You can test existing applications or applications that you are currently developing. Any real applications that are part of a CPI-C transaction program simulation must be APPC applications (that is, they must use LU 6.2 communication protocols), but they need not use the CPI-C application program interface (API).

WSim provides support for CPI-C Version 1 Release 1. This level of CPI-C architecture is described in *SAA Common Programming Interface Communications Reference* (SC26-4399-06). Your WSim scripts that define CPI-C transaction programs must conform to this level of architecture.

WSim support for CPI-C transaction program simulations

WSim can simulate transaction programs that are clients, servers, or both. WSim determines if a transaction program is a client or server based on the program's role in a given conversation. A client transaction program is a program that allocates an outbound conversation, and a server transaction program is a program that accepts an inbound conversation. A transaction program may have both inbound and outbound conversations active at the same time, and act as a client for some conversations and a server for others. You can use CPI-C transaction program simulations in the following ways:

- to simulate a client communicating with a real server
- to simulate a server communicating with a real client
- to simulate a client communicating with a simulated server

WSim can simulate multiple clients and servers in a network, and each transaction program can have multiple conversations active at any given time. However, a server transaction program can only accept one inbound conversation.

WSim can simulate multiple instances of a given transaction program. A transaction program instance is a copy of the transaction program running on a given LU. You can simulate up to 32,767 concurrently active instances of a transaction program. The instances are identified by an instance number from 1 to 99,999. The instance number rolls over at 99,999.

One or more message deck paths define the transaction program to WSim. The message deck specified as FRSTTXT, or the first deck in the first path defining the transaction program, receives control when a new transaction program instance is activated. WSim activates transaction program instances in these ways:

- Initial instances are activated when the network is started. If the specified maximum concurrent instances is achieved before all initial instances are started, start-up for the remaining initial instances is delayed.
- When an instance completes, if all initial instances have not been activated, another instance is activated.
- When an attach request is received for a transaction program, WSim activates a new instance under the following conditions:

- No instance of the transaction program is active.
- Instances of the transaction program are active, and the following conditions are met:
  - All of the active instances have already accepted an inbound conversation.
  - The number of active instances is less than the maximum concurrent instances specified for the transaction program.

Normally the transaction program terminates when WSim reaches the end of the message deck path defining the transaction program. However, if the transaction program is defined as repeating, WSim repeats the message deck path. WSim continuously repeats the message deck path of a repeating transaction program until a message deck construct or operator command stops execution, or the simulation ends.

The CPITRACE operand of the TP network definition controls the logging of CPI-C transaction program trace information. WSim provides the following options:
- WSim logs each CPI-C verb and its parameters when it is issued and when it completes.
- WSim logs each CPI-C verb only when it completes.
- WSim logs messages that trace the issuance and completion of CPI-C verbs.
- WSim does not log any CPI-C trace information.

Specify the CTRC control command to request that CPI-C trace information be printed when running the Loglist Utility. WSim logs CPI-C attach requests and send and receive data as XMIT and RECV records. The MLOG network definition operand controls the logging of these records. Refer to Chapter 26, “Loglist examples,” on page 375 for sample loglist output of a CPI-C transaction program simulation.

Defining CPI-C simulation resources

This section describes the network definition statements you can use to define CPI-C simulation resources. It also discusses coding considerations you must be aware of when defining these resources.

Network definition statements for CPI-C resources

To simulate CPI-C transaction programs, make the following additions to your network definition:
- Specify one or more message deck paths to represent the transaction program.
- Use one or more APPCLU statements to specify a logical unit type 6.2 through which the transaction program accesses the network.
- Use one or more TP statements to specify each transaction program that resides on a given logical unit. Include in these statement parameters to specify the number of transaction program instances to beactivated initially by WSim, and the maximum number of instances that can be active concurrently.
- Optionally, use one of the following methods to specify CPI-C side information. This information associates a symbolic destination name with the partner logical unit, the name of the partner transaction program, and the mode to be used for conversations with that logical unit and transaction program combination:
  - Include a SIDEINFO operand on the APPCLU statement. The definitions supplied by this operand can be used by all transaction programs residing on this logical unit.

– Specify a SIDEINFO network definition table by using the SIDEINFO, SIDEENT, and SIDEEND statements. This table defines network-wide side information available for use by all transaction programs in the network.

Note: If symbolic destination names are not defined in a side information table, the destination information must be specified in the script using CPI-C statements.

The network definition statements used to simulate CPI-C transaction programs are described as follows. Refer to *WSim Script Guide and Reference* for complete statement definitions.

SIDEINFO Statement Group

Use the SIDEINFO statement group to define a network-wide CPI-C side information table. The side information table defines symbolic destination namesused to refer to an LU name, mode name, and TP name triplet. The SIDEINFO group is defined as follows:

```
SIDEINFO
SIDEENT  DESTNAME=...
  .
  .
SIDEENT
SIDEEND
```

Specify one SIDEENT statement for each symbolic destination name to be defined. A network can contain only one SIDEINFO statement group.

APPCLU Statement

Use the APPCLU statement to define a CPI-C APPC logical unit. The CPI-C APPC logical unit is implemented as a VTAM application program using APPC (LU 6.2) communication protocols. You can use VTAM-unique operands on the APPCLU statement to build a VTAM access method control block (ACB). You can define any number of APPCLUs in a network definition. The following operands on the APPCLU statement specify characteristics of the simulated VTAM application program:

- APPLID defines the symbolic name of the VTAM application program. This name must match an entry in VTAM's configuration tables (VTAMLST) created using a VTAM APPL definition statement. The name specified is the name of the APPL statement or ACBNAME operand value coded on an APPL statement. The corresponding VTAM APPL definition statement must specify APPC=YES. The APPLID name must be unique within all APPC logical units defined by the simulation. If the APPLID operand is not specified, it defaults to the name of the APPCLU statement.
- PASSWD specifies the password associated with the application program. This password must be the same as the PRTCT operand value coded on VTAM's APPL definition statement.

In addition, you can specify the following optional operands on the APPCLU statement to supply information used when establishing LU 6.2 sessions and CPI-C conversations:

- CNOS specifies session limits for the LU 6.2 sessions established between this logical unit and other logical units defined in your network.

- SIDEINFO defines symbolic destination names available for use by transaction programs on this logical unit when initializing conversations. These definitions are global to all transaction programs residing on this logical unit.

TP Statement

Use the TP statement to define CPI-C transaction programs to WSim. For each transaction program residing on a given logical unit, you must provide a TP statement following the APPCLU statement that defines the CPI-C logical unit. You must provide at least one TP statement after each APPCLU statement. You can specify any number of TP statements following an APPCLU statement. You can specify the same TP name after multiple APPCLU statements (the associated path list need not be the same). The name of the TP statement identifies the resource to WSim. The following operands on the TP statement specify various characteristics of the simulated transaction program:

- CPITRACE specifies the level of CPI-C tracing to be performed.
- FRSTTXT specifies the first message generation deck to be used when the transaction program is started.
- INSTANCE specifies the number of instances of the transaction program to be activated by WSim when the network is started, and the maximum number of concurrent instances that are supported.
- PATH specifies the PATH statements for message generation deck selection this transaction program references in controlling message generation.
- TPNAME specifies the name of the transaction program to be simulated on this logical unit.
- TPREPEAT specifies whether message generation should repeat the paths defined for the transaction program, or whether message generation should end for the transaction program when the end of the path sequence is reached.
- TPSTATS specifies whether WSim should keep message sent and received statistics for each individual transaction program instance.
- TPSTIME specifies the stagger time to be used by WSim in initiating multiple transaction program instances at network start-up.
- TPTYPE specifies whether the transaction program is a clientor a server.
- UCD specifies whether the transaction program is to recognize user control data and treat it as if it were application data.

Figure 12 on page 39 shows the logical configuration of a CPI-C network in which WSim simulates two APPC logical units, each with two CPI-C transaction programs.

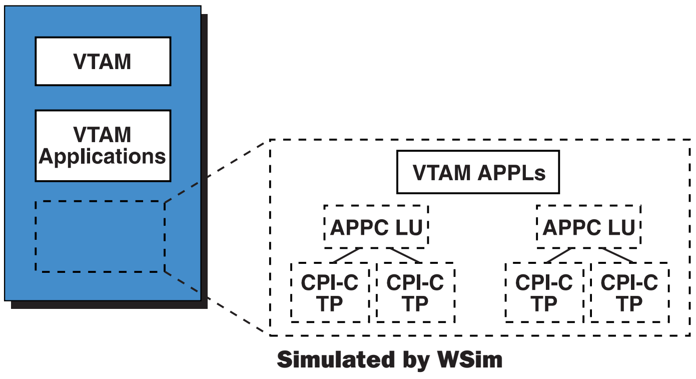


Figure 12. CPI-C network simulating LUs and TPs

Figure 13 shows the network definition for this CPI-C network. In this example, VTAM knows the APPC logical units as APPCLU1 and APPCLU2. WSim knows the transaction programs for APPCLU1 as LU1TP1 and LU1TP2, and the transaction programs for APPCLU2 as LU2TP1 and LU2TP2.

```
CPIC     NTWRK

CLIENT1  PATH     CLIENT1
SERVER1  PATH     SERVER1
CLIENT2  PATH     CLIENT2
SERVER2  PATH     SERVER2

APPCLU1  APPCLU
LU1TP1   TP       TPTYPE=CLIENT,PATH=(CLIENT1),INSTANCE=(1,1)
LU1TP2   TP       TPTYPE=SERVER,PATH=(SERVER1),INSTANCE=(0,1)

APPCLU2  APPCLU
LU2TP1   TP       TPTYPE=CLIENT,PATH=(CLIENT2),INSTANCE=(1,1)
LU2TP2   TP       TPTYPE=SERVER,PATH=(SERVER2),INSTANCE=(0,1)
```

Figure 13. CPI-C network definition

Designing your CPI-C transaction program simulations

When you design your CPI-C transaction program simulation, you must consider each of the following areas:

- Network definition
- Scripting
- CPI-C architecture
- VTAM definitions.

Here are some considerations you should be aware of when coding CPI-C simulations.

Network definition considerations

- Each APPCLU statement in a simulation must have a unique APPLID name (either coded of defaulted). This name must match an entry in VTAM's configuration tables (VTAMLST) you created by using a VTAM APPL definition statement. The name must match the name of the APPL statement or the ACBNAME operand value coded on the APPL statement. If multiple simulations are running on the same VTAM, the APPLID names must be unique across all simulations. This is necessary because VTAM associates a conversation to an APPLID name and receives requests by this name, rather than by the LU or TP name. In addition, for each APPC logical unit referenced by a CPI-C network, you must specify APPC=YES on the APPL statement in the VTAMLST dataset.

- There is no concept of CNOS at the user interface level of CPI-C. However, WSim provides a CNOS interface to allow the user to have control over the number of sessions established between pairs of logical units.
- When you are simulating both the client and server logical units, client logical units may come active before server logical units. This may cause CNOS failures or allocate failures or both. To prevent this, your simulation must allow server logical units to complete startup before CNOS processing or conversation allocation takes place. Do one of the following actions to make this happens:
  - Specify a nonzero **user time interval** (UTI) in your simulation, and issue a small delay before the first CMALLC verb is issued by a client transaction program. You might need to experiment with the delay value.
  - In the network definition, define server-only logical units first, and define server transaction programs before client transaction programs on mixed logical units.
- WSim supports 17-byte partner LU names and 64-byte transaction program names. However, WSim limits all names externalized on reports to 8 bytes. The names on the reports are the network definition name fields specified on the APPCLU and TP statements respectively.
- Exercise caution in defining CPI-C networks when you use the DIST network definition statement, or the CYCLIC operand on the PATH statement. Do not use the DIST statement or CYCLIC operand when defining server transaction programs. Keep the following points in mind if you choose to use the DIST statement or the CYCLIC operand when defining client transaction programs:
  - Use one or more PATH statements when you define the CPI-C transaction program to WSim.
  - WSim executes the path or paths that define a transaction program, by default, only one time. When WSim executes the last message deck in the last path, the simulated transaction program terminates. If the scenario requires multiple iterations of the transaction program paths, specify the TPREPEAT operandon the TP statement.
  - If individual message decks represent pieces of a transaction program, you must not use the DIST statement or CYCLIC operand. Use of these options makes sense only if each message deck represents a complete transaction program.
  - If each message deck in a path represents a complete transaction program, you can use the DIST statement to execute different client transaction programs according to the distribution pattern.
  - If each message deck in a path represents a complete transaction program, you can use the CYCLIC operandto cycle through each of the client TPs specified in the PATH statement.

Scripting considerations

- WSim does not provide an automatic script generation facility for CPI-C transaction programs. CPI-C scripts must be coded manually in either WSim scripting language or in STL. It is recommended that STL be used. In STL, the CPI-C parameters and values are predefined STL variables, making the coding of CPI-C statements easier.

- If you use STL to code the scripts, and you code CPI-C input parameters using literal values or named constants, the STL translator generates device save area or device counter references, or both, for the literal or constant values. The STL translator requires a free device save area if you use string literals or named constants, and a free device counter if you use integer literals or named constants. If a free save area or counter is not available when required, the STL translator issues error message ITP3027I or ITP3028I.
- Using network save areas, and network, line, or term counters allows sharing of data across transaction programs. When sharing data, one transaction program can change data another transaction program is using. This causes unexpected results. Before changing a shared save area or counter, investigate the implications to other transaction programs that may be sharing the same data. Because timing of transaction program execution is difficult to predict, the point in time a particular transaction program will be finished using a particular data item may be unclear. If you are getting unexpected results when using shared data, make sure one transaction program is not changing the data while another one is still using it.
- When a transaction program terminates, all signal event actions, ON conditions, and waits established by the transaction program are canceled. In addition, all signal event actions, ON conditions, and waits that have the terminating transaction program as an object are canceled.
- All general definition message generation statements are valid for CPI-C simulations, except for the TEXT statement and input and output IF statements. WSim ignores TEXT statements and input and output IF statements that are specified in a CPI-C simulation. In addition, WSim ignores message generation statements for SNA and special device support if they are specified in a CPI-C simulation.
- All STL statements are valid for CPI-C simulations except the TYPE, TRANSMIT, ONIN, and ONOUT statements. WSim ignores TYPE, ONIN, and ONOUT statements if they are specified in a CPI-C simulation. Do not code the TRANSMIT statement in a CPI-C simulation because it causes message generation to be interrupted for the transaction program.

CPI-C architecture considerations

WSim differs from the CPI-C 1.1 architecture in the following ways:

- The maximum send length you can specify on mapped conversations is 32K-4 (32,763) instead of the 32,767 value specified by the architecture.
- WSim does not support the CM_SYNC_POINT conversation synchronization level.

VTAM APPL coding considerations

Each simulated APPC logical unit defined in a network requires an active VTAM APPL when WSim opens the ACB associated with the simulated APPC logical unit. The VTAM APPL must not be in use by any other process when you start the network. You define the VTAM APPL definition statement in the VTAMLST dataset. Use operands on the VTAM APPL definition statement to specify the name of the APPL and the authority of the APPL to use certain VTAM functions. You must specify APPC=YES on the APPL statement.

A sample VTAM APPL definition statement for a CPI-C APPC LU is:

```
APPCLU1  APPL APPC=YES,EAS=29
```

Notes:

- For VTAM application definitions used to simulate CPI-C transaction programs, do not specify SESSLIM=YES on the VTAMLST APPL statements.
- The EAS operand (estimated number of concurrent sessions) should be specified with the lowest practical value in order to hold down the amount of CSA storage VTAM allocates for the APPL. Values from EAS=1 to EAS=29 will allocate the least amount of CSA storage.

Coordinating WSim, VTAM, and subsystem definitions

Because CPI-C transaction program simulations use simulated VTAM applications, the coordination issues relevant to VTAM application simulations are also relevant to CPI-C application simulations. You can find a discussion of these issues in "Coordinating WSim, VTAM, and subsystem definitions" on page 23.

WSim support for CPI-C simulation requires VTAM Version 3 Release 2 or later. The WSim CPI-C interface uses the VTAM LU 6.2 command interface (APPCCMD) that was initially released in Version 3 Release 2.

To get the best performance on MVS, make WSim APF-authorized.

CPI-C transaction program network definition example

Figure 14 shows a complete network definition for a CPI-C transaction program simulation. The figure includes a sample script coded in STL. Figure 15 on page 46 shows a functionally equivalent script coded in WSim scripting language. When you use the WSim ISPF Interface to edit a new script, these sample scripts are included in the list of model scripts provided.

Note: When you write your simulation script in STL, you can find the STL definitions for the CPI-C statement parameters and their constant values in the CPICVAR and CPICCON *include* members. These *include* members are described in *WSim Script Guide and Reference* as STL Variable Declarations for CPI-C Verb Parameters.

Coding CPI-C network definition and STL statements

Figure 14. Network and STL definition for CPI-C TP simulation

```
/* CPI-C Transaction Program simulation                             */
@NETWORK
*********************************************************************
* Network Configuration:  CPI-C Transaction Program simulation (CPIC)  *
*                                                                   *
* Description:  This WSim script will simulate a CPI-C client       *
*               Transaction Program communicating with two CPI-C    *
*               Server Transaction Programs.  The client sends      *
*               data to one server and receives data from the other. *
*               The sync-level is "none" on the first conversation, *
*               and "confirm" on the second conversation.           *
*                                                                   *
*               To illustrate that a network-wide Side Information  *
*               Table can be overridden at the APPCLU level, the    *
*               network-wide table contains an entry that points    *
*               to a nonexistent TP.  This entry is then overridden *
*               by the APPCLU statement.                            *
*                                                                   *
*               Some values may need to be changed in this network in *
*               order to operate in your environment.  They are     *
*               indicated by the "==> " string.                     *
*                                                                    *
*                                                                    *
* Restrictions/Dependencies:                                         *
*  1) The APPLID names used in this network (APPLID1 and APPLID2)    *
*     must be defined to VTAM and must be active.                    *
*                                                                    *
* Graphical Representation of Network:                               *
*                                                                    *
*   APPCLU: APPLID1                          APPCLU: APPLID2         *
*   +----------------------+                 +---------------------+ *
*   |                      |                 |                     | *
*   |   +--------------+   |  conversation 1 |   +-------------+   | *
*   |   |              ==========================> TP: TPSERV1 |   | *
*   |   |              |   |   mode=#inter   |   +-------------+   | *
*   |   | TP: TPCLIENT |   |                 |                     | *
*   |   |              |   |  conversation 2 |   +-------------+   | *
*   |   |              ==========================> TP: TPSERV2 |   | *
*   |   +--------------+   |   mode=#batch   |   +-------------+   | *
*   |                      |                 |                     | *
*   +----------------------+                 +---------------------+ *
*                                                                    *
* Notes:                                                             *
*  1. Conversation 1 uses mode name #INTER.  The conversation        *
*     sync-level is "none".                                          *
*  2. Conversation 2 uses mode name #BATCH.  The conversation        *
*     sync-level is "confirm".                                       *
*  3. The CNOS operand on the APPCLU1 definition is only required    *
*     if you want to control the number of sessions.  If the operand *
*     is not specified, sessions will be managed by WSim as required *
*     by the simulation.                                             *
*                                                                    *
*                                                                    *
**********************************************************************
*--------------------------------------------------------------------*
* Network statement operands.                                        *
*--------------------------------------------------------------------*
CPIC     NTWRK    HEAD='CPI-C NETWORK MODEL',
                  ITIME=1,           * Ntwrk interval rpt every minute *

*--------------------------------------------------------------------*
* TP operands coded on the network statement.  These values will     *
* be the default for every TP in the network.                        *
*--------------------------------------------------------------------*
                  TPSTATS=YES,       * Keep stats for all TP instances *
                  CPITRACE=VERB      * Trace CPI-C verbs in the log    *

*--------------------------------------------------------------------*
* Define a network-wide Side Information Table.                      *
*--------------------------------------------------------------------*
         SIDEINFO
         SIDEENT  DESTNAME=SERVER1,MODENAME=#INTER,
                  LUNAME=APPLID2,TPNAME=TPSERVER
         SIDEENT  DESTNAME=SERVER2,MODENAME=#BATCH,
                  LUNAME=APPLID2,TPNAME=TPSERV2
         SIDEEND
*--------------------------------------------------------------------*
* Define the Transaction Program paths.                              *
*--------------------------------------------------------------------*
CLIENT   PATH     CLNTDCK            * Define the CLIENT TP path       *
SERVER1  PATH     SERV1DCK           * Define the SERVER1 TP path      *
SERVER2  PATH     SERV2DCK           * Define the SERVER2 TP path      *
*--------------------------------------------------------------------*
*--------------------------------------------------------------------*
* Define the network resources.                                      *
*                                                                    *
* ==> CHANGE the APPLID names APPLID1 and APPLID2 as needed to match *
*     names in your environment.  If you change APPLID2, also change *
*     the LUNAME specification in the SIDEINFO and CNOS operands to    *
*     match this name.  These names must be defined to VTAM.           *
*----------------------------------------------------------------------*
APPCLU1  APPCLU APPLID=APPLID1,      * APPC LU; VTAM APPLID is APPLID1 *
         SIDEINFO=((DESTNAME=SERVER1,MODENAME=#INTER,
                    LUNAME=APPLID2,TPNAME=TPSERV1)),
*                                    * Override SERVER1 dest name      *
         CNOS=((LUNAME=APPLID2,MODENAME=#INTER,
                SESSIONS=2,CWL=1,CWP=1))
*                                    * Specify CNOS values             *
TPC      TP TPNAME=TPCLIENT,         * TP name is TPCLIENT             *
            PATH=(CLIENT),           * TP is defined by CLIENT path    *
            TPTYPE=CLIENT,           * TP type is CLIENT               *
            INSTANCE=(1,1)           * 1 initial TP instance           *
APPCLU2  APPCLU APPLID=APPLID2       * APPC LU; VTAM APPLID is APPLID2 *
TPS1     TP TPNAME=TPSERV1,          * TP name is TPSERV1              *
            PATH=(SERVER1),          * TP is defined by SERVER1 path   *
            TPTYPE=SERVER,           * TP type is SERVER               *
            INSTANCE=(0,1)           * No initial TP instances         *
TPS2     TP TPNAME=TPSERV2,          * TP name is TPSERV2              *
            PATH=(SERVER2),          * TP is defined by SERVER2 path   *
            TPTYPE=SERVER,           * TP type is SERVER               *
            INSTANCE=(0,1)           * No initial TP instances         *
@ENDNETWORK
@EJECT
@PROGRAM=CPIC
@include cpicvar
@include cpiccon
CLNTDCK: msgtxt
/***********************************************************************
* STL deck defining the TPCLIENT Transaction Program.                  *
***********************************************************************/
say 'Transaction Program 'tpid() 'starting.'
/**********************************************************************/
/* Initialize and allocate a conversation with TPSERV1.               */
/* Set the symbolic destination name to "SERVER1". */
sym_dest_name='SERVER1'
/* Initialize the conversation. */
CMINIT (conversation_ID, sym_dest_name, return_code)
/* Allocate the conversation; the sync-level defaults to "none", */
/* and the conversation type defaults to "mapped".               */
CMALLC (conversation_ID, return_code)
/* Setup the send buffer and length. */
send_buffer = 'LU' appcluid()', TP' tpid()tpinstno()||,
             ': Data sent from client to server.'
send_length = length(send_buffer)
/* Send data to TPSERV1. */
CMSEND (conversation_ID, send_buffer, send_length,,
        request_to_send_received, return_code)
/* Deallocate the conversation with TPSERV1. */
CMDEAL (conversation_ID, return_code)
/**********************************************************************/
/* Initialize and allocate a conversation with TPSERV2.               */
/* Set the symbolic destination name to "SERVER2". */
sym_dest_name='SERVER2'
/* Initialize the conversation. */
CMINIT (conversation_ID, sym_dest_name, return_code)
/* Set the conversation sync-level to "confirm". */
CMSSL  (conversation_ID, cm_confirm, return_code)
/* Allocate the conversation; the conversation type defaults to  */
/* "mapped".                                                     */
CMALLC (conversation_ID, return_code)
/* Receive data from TPSERV2. */
CMRCV  (conversation_ID, receive_buffer, 100, data_received,,
        received_length, status_received,,
        request_to_send_received, return_code)
```

```
/* Confirm the data was received. */
CMCFMD (conversation_ID, return_code)
/* Receive the confirm deallocate status. */
CMRCV  (conversation_ID, receive_buffer, 100, data_received,,
        received_length, status_received,,
        request_to_send_received, return_code)
/* Confirm the deallocate */
CMCFMD (conversation_ID, return_code)
say 'Transaction Program 'tpid() 'complete.'
say 'Simulation complete.'
endtxt

SERV1DCK: msgtxt
/*********************************************************************
* STL deck defining the TPSERV1 Transaction Program.                *
*********************************************************************/
say 'Transaction Program 'tpid() 'starting.'
/* Accept the conversation with TPCLIENT. */
CMACCP (conversation_ID,,
        return_code)
/* Receive data from TPCLIENT. */
CMRCV  (conversation_ID,,
        receive_buffer,,
        100,,
        data_received,,
        received_length,,
        status_received,,
        request_to_send_received,,
        return_code)
say 'Transaction Program 'tpid() 'complete.'
endtxt
SERV2DCK: msgtxt
/*********************************************************************
* STL deck defining the TPSERV2 Transaction Program.                *
*********************************************************************/
say 'Transaction Program 'tpid() 'starting.'
/* Accept the conversation with TPCLIENT. */
CMACCP (conversation_ID,,
        return_code)
/* Set requested length for receive. */
requested_length=100
/* Receive send status from TPCLIENT. */
CMRCV  (conversation_ID,,
        receive_buffer,,
        requested_length,,
        data_received,,
        received_length,,
        status_received,,
        request_to_send_received,,
        return_code)
/* Setup the send buffer and length. */
send_buffer = 'LU' appcluid()', TP' tpid()tpinstno()||,
              ': Data sent from server to client.'
send_length = length(send_buffer)
/* Send data to TPCLIENT. */
CMSEND (conversation_ID,,
        send_buffer,,
        send_length,,
        request_to_send_received,,
        return_code)

/* Request confirmation that the data was received. */
CMCFM  (conversation_ID,,
        request_to_send_received,,
        return_code)
/* Deallocate the conversation with TPCLIENT. */
```

```
CMDEAL (conversation_ID,,
        return_code)
say 'Transaction Program 'tpid() 'complete.'
endtxt
```

Coding CPI-C message generation decks

Figure 15 shows how you can code message generation decks that are functionally equivalent to the STL decks shown in Figure 14 on page 42.

Figure 15. Message deck definition for CPI-C TP simulation

```
CLNTDCK  MSGTXT
*********************************************************************
* Message deck defining the TPCLIENT Transaction Program.           *
*********************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=destination name
*  3=send buffer
*  4=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=send length
*  dc3=request-to-send received
*  dc4=sync-level
*  dc5=requested length
*  dc6=data received
*  dc7=received length
*  dc8=status received
*
         WTO  (Transaction Program $TPID$ starting.)
*
*********************************************************************
*        Initialize and allocate a conversation with TPSERV1.         *
*
*
*        Set the sumbolic destination name to "SERVER1".
         DATASAVE AREA=2,TEXT=(SERVER1)
*
*        Initialize the conversation.
         CMINIT(1,2,DC1)
*
*        Allocate the conversation; the sync-level defaults to "none",
*        and the conversation type defaults to "mapped".
*
         CMALLC(1,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=3,TEXT=(LU $APPCLUID$, TP $TPID$$TPINSTNO$:)+
                  ( Data sent from client to server.)    * Send buffer
         SET  DC2=LENG(3)                                * Send length
*
*        Send data to TPSERV1.
         CMSEND(1,3,DC2,DC3,DC1)
*
*        Deallocate the conversation with TPSERV1.
         CMDEAL(1,DC1)
*
*********************************************************************
*        Initialize and allocate a conversation with TPSERV2.         *
*
*        Set the symbolic destination name to "SERVER2".
```

```
         DATASAVE AREA=2,TEXT=(SERVER2)
*
*        Initialize the conversation.
         CMINIT(1,2,DC1)
*
*        Set the conversation sync-level to "confirm".
         SET  DC4=1                                      * Sync-level
         CMSSL(1,DC4,DC1)
*
*        Allocate the conversation; the conversation type defaults
*        to "mapped".
         CMALLC(1,DC1)
*
*        Set requested length for receive.
         SET  DC5=100
*
*        Receive data from TPSERV2.
         CMRCV(1,4,DC5,DC6,DC7,DC8,DC3,DC1)
*
*        Confirm the data was received.
         CMCFMD(1,DC1)
*
*        Receive the confirm deallocate status.
         CMRCV(1,4,DC5,DC6,DC7,DC8,DC3,DC1)
*
*        Confirm the deallocate
         CMCFMD(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
         WTO  (Simulation complete.)
*
         ENDTXT
SERV1DCK MSGTXT
************************************************************************
* Message deck defining the TPSERV1 Transaction Program.               *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPCLIENT.
         CMACCP(1,DC1)
*
*        Set requested length for receive.
         SET  DC2=100
*
*        Receive data from TPCLIENT.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT
SERV2DCK MSGTXT
************************************************************************
* Message deck defining the TPSERV2 Transaction Program.               *
************************************************************************
```

```
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*  3=send buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*  dc7=send length
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPCLIENT.
         CMACCP(1,DC1)
*
*        Set requested length for receive.
         SET  DC2=100
*
*        Receive send status from TPCLIENT.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=3,TEXT=(LU $APPCLUID$, TP $TPID$$TPINSTNO$:)+
                  ( Data sent from server to client.)    * Send buffer
*
         SET      DC7=LENG(3)                            * Send length
*
*        Send data to TPCLIENT.
         CMSEND(1,3,DC7,DC6,DC1)
*
*        Request confirmation that the data was received.
         CMCFM(1,DC6,DC1)
*
*        Deallocate the conversation with TPCLIENT.
         CMDEAL(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT
```

Chapter 5. Simulating TCP/IP devices

This chapter discusses how to use WSim to simulate TCP/IP client applications. These applications can represent clients using Telnet 3270, 3270E, 5250, or NVT protocol,File Transfer Protocol (FTP), orSimple TCP or UDP transmit/receive protocol. This chapter:

- Describes how WSim interfaces to the IBM TCP/IP product to provide the TCP/IP support.
- Provides general information that applies to all of the supported protocols.
- Discusses the specific support for each protocol.

Using the TCP/IP connection protocol

WSim provides native support of multiple client applications that run "on top" of TCP/IP; that is, TCP/IP handles all routing and delivery between the WSim host and the system under test. WSim communicates directly to TCP/IP on the local host to establish a connection with the server on the system under test.

When available for the TCP/IP instance, as specified by the TCPIP operand, WSim uses the TCP/IP High Performance Native Sockets API instead of the IUCV API. This improves performance and provide compatibility with future releases of the TCP/IP product.

Simulating TCP/IP clients

You can simulate TCP/IP clients to test application programs and network resources. Because WSim simulates only clients, it always initiates the primary connection for each client. WSim closes the connection whenever the TCP/IP server closes it or when a logical error is detected. For the Simple TCP and UDP protocol, the connection can also be closed at the direction of your script. For Telnet 3270, 3270E, 5250, NVT, and FTP simulations, connections are established before any messages are generated and automatically restarted 30 seconds after they are closed, unless the simulated client was quiesced by your script or by an operator command. For Simple TCP and UDP simulations, connections are established after a message to be transmitted was generated, and reestablished whenever a subsequent message is available and an earlier connection was closed.

You can use the **ALTER** Operator command to change the server address to which the next connection for a simulated client is to be made. You do this by using the **SERVADDR** operand of **ALTER**. You can also use the **QUIESCE** and **RELEASE** functions to inhibit or allow further connections. For example, you might quiesce a simulated client in a script when generating a "logoff" sequence, and then release it later by operator command to allow a reconnection and subsequent "logon".

Defining TCP/IP application configurations

Figure 16 on page 50 illustrates the logical configuration of a network that simulates TCP/IP clients accessing a TCP/IP server. You would use this logical configuration to test client applications, to see how many clients your TCP/IP network can support, or to see the impact of many clients on your TCP/IP network.

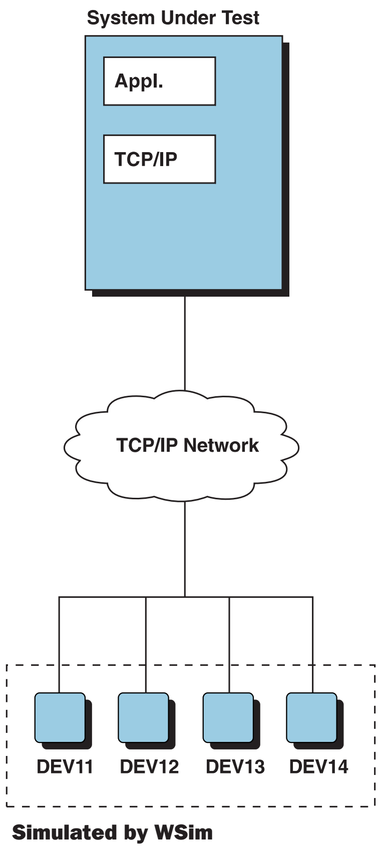


Figure 16. TCP/IP network—logical configuration

A physical configuration that corresponds to this logical configuration is shown in Figure 17.

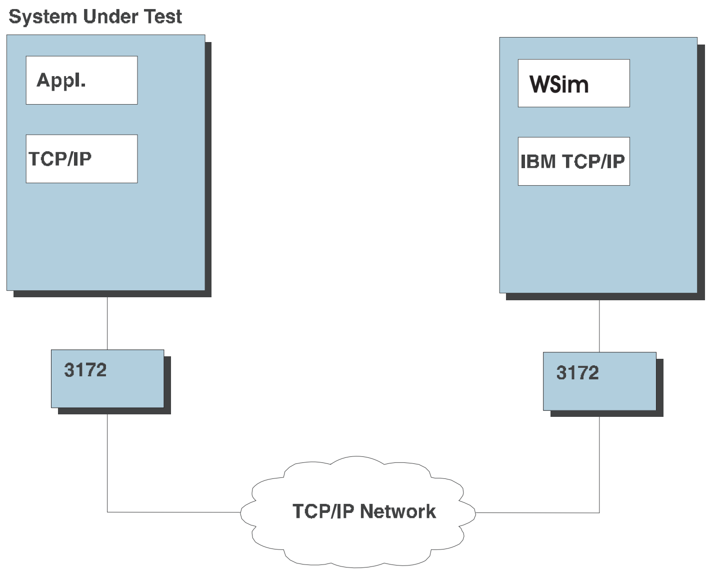


Figure 17. TCP/IP network—physical configuration

In this configuration, WSim runs as an application program of IBM TCP/IP for MVS. WSim uses the socket interface to send traffic to your TCP/IP network.

WSim does not drive any hardware directly when executing as a TCP/IP application. Instead, it uses the TCP/IP socket interface to send and receive messages, eliminating the need for any extra hardware resources. To run as a TCP/IP application program without a communication controller, WSim only requires a currently supported release of IBM TCP/IP. Refer to *TCP/IP Planning and Customization* for MVS for TCP/IP hardware requirements.

Using TCP/IP client simulation

You can use the TCP/IP application configuration to simulate TCP/IP clients to test application programs and network resources. For example, you can use the TCP/IP application configuration to simulate clients accessing your mainframe applications through a TCP/IP network.

Figure 17 on page 50 shows WSim in a different host processor than the system under test. This is not a requirement. WSim can access servers or applications in the same host or a different host. Nor does the remote host necessarily need to be a S/390 or use IBM TCP/IP. WSim uses the transport mechanisms of TCP/IP and the rest of the network so that WSim can send traffic to any other node in the TCP/IP network.

Coding the network definition

To define a TCP/IP network, code TCPIP and DEV statements. Code TCPIP statements to define the interface to the IBM TCP/IP product, and code DEV statements to define TCP/IP clients to be simulated.

An example of a network definition for simulating TCP/IP clients is shown below.

```
*----------------------------------------------------------------------
TN3270   NTWRK HEAD='TCPIP TEST NETWORK',
               THKTIME=UNLOCK,
               UTI=100,
               TCPNAME=TCPIP,
               SERVADDR=9.67.6.1
*----------------------------------------------------------------------
TN3270P  PATH  TN3270S
TN3270EP PATH  TN3270ES
TN3270PP PATH  TN3270PS
TN5250P  PATH  TN5250S
TNNVTP   PATH  TNNVTS
FTPP     PATH  FTPS
STCPP    PATH  STCPS
SUDPP    PATH  SUDPS
*----------------------------------------------------------------------
TCONN1   TCPIP
TN3270D1 DEV   TYPE=TN3270,PATH=(TN3270P)        * Telnet 3270 device
TN3270E1 DEV   TYPE=TN3270E,PATH=(TN3270EP)      * Telnet 3270E device
TN3270P1 DEV   TYPE=TN3270P,PATH=(TN3270PP)      * Telnet 3270P device
TN5250D1 DEV   TYPE=TN5250,PATH=(TN5250P)        * Telnet 5250 device
TNNVTD1  DEV   TYPE=TNNVT,PATH=(TNNVTP)          * Telnet NVT device
FTPD1    DEV   TYPE=FTP,PATH=(FTPP)              * FTP device
STCPD1   DEV   TYPE=STCP,PATH=(STCPP)            * STCP device
SUDPD1   DEV   TYPE=SUDP,PATH=(SUDPP)            * SUDP device
```

Defining the TCP/IP interface connection

To define a TCP/IP connection, code the TCPIP statement. You can code several TCPIP statements within a network definition to define several TCP/IP connections.

Code the following operands on the TCPIP statement to define general simulation characteristics of the TCP/IP connection:

- BUFSIZE operand specifies buffer size. The value specified for BUFSIZE indicates the maximum size individual message that can be generated for transmission by the simulated clients. It also indicates the maximum amount of data to be read from the TCP/IP product at one time (except when reading file data for FTP simulation, where a maximum value of 32000 is always used).
- FTPPORT operand specifies the default port number for FTP clients.
- MLEN operand specifies the maximum number of data characters to be written to the log data set for each data transfer.
- MLOG operand specifies whether this device uses the message logging function.
- STCPPORT operand specifies the default port number for STCP clients.
- SUDPPORT operand specifies the default port number for SUDP clients.
- TCPNAME operand specifies the name of the TCP/IP virtual machine or address space on the local host.
- TNPORT operand specifies the default port number for Telnet 3270, 3270E, 5250, or NVT clients.

See *WSim Script Guide and Reference* for complete statement definitions.

Defining TCP/IP clients

To define TCP/IP clients, code DEV statements. You can code several DEV statements under a TCPIP network definition statement to define several clients on a TCP/IP connection.

WSim allows any number of clients to be defined following a single TCPIP statement, but the number you code should not exceed the maximum number of sockets that the IBM TCP/IP product supports on a single connection.If you exceed the maximum, the additional DEVs do not execute. In practice, you might experience better performance by using multiple TCP/IP connections and keeping the number of devices below the maximum. To help avoid immediate congestion on a foreign host, use a random initial delay when starting devices.

Code the following operands on the DEV statement to define general simulation characteristics of the simulated client:

- ASSOC operand specifies whether the ASSOCIATE command is to be used when representing a device-type that represents a printer.
- ATRABORT operand specifies whether the current message generation deck, STL procedure, or path is aborted when the simulated client enters automatic console recovery.
- ATRDECK operand specifies the name of the message generation deck or STL procedure to be called when the simulated client enters automatic console recovery.
- CRDATALN operand specifies the length of the data to be reported in the Last Message Transmitted and Last Message Received fields for an inactivity report, a response to the Q (Query) operator command, or when the simulated client is in console recovery.
- DELAY operand specifies the value to be multiplied by the active UTI to define the default intermessage delay.
- FRSTTXT operand defines the first message generation deck or STL procedure to use when you start the network.
- FUNCTS operand specifies the list of 3270 options supported for the specific FUNCTIONS REQUEST command that the sender would like to see supported on this session.

- IUTI operand specifies the name of a UTI used in calculating all delays for this client.
- LOCLPORT operand specifies the local port number to be used by a Simple TCP or Simple UDP device.
- MAXCALL operand specifies the maximum number of outstanding message generation deck or STL procedure calls for this client.
- MSGTRACE operand specifies whether message generation trace records for this client are written to the log data set.
- PATH operand specifies the PATH statements for message generation deck or STL procedure selection to be referenced by this client in controlling message
111 generation. To iterate a particular path, append *n to the name, where n is the
111 number of iterations that is needed.
- PORT operand specifies the TCP/IP port that WSim is to use when establishing a connection.
- PRTSPD operand specifies the speed at which the device being simulated will print the data received.
- QUIESCE operand specifies whether the client is automatically marked quiesced during network initialization.
- RESOURCE operand specifies the TN3270E LU name to connect or associate with.
- RSTATS operand specifies whether online response time statistics are accumulated for this client, and reported when you issue the W (RSTATS Query) operator command.
- SAVEAREA operand specifies the number and size of the static save areas to be allocated for input data save and recall.
- SEQ operand specifies the initial value for the sequence counter at network initialization or after a network reset.
- SERVADDR operand specifies the host address to which you want to connect in a TCP/IP simulation.
- STCPHCLR operand specifies Half-Close Receive support for a Simple TCP device.
- STCPHCLX operand specifies Half-Close Transmit support for a Simple TCP device.
- STCPROLE operand specifies whether a device is to act as a client or server.
- STLTRACE operand specifies whether STL trace records for this client are written to the log data set.
- THKTIME operand specifies when the message delay starts.
- TYPE operand specifies the TCP/IP client type (Telnet 3270, 3270E, 5250, or NVT client, FTP client, or Simple TCP or UDP client) represented by this DEV statement.
- USERAREA operand defines an area of storage to be used for a scratch pad or user exit workarea.

See *WSim Script Guide and Reference* for complete statement definitions.

Simulating Telnet 3270 clients

This section provides information that is unique for Telnet 3270 and 3270E client simulations. Code TYPE=TN3270 on the DEV statement to designate the client as Telnet 3270. Code TYPE=3270E on the DEV statement to designate the client as Telnet 3270E. “Defining display characteristics” on page 54 discusses the way you define the characteristics of the display, and “Defining 3270 characteristics” defines the 3270 terminal simulation characteristics.

WSim communicates directly to TCP/IP on the local host to establish a session with the Telnet 3270 or 3270Eserver on the system under test. During this session establishment, WSim negotiates with the server by asserting and confirming options that specify 3270 data streams are to be used. As part of negotiations with the server, WSim negotiates the following options:

TRANSMIT-BINARY	Do not interpret data being transmitted as ASCII
TERMINAL-TYPE	Use a 3270 terminal type based on the extended function capability and display size being simulated. Default is IBM-3278-2-E.
END-OF-RECORD	Use the END-OF-RECORD flag (X'EF') as the data delimiter

When negotiations are complete, a Telnet 3270 session is established with the system under test. WSim then begins message generation as with other types of simulated 3270 devices.

Defining display characteristics

Code the following operands on the DEV statement to define display simulation characteristics of the simulated Telnet 3270 or 3270E client:

- DISPLAY operand specifies the default and alternate screen sizes for displays.
- LOGDSPLY operand specifies whether Telnet 3270 or 3270E client display buffers are automatically written to the log data set for formatting by the Loglist Utility.
- PROTMSG operand specifies whether the field-protected message ITP403I is written to the log data set.

Defining 3270 characteristics

Code the following operands on the DEV statement to define 3270 terminal simulation characteristics of the simulated Telnet 3270 or 3270E client:

- ALTCSET operand specifies whether the 3270 APL/TEXT character set is supported.
- APLCSID operand specifies the character set ID for the 3270 APL character set.
- BASECSID operand specifies the character set ID for the 3270 base character set.
- CCSIZE operand specifies the display character cell size for a Telnet 3270 or 3270E client that extended function support.
- COLOR operand specifies whether seven color display support is provided for a Telnet 3270 or 3270E client that extended function support.
- DBCS specifies whether double-byte character set (DBCS) is supported.
- DBCSCSID specifies the character set ID for the 3270 DBCS character set.
- EXTFUN operand specifies whether extended function support is provided for a Telnet 3270 or 3270E client.
- FLDOUTLN operand specifies whether field outlining support is provided for a Telnet 3270 or 3270E client.
- FLDVALID operand specifies whether field validation support is provided for a Telnet 3270 or 3270E client that extended function support.
- HIGHLITE operand specifies whether highlighting support is provided for a Telnet 3270 or 3270E client that extended function support.

- MAXNOPTN operand specifies the maximum number of display partitions that can be defined.
- MAXPTNSZ operand specifies the maximum size of a partition in bytes.
- PS operand specifies the number and types of Programmed Symbols character sets for a Telnet 3270 or 3270E client that extended function support.
- UASIZE operand specifies the size of the display screen in PELs (picture elements).
- UOM operand specifies the unit of measurement for the distance between PELs on the screen of a display.

Simulating Telnet 5250 and NVT clients

WSim 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
```

WSim now simulates Telnet Line Mode Network Virtual Terminal clients connecting to a Telnet server. The client looks like a Network Virtual Terminal. The WSim 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 WSim line mode support. For logic testing, only the incoming data streams are checked.

Simulating FTP clients

This section provides information that is unique for FTP client simulations. Code TYPE=FTP on the DEV statement to designate that the client uses FTP protocol. “Defining FILE characteristics” discusses the way that you define a file, and “Generating FTP commands and messages” on page 56 discusses how to generate FTP commands.

Defining FILE characteristics

Specify a FILE definition statement in the general definition statements section of your network definition to indicate the characteristics of a file to be simulated. For performance and convenience reasons, WSim does not transfer real files. Instead, WSim uses FILE statements to create simulated files.

Data received as files is logged and then discarded, rather than being saved as files.

You can code the following operands on the FILE statement to define the simulation characteristics of the file:

- DATA operand specifies a user table (UTBL or MSGUTBL) from which the data for this file is to be obtained.
- NUMREC operand specifies the number of records to be sent as part of this file.
- TYPE operand specifies whether the data is to be treated as EBCDIC (E) or ASCII (A).
- RECFM operand specifies whether the records in this simulated file are to be considered variable (V) or fixed (F) length records.
- RECLEN operand specifies the length (for fixed-length records) or maximum length (for variable length records) of each record that is a part of this file.

- MINLEN operand specifies the minimum length record to generate when RECFM=V and DATA is not specified.

Generating FTP commands and messages

WSim support for FTP is structured so that FTP commands are generated using the TYPE (in STL) or TEXT statements in the same way that messages to be transmitted are generated for other terminal types. Rather than transmitting this data directly, the FTP support interrogates the data and interprets it as an FTP command. The WSim FTP support builds the data transmitted to the FTP server. This data might consist of one or more server commands and associated actions. Some commands might not result in any transmission to the server at all, but might set local indicators only, where others might involve several transactions.

WSim makes all data received from the server available for inspection using logic testing statements (IF, ONIN). In some cases, WSim provides a message to the operator as if it had been received. Such a message is also available for logic testing. This is important when no server response can be expected for a given command. At the end of each file transfer, WSim builds a message indicating transfer size and rate and treats the message as if it had been received. All such messages built by WSim begin with a standard message prefix and number that can be used in logic testing. FTP support uses message numbers ITP482I through ITP485I and ITP487I through ITP489I. Refer to *WSim Messages and Codes* for further information on these messages.

WSim supports FTP commands that are a subset of those supported by the IBM TCP/IP products for MVS. WSim can simulate FTP clients that interact with real FTP servers, but it does not attempt to provide a full function FTP. WSim provides a facility to enable the loading and testing of a network that uses FTP protocols, but not a facility for transferring and storing real files using FTP.

Because of its operating environment and its objectives, WSim does not attempt to support FTP commands that are strictly local in nature. In addition, the commands that reference local files to be transferred require that the file name match the name of a FILE statement. You do not need to specify a local file name on commands that transfer files to WSim. Where remote foreign files or directories are required, specify them in a format acceptable to the remote FTP server.

Table 1 shows the subset of FTP commands supported by WSim, their minimum abbreviations, their acceptable operands, and a brief description of their functions. Some additional notes about the operands follow the table. Additional information about the general function of these commands can be found in the *IBM TCP/IP for MVS User's Guide*.

Table 1. WSim FTP subcommand summary

Subcommand	**Minimum Abbreviation**	**Operands**	**Description**
ACCOUNT	AC	*account_information*	Sends host-dependent account information.
APPEND	AP	*WSIM_file* [*foreign_file*]	Appends a file defined by a FILE statement to a file on the server.
ASCII	AS		Sets the file transfer type to ASCII.
BINARY	B		Sets the file transfer type to IMAGE.
CD	CD	*foreign_directory*	Changes the working directory on the server.

Table 1. WSim FTP subcommand summary (continued)

Subcommand	Minimum Abbreviation	Operands	Description
CLOSE	CL		Disconnects from the server.
CWD	CW	*foreign_directory*	Changes the working directory on the server.
DELETE	DELE	*foreign_file*	Deletes a file on the server.
DIR	DI	[*foreign_directory*]	Retrieves the directory entries for files on the server.
EBCDIC	EB		Sets the file transfer type to EBCDIC.
GET	G	*foreign_file*	Transfers the data from a file on the server, but does not save it.
LS	LS	[*foreign_directory*]	Retrieves the names of files on the server.
MKDIR	MK	*foreign_directory*	Creates a directory on the server.
MODE	MO	B | S	Specifies the file transfer mode as Block or Stream.
NOOP	NO		Checks whether the server is still responding.
PASS	PA	*foreign_password*	Supplies a password to the server.
PUT	PU	*WSIM_file* [*foreign_file*]	Transfers the FILE data defined by a FILE statement to a file on the server.
PWD	PW		Retrieves the name of the active working directory on the server.
QUIT	QUI		Disconnects from the server.
QUOTE	QUO	*string*	Sends an uninterpreted string of data to the server.
RENAME	REN	*foreign_file new_foreign_file*	Renames a file on the server.
SENDPORT	SENDP		Enables or disables automatic transmission of the PORT subcommand.
SENDSITE	SENDS		Enables or disables automatic transmission of the SITE subcommand.
SITE	SI	*foreign_site_data*	Sends information to the server using site-specific commands.
STATUS	STA		Retrieves status information from the server.
STRUCT	STR		Sets the file transfer structure.
SUNIQUE	SU		Toggles the storage methods.
SYSTEM	SY		Retrieves the name of the server's operating system.
TYPE	TY	A | E | I	Specifies the file transfer type as ASCII or EBCDIC or IMAGE.
USER	U	*user_name*	Identifies the user to the server.

Table 1. WSim FTP subcommand summary (continued)

Subcommand	Minimum Abbreviation	Operands	Description

Notes:

- *WSIM_file* indicates a place where the name of a FILE statement within the active Network Definition must be coded.
- Use the format required by the remote FTP server for each operand identified as a *foreign_file* or *foreign_directory*. These operands are passed as specified to that server.
- If no *foreign_file* is specified on the PUT or APPEND commands, WSim provides a name of the form *WSIM_file*.$DEFAULT.
- All commands and responses are translated to or from ASCII code. File data that is associated with the commands is translated when required.
- When a simulated file is to be transferred, WSim generates a minimal SITE command unless this function was toggled off using the SENDSITE command. This SITE command specifies a record format and, if possible and necessary, a logical record length using the MVS format. If other SITE information is needed, a separate SITE command can be sent before the PUT. Specifically, the automatically built SITE command has one of the following formats:
  - If RECFM=V is specified or defaulted,

    ```
    SITE  VARrecfm    RECFM=VB LRECL=vlen
    ```

    where *vlen* is the value specified by RECLEN plus four, unless RECLEN was not coded or the value would be greater than 32756. In that case LRECL is omitted.
  - If RECFM=F is specified,

    ```
    SITE  FIXrecfm  flen   RECFM=FB LRECL=flen
    ```

    where *flen* is the value specified by RECLEN. If *flen* is greater than 32760, the LRECL specification is omitted.
- The amount of formatting and translation done by WSim for the simulated files it transfers depends upon the specified or assumed characteristics of the designated FILE and the MODE and transfer TYPE in effect at the time the PUT or APPEND is issued.

  If Block mode is specified, WSim inserts a 3-byte block header in front of each logical record transmitted. If Stream mode is specified and the file transfer type is ASCII or EBCDIC, WSim inserts an appropriate end of record indicator at the end of each record (CRLF for ASCII transfers and NL for EBCDIC transfers). No end of record indication is inserted if the transfer type is Image.

  Data translation occurs if the transfer type is ASCII and the FILE data type is EBCDIC, or the transfer type is EBCDIC and the FILE data type is ASCII. All data transmitted on the command connection is translated to ASCII and all data received on the command connection is translated to EBCDIC.

Simulating simple TCP clients

This section provides information that is unique for Simple TCP client simulations. Code TYPE=STCP on the DEV statement to designate the device as Simple TCP client.

This type of simulation can be useful for testing a wide variety of server types and protocols, but it requires knowledge of the structure of the messages sent and received for the protocols being used.

As with the other TCP/IP protocols, the IBM TCP/IP product handles all of the routing and delivery of messages transmitted and received for Simple TCP clients. WSim uses basic socket calls to establish and close connections to a server, and it sends and receives data on these connections.

Simple TCP Client support provides a means of simply sending user-defined messages and receiving server-transmitted messages via TCP/IP connections without any manipulation of the data other than that provided by the script. The script can also specify opening and closing of the connections to the server.

Unlike for Telnet 3270, 3270E, 5250, NVT, and FTP simulations, WSim does not establish a connection for Simple TCP until a message was generated and is available to send. At that point WSim obtains a socket and initiates a connection to the server and port specified by the DEV operands. After the connection is made, WSim transmits the data. The connection is then available for receipt of data from the server or additional transmission of data.

All data transmitted and received is available for logic testing using IF or ONOUT and ONIN scripting statements. It is then discarded. If the script generates a null message, WSim interprets it as a signal to close the connection to the server. When a null message is received indicating that the server closed the connection, this message is available for logic testing just as other messages.

Limited server

For STCP devices, you can 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 an 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 that 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 socket interface, which is as follows:

Table 2. 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 WSim scripting language), ITPGSIPA moves the INET address from network save area 13 into the internal WSim 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 2 on page 59.

Simulating simple UDP clients

WSim supports simulation of Simple UDP terminals. The terminal type provides simple client UDP support in WSim. 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.

Chapter 6. Simulating SNA resources and subareas

This chapter discusses how to use WSim to simulate Systems Network Architecture (SNA) resources. The first part of this chapter briefly explains how WSim supports SNA simulations. It then explains how to define SNA terminals. Instructions and examples show how to code SNA resources within the host. This chapter discusses how resources are activated and deactivated in SNA simulations and how WSim processes SNA request and response units (RUs).

How WSim processes request/response units

This section discusses how WSim processes request/response units (RUs) sent to and from the system under test during LU-LU sessions.

Request/response units in terminal simulations

If a WSim logical unit is in session with a logical unit in the system under test, normal data traffic between the two LUs can occur in addition to the traffic between the logical units and VTAM. The requests and responses processed during a session fall into 3 categories:

- Session control (SC)
- Data flow control (DFC)
- Function management data (FMD).

Messages from each of these categories contain command codes in the first byte of the RU. The FMD category also includes messages that contain user data in the RU rather than SNA commands. Table 3 through Table 5 on page 62 list the SNA commands that can be sent or received by WSim when simulating terminals in the same domain as the system under test.

Table 3. Session control commands and responses

RU Request Code (hex)	Command Name	Sent by WSim	Received by WSim
31	Bind Session (BIND)		X
32	Unbind Session (UNBIND)		X
A0	Start Data Traffic (SDT)		X
A1	Clear (CLEAR)		X
A2	Set and Test Sequence Numbers (STSN)	X	X
A3	Request Recovery (RQR)	X	

Table 4. Data flow control commands and responses

RU Request Code (hex)	Command Name	Sent by WSim	Received by WSim
04	Logical Unit Status (LUSTAT)	X	X
05	Ready to Receive (RTR)	X	
70	Bracket Initiation (SBI)	X	X
71	Stop Bracket Initiation (SBI)	X	X
80	Quiesce at End of Chain (QEC)	X	X

Table 4. Data flow control commands and responses (continued)

RU Request Code (hex)	Command Name	Sent by WSim	Received by WSim
81	Quiesce Complete (QC)	X	X
82	Release Quiesce (RELQ)	X	X
83	Cancel (CANCEL)	X	X
84	Chase (CHASE)	X	X
C0	Shutdown (SHUTD)		X
C1	Shutdown Complete (SHUTC)	X	
C2	Request Shutdown (RSHUTD)	X	
C8	Bid (BID)		X
C9	Signal (SIG)	X	X
F8	Pseudobid (PSEUDOBID)		X

Table 5. Function management data and network services commands and responses

RU Request Code (hex)	Command Name	Sent by WSim	Received by WSim
010681	Initiate® Self, Format 0 (INIT-SELF)	X	
010683	Terminate Self, Format 0 (TERM-SELF)	X	
810620	Notify (SSCP-->LU, LU-->SSCP) (NOTIFY)	X	X
810629	Cleanup (CLEANUP)		X

Receiving messages

When a simulated SNA resource (LU) receives a message from the system under test,WSim performs the following series of functions for the data:

1. Logs the message.
2. Passes the message to the network user exit.
3. Performs internal WSim processing on the message.
4. Passes the message to the logic testing routine.

When a WSim logical unit receives an SNA command or command response (SC, DFC, or FMD(NS)), it updates any applicable SNA states and saves any information from the message that may be used later. For example, the session rules defined in a BIND command are remembered by a WSim LU for reference throughout a session. The WSim LU enforces all BIND session rules on all received data, such as RU chaining, bracket states, and types of responses to be sent. A WSim LU automatically builds and sends a response RU for any SNA command received.

When a WSim LU receives an FM data request, it processes the information in the SNA headers (TH and RH) to update its state values. For most SNA terminals, WSim does not perform any automatic processing on the user data in an FMD RU. You can process the data in a user exit, or you can perform logic tests on the data using the IF statements.WSim does provide complete RU processing for the 3270 and 5250 display terminals. A received data stream is processed for an SNA display terminal by analyzing the commands and orders in the data, and by maintaining a screen image buffer in storage for the terminal.

When WSim is ready to build an SNA response for an FMD request received by an LU, it checks to see if you have coded a value for the PRTSPD operand for the LU. This operand specifies the speed at which the simulated terminal prints the data received. If the PRTSPD value exists, a delay is calculated and observed before sending the response; otherwise, the response is sent immediately.

After WSim has performed all of its internal processing on a received message, the message is passed to the logic test routine so that you can make decisions affecting message generation based on the received data.

Transmitting generated messages

When trying to generate a message for a specific LU, WSim first checks for an SNA response that must be sent because of a request previously received by the LU. If no SNA response is pending for the terminal, WSim attempts to generate an SNA command for the LU.WSim can generate a network services, session control, or data flow control command depending on the current state of the LU as determined by the previous message transfers in the session.

If an LU does not have a pending SNA response and a command cannot be generated, WSim attempts to invoke message generation. All of the normal conditions for entry to message generation must be met by the LU. (These conditions are defined in Part 3, “Coding message generation statements,” on page 105.) In addition, the LU must be in the correct SNA state for transmitting data. If message generation can be entered for the LU, WSim builds default SNA headers for the message and then processes the statements in your message generation deck until a message has been generated. The LU type and the current state of the LU determine the contents of the default headers. You can use the TH and RH statements to alter the default SNA headers built by WSim and to control chaining. See Part 3, “Coding message generation statements,” on page 105 for information about using these statements. If you are using STL and want to modify the TH and RH or control chaining, refer to *WSim Script Guide and Reference* for more information.

Request/response units in SNA simulations

Below lists the SNA commands that can be sent and received by WSim when performing SNA simulations.

Session Control

Code (hex)	**Command Name**
31	Bind Session (BIND)
32	Unbind Session (UNBIND)
A0	Start Data Traffic (SDT)
A1	Clear(CLEAR)
A2	Set and Test Sequence Numbers (STSN)
A3	Request Recovery (RQR)

Data Flow Control

Code (hex)	**Command Name**
04	Logical Unit Status (LUSTAT)
05	Ready to Receive (RTR)
70	Bracket Initiation Stopped (BIS)

Data Flow Control

Code (hex)	Command Name
71	Stop Bracket Initiation (SBI)
80	Quiesce at End of Chain (QEC)
81	Quiesce Complete (QC)
82	Release Quiesce (RELQ)
83	Cancel(CANCEL)
84	Chase(CHASE)
C0	Shutdown(SHUTD)
C1	Shutdown Complete (SHUTC)
C2	Request Shutdown (RSHUTD)
C8	Bid(BID)
C9	Signal(SIG)
F8	Pseudobid(PSEUDOBID)

Function Management Data, Network Services

Code (hex)	Command Name
010681	Initiate Self, Format 0 (INIT-SELF)
010683	Terminate Self, Format 0 (TERM-SELF)
810601	Control Initiate (CINIT)
810620	Notify (SSCP-->LU, LU-->SSCP) (NOTIFY)
810629	Cleanup(CLEANUP)

Chapter 7. Simulating specific devices

This chapter provides instructions for simulating devices that require special coding considerations. These special considerations include factors such as character set identification and function restrictions, and what types of coding you must include in network definitions to simulate certain features of specific devices.

The remaining sections in this chapter discuss the following devices:
- IBM 3270 Information Display System
- IBM 3290 Information Panel
- IBM 5250 Display System

IBM 3270 Information Display System

When simulating a 3270 display system, WSim has certain requirements regarding character set identification and the Display Monitor Facility. There are also certain restrictions on 3270 functions thatWSim can simulate.

3270 character set identification

The Coded Graphic Character Set Global Identifier (CGCSGID), that is, character set ID, for the simulated 3270 character sets can be specified using the BASECSID, APLCSID, and DBCSCSID operands. The character set IDs are passed to the host application program in response to a 3270 Query request. NLS enabled application programs use the character set IDs when referencing panels formatted uniquely for different countries.

Display Monitor Facility

With Display Monitor Facility, you can view simulated 3270 display images on a monitoring 3270 display during a simulation run. Refer to *WSim User's Guide* for a detailed description of the Display Monitor Facility.

Restrictions

WSim does not support the following 3270 functions as an LU Type 2:
- Vector-to-Rastor Graphics (3179-GX and 3192-G)
- Image Support
- PC File Transfer.

Other LU types, for example LU Type 0, can be used if the appropriate data stream is coded or generated using the Script Generator Utility.

IBM 3290 Information Panel

The IBM 3290 Information Panel can display up to four concurrent logical terminals on one physical display screen. Each of these terminals is an independent logical unit as far as the host applications are concerned. Each logical unit has a unique address.

The 3290 simulation is based upon the logically separate terminals. You define each of the logical terminals as individual logical units and associate them as a set through your message generation decks. You can simulate 3290 devices using SNA protocol by coding one of the following definition:

```
LU   LUTYPE=LU2
```

You can code an SNA 3290 as an LU2 logical unit in a VTAMAPPL simulation.WSim sends the proper data stream for the 3290. Since WSim considers each logical terminal in a 3290 set as a separate LU, you can code from one to four LU statements to represent a single 3290.

You must associate the LU statements in your 3290 definition as a set by following resource limits for the real 3290. As far as the host is concerned, no relationship exists between the logical terminals. If operator interaction between the screens is important to the simulation, use EVENT statements and logic tests in your message generation decks to simulate the interaction.

Operands required for 3290 simulation

The following list is a list of the operands and values you must code on theLU statement to simulate the 3290:

- LUTYPE=LU2

  LU2 means that the 3290 is an SNA LU Type 2 device.
- COLOR=ORANGE

  The 3290 accepts and transmits the color attribute even though it is only able to display in orange.
- UOM=MM

  The unit of measure for screen sizes is millimeters. 1 millimeter is equivalent to one picture element (PEL). For other 3270 devices, size is represented in inches (cells).
- FLDVALID=NO

  The 3290 does not support field validation.
- MAXNOPTN=(0, 8, or 16)

  The maximum number of partitions for the logical terminal set is 16. Since 0, 8, and 16 are the only values allowed, a terminal set can have at most one logical terminal with 16 partitions or 2 with 8 partitions each.
- EXTFUN, HIGHLITE, ALTCSET, PS

  The number of PS sets allowed for the 3290 is six. Specify only single-plane character sets. Only one logical terminal in the set can have PS support. You can select EXTFUN, HIGHLITE, and ALTCSET as with any 3270 device.
- UASIZE, CCSIZE, and DISPLAY

  Use UASIZE=(*w*,*h*) and CCSIZE=(0,0) for variable cell sizes.

  Use CCSIZE=(*x*,*y*) for fixed character cell size.

  Use DISPLAY=(*r*,*c*).

The following section explains how to determine values for UASIZE, CCSIZE, and DISPLAY.

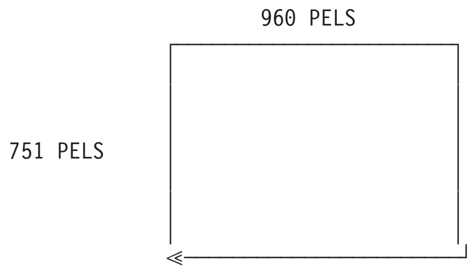
Logical terminal screen definition

To define the logical terminal screens that are displayed on the 3290 physical screen, you must select values for the UASIZE, DISPLAY, and CCSIZE operands that ensure the screens fit on the 3290 physical screen.

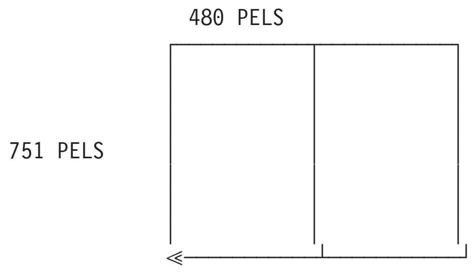
Determining screen size (UASIZE) from the screen split

To ensure that the screens for the logical terminals fit on the 3290 physical screen, you must first select the screen split you wish to simulate. Once you have selected a screen split, use the following guidelines to determine UASIZE:

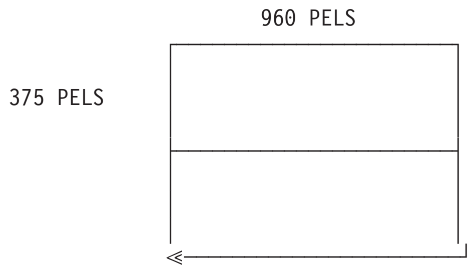
- Full Screen—UASIZE=(960,751)



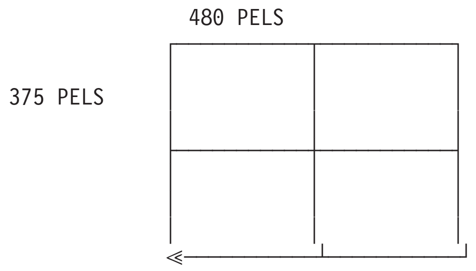
- Vertical Half Screen—UASIZE=(480,751)



- Horizontal Half Screen—UASIZE=(960,375)



- Quarter Screen—UASIZE=(480,375)



Determining character cell size (CCSIZE) and display size (DISPLAY)

Use Table 6 on page 68 and Table 7 on page 68 and the following guidelines to determine the CCSIZE and DISPLAY values for the logical terminal. Examples follow the tables.

- If the character cell size is variable (CCSIZE=(0,0)) or you know the display size only, use the number of rows and columns for the specific type of physical screen to determine the cell size from the tables.
- If the character cell size is fixed (CCSIZE=(*x*,*y*)), use the dimensions of the cell and the type of physical screen (full, vertical half, horizontal half, or quarter) to determine the number of rows and columns from the tables.

Table 6. Table to determine rows on screen or height of character cell

Height of Character Cell	Number of Rows for Full Screen or Vertical Half Screen	Number of Rows for Horizontal Half Screen or Quarter Screen
31	24	12
30	25	12
29	25	12
28	26	13
27	27	13
26	28	14
25	29-30	15
24	31	15
23	32	16
22	33-34	17
21	35	17
20	36-37	18
19	38-39	19
18	40-41	20
17	42-44	21-22
16	45-46	23
15	47-50	24-25
14	51-53	26
13	54-57	27-28
12	58-62	29-31

Table 7. Table to determine columns on screen or width of character cell

Width of Character Cell	Number of Columns for Full Screen or Horizontal Half Screen	Number of Columns for Vertical Half Screen or Quarter Screen
12	80	40
11	81-87	41-43
10	88-96	44-48
9	97-106	49-53
8	107-120	54-60
7	121-137	61-68
6	138-160	69-80

The following four examples demonstrate how you can use these tables to determine the CCSIZE and DISPLAY values.

***Example 1*:** CCSIZE=(8,12) displayed on a horizontal half screen.

1. Find where height=12 and number of rows for horizontal half screen intersect in Table 6.
2. Pick a number of rows from the range listed (29-31).

3. Find where width=8 and number of columns for horizontal half screen intersect in Table 7 on page 68.
4. Pick a number of columns from the range listed (107-120).

A valid definition for this example is CCSIZE=(8,12), DISPLAY=(31,120). You can omit UASIZE when the cell size is fixed.

***Example 2*:** DISPLAY=(24,80) displayed on a quarter screen with fixed CCSIZE.

1. Find where number of rows=24 falls within the range listed under quarter screen in Table 6 on page 68. In this case, it is where the cell height is 15.
2. Find where number of columns=80 falls within the range listed under quarter screen in Table 7 on page 68. In this case, it is where the cell width is 6.

A valid definition for this example is DISPLAY=(24,80), CCSIZE=(6,15). You can omit UASIZE when the cell size is fixed.

***Example 3*:** DISPLAY=(24,80) displayed on a quarter screen with variable CCSIZE.

1. Verify that number of rows=24 falls within the range listed under quarter screen in Table 6 on page 68. From the table, 12-31 rows are valid for a quarter screen.
2. Verify that number of columns=80 falls within the range listed under quarter screen in Table 7 on page 68. From the table, 40-80 columns are valid for a quarter screen.

A valid definition for this example is DISPLAY=(24,80), CCSIZE=(0,0), UASIZE=(480,375).

***Example 4*:** DISPLAY=(31,160) displayed on a vertical half screen with variable CCSIZE.

1. Verify that number of rows=31 falls within the range listed under vertical half screen in Table 6 on page 68. From the table, 24-62 rows are valid for a vertical half screen.
2. Verify that number of columns=160 falls within the range listed under vertical half screen in Table 7 on page 68. From the table, only 40-80 columns are valid for a vertical half screen. Therefore, this DISPLAY size cannot work.

Simulating 3270 DBCS devices

To simulate IBM 3270 DBCS devices, such as an IBM PS/55 executing an IBM 3270 DBCS emulation program or an IBM InfoWindow 7672-JC1 3270 DBCS display, code DBCS=YES and FLDOUTLN=YES. These operands enable the DBCS and Field Outlining support.

When DBCS=YES is coded, WSim accepts 3270 data streams with DBCS data to be displayed. The 3270 Query reply, sent in response to the 3270 Query request received from the host application program, indicates support of DBCS-Asia and also indicate support of a DBCS character set. The message generation process accepts DBCS data from the script and update the simulated screen image accordingly.

Note: See Part 2, “Introducing message generation decks,” on page 93 for message generation considerations.

When FLDOUTLN=YES is coded, WSim accepts 3270 data streams with Field Outlining extended attributes. The 3270 Query reply also indicates support of Field Outlining.

Note: In WSim, field outlining might be supported without supporting DBCS. However, field outlining is normally only associated with DBCS 3270 emulation.

IBM 5250 Display System

WSim simulates an IBM 5250 display terminal as a logical unit Type 7 (device type LU7) and an IBM 5250 printer as a logical unit Type 4 (device type LU4).WSim maintains a screen image buffer and a format table for each terminal as supported by the 5251 controller. The buffer can be modified by messages generated by WSim and by commands and orders received from the system under test. The format table can be modified only by commands and orders received from the system under test.

The format table consists of header information that is supplied by the start of header (SOH) order, and one entry for each field defined in the screen image buffer. Each entry in the format table consists of at least a starting and ending address that defines the field limits on the screen image and a field format word (FFW) that contains information pertinent to that field. Such information might be the type of field, the modified data tag bit, and various field specifications that govern how that field is to be processed.

Data generated by WSim is placed into a terminal buffer under control of a Read command that must be received from the system under test before message generation can be entered.WSim automatically breaks the transmitted data into RU chain elements.

When Read commands for a terminal are received in the data stream, they are placed in a queue. When a poll is received and all conditions for entering message generation are met, a message is generated into the terminal buffer under control of the FFW for the field being written into.

A terminal is placed in Normal Lock State when a message is generated or when certain commands are received. It is placed in Normal Unlock State when its message delay expires or when a Read command is received with the unlock keyboard bit set to B'1'.

Logic testing

You can use the IF statements to perform logic tests for 5250 terminals on the screen image buffer data or the incoming or outgoing data stream.

You can specify a logic test on the screen image buffer for the terminal by coding the B+, B-, C+, C-, or (*row,col*) location options on an IF statement. This type of logic test operates on the data as it would be displayed at a real terminal. All logic tests on a screen image buffer are performed after the buffer has been modified according to the message generated or received. If a received message contains invalid commands or orders, the data received before the invalid command or order is processed and can modify the screen image buffer, but no data following the invalid command or order is processed.

You can specify a logic test on an incoming or outgoing data stream, including headers, commands, and orders, by coding the D+, TH+, RH+, or RU+ location option on an IF statement. See *WSim Script Guide and Reference* for information about how to do this in STL.

Formatting the screen image buffer and format table

The maintained 5250 screen image buffer and format table can be written to the log data set and later formatted by the Loglist Utility into screen images as the 5250 display operator would see them. See the *WSim Script Guide and Reference* for information about using the LOGDSPLY operand to write the data to the log data set. See *WSim Utilities Guide* for information about using the Loglist Utility.

Chapter 8. Coding network options

This chapter discusses some of the optional statements and operands you can code after the NTWRK statement when you define your simulated network. You can use these statements and operands to control many of the features of your simulated network such as the time frame used for starting devices in the network, the order in which message generation decks or STL procedures are used, and the method of logging messages sent and received by WSim.

The statements and operands discussed in this chapter are the ones you use most frequently. This chapter provides a brief introduction to these options and indicates how you code them. For complete explanations of other options you can code on the NTWRK statement and the syntax of the statements and operands presented in this chapter, refer to *WSim Script Guide and Reference*.

This chapter discusses the following network options:
- Counters and STL integer variables
- Future events and start time
- Message generation delays and transmit interrupts
- Message logging
- Network logic tests
- Online response-time statistics
- Paths for message generation decks and STL programs
- Random number generation
- Terminal scanning and automatic terminal recovery
- Tracing of messages and Structured Translator Language (STL) programs
- User data tables
- User exit routines.

Counters and STL integer variables

WSim provides two types of counters: sequence and index. A sequence counter is incremented before its value is inserted into a data field. An index counter is used to select an entry from a user table. Its value might be incremented depending on how you reference the counter. STL supports index counters only, which in STL are called integer variables.

Sequence and index counters can be used to generate data that is inserted directly into messages generated by WSim or to reference entries in user tables. You can also use counters to specify cursor locations for simulated display devices and to specify variable offsets in data buffers and save areas.

The value of a sequence or index counter automatically wraps to 0 after reaching 2 147 483 647. However, for a sequence counter, 0 is never inserted into a message because the value is updated to 1 when the counter is referenced.

Sequence counters

WSim maintains a sequence counter for each network, VTAM application, CPI-C transaction program, and TCP/IP connection, device, and logical unit. The counters are called NSEQ, LSEQ, TSEQ, and DSEQ, where the first letter stands for network, line, terminal, and device, respectively. While VTAM applications, CPI-C transaction programs, and TCP/IP connections do not have lines or terminals associated with them, each has a line sequence counter (LSEQ) and a terminal sequence counter (TSEQ). Each LU defined for a VTAM application, transaction program defined for an APPC LU, or device defined for a TCP/IP connection has an associated device sequence counter.

Index counters

WSim maintains from 3 to 4095 index counters or STL integer variables for each network, line, VTAM application, CPI-C transaction program, and TCP/IP connection, device, and logical unit. The counters are called NC*n*, LC*n*, TC*n*, and DC*n*, where the first letter stands for network, line, terminal, and device, respectively, and *n* refers to the number of the specific counter. In STL, the variable names can be any valid STL variable name; see *WSim Script Guide and Reference* for more information. STL shared integer variables reference NC*n* counters and unshared integer variables reference DC*n* counters. While VTAM applications, CPI-C transaction programs, and TCP/IP connections do not have lines or terminals associated with them, each can have from 3 to 4095 line index counters (LC*n*) and from 3 to 4095 terminal index counters (TC*n*). Each LU defined for a VTAM application, transaction program defined for an APPC LU, or device defined for a TCP/IP connection can have from 3 to 4095 device index counters (DC*n*).

The CNTRS operand on the NTWRK statement determines the number of index counters that WSim maintains for each resource. If you specify any user exits, WSim allocates either the number of index counters or STL integer variables specified on the CNTRS operand or the number of counters referenced in the network, whichever is greater. If you do not specify any user exits, WSim allocates as many index counters as are needed and ignores the CNTRS operand. WSim allocates a minimum of three index counters.

You need to code the CNTRS operand only if you use a user exit that depends on the number of counters allocated.

Allocation of counters for resources

Table 8 illustrates the counters that are allocated for each simulated resource.

Table 8. How counters are allocated for resources

Simulation Type	**LSEQ LC1 - LC*n***	**TSEQ TC1 - TC*n***	**DSEQ DC1 - DC*n***
CPI-C TP			
APPCLU	X		
TP		X Note 1	X Note 2
VTAM Application			
VTAMAPPL	X	X	
LU			X
TCP/IP Connection			
TCPIP	X	X	
DEV			X

Table 8. How counters are allocated for resources (continued)

Simulation Type	LSEQ LC1 - LC*n*	TSEQ TC1 - TC*n*	DSEQ DC1 - DC*n*

Notes:

1. This set of counters is available to all instances of a given CPI-C transaction program.
2. This set of counters is unique for each CPI-C transaction program instance.

Altering the values of counters and STL integer variables

Counters:

You can alter the values of sequence and index counters by using the SET statement in a message generation deck. You can use the SET statement to set a counter to any of the following values:

- An integer
- A random number
- The value of another counter
- The result of addition, subtraction, multiplication, division, or remainder division between the counter and a constant or between the counter and another counter
- 1 to 4 bytes of hexadecimal data in a save area
- The EBCDIC character representation of a number in data
- The simulated cursor's position
- The offset within a save area, user area, buffer, or data stream where matching text is found as the result of a logic test.

See Part 2, "Introducing message generation decks," on page 93 for information about how to use the SET statement to alter the values of counters and how to reference counters in message generation statements.

STL integer variables:

You can alter the value of STL integer variables by using an STL assignment statement. See *WSim Script Guide and Reference* for information about using assignment statements.

Future events and start time

The Future Event (FE) network definition statement and the Start Time (STIME) operand provide two ways of automatically controlling network simulations. These definitions apply to overall network control rather than individual terminal control.

Use the Future Event (FE) statement to specify an operator command that is automatically executed during the simulation run under the following conditions:

1. When a timer expires
2. When an event is signaled.

For example, the following statement causes a user time interval (UTI) to be set automatically to 0 at 1 minute 30 seconds into the run.

```
FE TIME=90,
   COMMAND=(A NTWRK,U=0)
```

Multiple FE statements can be coded within one network definition.

Use the STIME operand on the NTWRK statement to stagger the start delay for each VTAMAPPL, APPCLU, or TCPIP in the network. For example, the following operand value starts the VTAMAPPLs, APPCLUs, and TCPIPs (in the order coded in the network definition) one at a time at 60-second intervals following the S (Start) operator command.

```
STIME=60
```

Refer to *WSim Script Guide and Reference* for more information about coding future events and start time.

Message generation delays and transmit interrupts

WSim provides several options for controlling the delays a simulated terminal uses between the messages it sends. You can use these options by specifying the statements and operands described below. Refer to *WSim Script Guide and Reference* for specific syntax information for these statements and operands.

In STL, this delay is referred to as the transmit interrupt. For more information about transmit interrupts when using STL, refer to *WSim Script Guide and Reference*.

UTI statement

Use the UTI statement in your network definition to define user time intervals for computing the message delays. Code UTI values in hundredths of seconds. You can code any number of UTIs to allow devices to operate at different speeds within the same network. WSim uses the UTI values that you code as scale factors by which user-specified delay values are multiplied.

The UTI statement includes a label and a UTI value as shown in the following examples:

```
UTI1   UTI   100
UTI2   UTI   0
```

Note that if you specify UTI with a value of 0, there is no delay between messages generated and sent by WSim when using this UTI.

You can reference a UTI on a DEV, LU, or TP statement by using the IUTI operand as shown in the following example:

```
DEV2   DEV   IUTI=UTI1
```

The IUTI operand specifies the individual UTI (UTI1) that is to be used in calculating all delays for this device. It must reference a UTI statement that is defined within the network definition.

You can also reference UTIs from message generation decks and alter UTIs using operator commands. Refer to Part 2, “Introducing message generation decks,” on page 93 and *WSim User’s Guide* for more information about referencing and altering UTIs. Refer to *WSim Script Guide and Reference* for more information about the UTI statements and operands.

The following example shows a network definition and message generation deck that uses multiple UTIs.

```
SAMPLE1  NTWRK    UTI=100
:
UTIA     UTI      0
UTIB     UTI      1000
UTI1     UTI      1
UTI2     UTI      2
:
         VTAMAPPL IUTI=UTIB
LU1        LU
LU2        LU     IUTI=UTI1
         VTAMAPPL
LU3        LU
ONE      MSGTXT
         SETUTI    UTI=UTI2
:
         ENDTXT
TWO      MSGTXT
         DELAY     TIME=F1,UTI=UTIB
:
         ENDTXT
```

The UTI for LU1 is UTIB. The UTI for LU2 is UTI1. And the UTI for LU3 is the network UTI. SETUTI alters the UTI value only for the device that is executing deck ONE. The network UTI is not altered. The DELAY message generation statement specifies an intermessage delay. The values specified by the TIME and UTI operands are multiplied together to obtain the value for DELAY.

The following example shows how to code the message generation decks shown in the above example using STL. For information about STL, see *WSim Script Guide and Reference*.

```
one:  msgtxt
      uti 'UTI2'

:
      endtxt
two:  msgtxt
      uti 'UTIB'
      delay(1,'UTIB')

:
      endtxt
```

UTI adjustment

WSim can adjust UTI values according to an expected message transfer rate that you specify for messages transmitted from WSim. You can specify this expected message transfer rate in your network definition. WSim automatically adjusts the UTIs at specified intervals to maintain the rate. By lowering UTIs and increasing rates of traffic from other simulated terminals, WSim can compensate for lines dropping out due to hardware or software problems.

WSim adjusts each UTI by taking the ratio of the observed rate for the previous interval to the expected rate for the entire network and multiplying all UTIs by this ratio. The limit of change for each calculation is 25 percent of the current value of the UTI.

Specify the expected message transfer rate and the interval at which adjustments take place on the EMTRATE operand of the NTWRK statement. You can also use the A (Alter) operator command to specify or alter the rate after the traffic rate is established. See *WSim User's Guide* for information about using this operator command.

For adjustments to work effectively, UTI values must be large enough to allow adjustment either upward or downward. WSim never adjusts a UTI to zero. If you set a UTI to zero, no further adjustment takes place. In addition, no adjustment takes place if the observed rate, the expected rate, or the adjustment interval is zero. When you change the adjustment interval, WSim requires three intervals to accumulate enough data to make any adjustment in the UTIs. Thereafter, each UTI is adjusted at each interval, if required, unless no messages were transferred in the interval just completed or in the interval preceding the one just completed.

If the expected rate is changed drastically during a run, you can obtain improved results by first adjusting the UTIs with the A (Alter) operator command to approximate the wanted rate. If the expected rate is low, the adjustment interval must be relatively large in order to ensure some message traffic in successive intervals.

The following list provides guidelines for adjusting UTIs:

- Use an adjustment interval large enough that the rate is relatively constant over successive intervals. When adjusting UTIs, WSim converts the rate to messages per minute. Distorted results might be obtained if the interval is too small.
- When the rate or the number of lines in a network is small, the adjustment interval must be relatively large. As the number of lines and the expected rate increase, you can decrease the interval.
- Do not use the A (Alter) operator command to make a drastic change in the expected rate. Such a change can cause UTIs to be adjusted unreasonably high with resulting long delays and the loss of most or all traffic. If you want a drastic change in the rate, turn off the automatic adjustment feature by setting the adjustment interval to zero and alter UTI values to approximate the rate wanted. Then, wait for a stable rate to be established and set a new expected rate and adjustment interval.
- Automatic UTI adjustment affects all UTIs in the network. If you have many UTIs, this adjustment might produce unexpected results. It is best to use automatic UTI adjustment with a few UTIs so that you can more easily control the results.
- Keep in mind that the purpose of this feature is to maintain a steady message rate for messages transferred from WSim, not to establish such a rate.

DELAY operand

Use the DELAY operand to define the delay between sending complete messages. You can specify a delay value that is a fixed value, two types of random values (uniformly distributed), or a value from a table of delays. You also specify the UTI to which the DELAY applies. The DELAY operand applies to the terminal going through message generation, but you can code it on higher-level statements. Refer to Part 2, "Introducing message generation decks," on page 93 for information about the DELAY message generation statement that you can use to override DELAYs set for terminals. Refer to *WSim Script Guide and Reference* for information about the STL DELAY statement that you can use to override DELAYs set for terminals.

RATE statement

If the type of delay that you specify is a table value, for example, DELAY=T0, you must code a RATE statement in your network definition to identify the rate table. The rate table is a member of the partitioned data set that is defined by the RATEDD DD statement in the WSim JCL for MVS. The integer name field on the RATE statement is used by the BLKDLY and DELAY operands and the DELAY and WAIT statements to reference a specific table. For more information about rate tables and how to generate rate tables, refer to Chapter 9, "Generating rate tables," on page 91.

THKTIME operand

Use the THKTIME operand to specify when WSim is to start the message delay. Use THKTIME=IMMED to specify that the delay for the next message for the terminal is to begin immediately after the current message is completed. The delay begins to expire even if the other conditions for entering message generation again, such as resetting a wait condition, have not been met. Use THKTIME=UNLOCK to specify that the delay for the next message for the terminal will not begin until the terminal is ready to generate a message. This means that all SNA responses arereceived, the terminal WAIT indicator is not set, and the keyboard is unlocked for a 3270 display. A delay specified by a WAIT statement is treated as if THKTIME=IMMED were specified.

CPI-C transaction programs always use THKTIME=IMMED, regardless of the value specified by the THKTIME operand.

If you code THKTIME=UNLOCK for SNA devices requiring exception responses only, there are no delays. Consider the sequence of message generation statements in the following example.

```
⋮
TEXT  (...)   FIRST TEXT STATEMENT
RH    EXC=ON  EXCEPTION RESPONSE REQUESTED
TEXT  (...)   SECOND TEXT STATEMENT
⋮
```

Since an exception response is requested on the first message sent in this example, there is no delay before the second message is sent even if you specify THKTIME=UNLOCK.

The following example shows how to code the message generation statements shown in the previous example using STL. For information about STL, see *WSim Script Guide and Reference.*

```
⋮
type '...'          /* First text statement. */
setrh on(exc)       /* Exception response requested. */
transmit
type '...'          /* Second text statement. */
⋮
```

Message logging

The message logging facility logs to a data set all data sent or received by a simulated device in a network. You can format this data using the Loglist Utility and use the output to debug message generation decks and network definitions. Message logging occurs automatically for all simulated devices in the network. To suppress message logging for all devices, code MLOG=NO on the NTWRK statement. To suppress message logging for individual resources, code MLOG=NO on VTAMAPPL, APPCLU,or TCPIP statements. You can override the NTWRK MLOG setting on these lower-level statements.

If you are simulating an IBM 3270 or IBM 5250, you can use the LOGDSPLY operand on the DEV or LU statement to specify that the display and printer buffers for the simulated device are automatically written to the log data set.

Separate log data sets for networks

Use the NTWRKLOG statement to specify a separate log data set for a network if you want to run multiple networks independently and analyze them separately later.

You can specify a NTWRKLOG data set where all data for the network is logged. If you do not specify a NTWRKLOG data set, all data for that network is logged to the general log data set. If it is not possible to associate some data with a specific network (as with operator commands), this data is logged to the general log data set. Operator commands, whether issued from the operator console or from an OPCMND statement, and their responses are always written to the general log data set.

The log record types that are written to the NTWRKLOG data sets include the XMIT, RECV, INFO, MTRC, MRKR, DSPY, LOG, STRC, CTRC, and VRFY records. CNSL records, which are the result of message generation deck WTO statements or are informational messages about the state of the network, are also written to the NTWRKLOG data set.

Include the NTWRKLOG statement in a network definition to specify that a network log data set is used for the network. Use one NTWRKLOG statement per network.

When using the NTWRKLOG statement, you also specify the number of buffers used to write to the log data set by coding a value for the NCP operand. You can specify the NCP value used in message logging in many places and on many log data set definitions. Below shows the order of precedence used to determine the NCP value on a specific log data set. The first value found is the one used on the log data set.

Precedence	**NCP Value**
1	NCP suboperand on DD statement
2	NCP operand on NTWRKLOG statement
3	NCP execution parameter
4	Default value 5

You can specify the maximum length of the data in each record of the log data set by using the MLEN operand on the NTWRK, APPCLU, VTAMAPPL, or TCPIP statements.

The DEBUG option

For Telnet 3270, 3270E, 5250, or NVT devices, specifying DEBUG causes WSim to log partial data buffers and Telnet options negotiations. For CPI-C simulations, if you specify DEBUG, WSim produces CPI-C VTAM trace data for APPCLUs.

Refer to *WSim User's Guide* for more information about how WSim logs data messages.

Inhibiting message logging to save space

WSim provides several methods that you can use to inhibit the logging of various messages in the log data set. These methods are described in the following sections.

Inhibiting the logging of specific messages

Normally, all Console messages (ITP001E - ITP399I) and all Log Data messages (ITP400I - ITP499I) are written to a log data set. Console messages are identified as CNSL records by the Loglist Utility. Log Data messages are identified as either INFO or MTRC records.

You might want to inhibit the logging of some of these messages. To inhibit these messages, use the INHBTMSG operand on the NTWRK statement or the MSGn= operand of the A (Alter) operator command. In addition to inhibiting the logging of messages, the INHBTMSG operand inhibits the display of these messages on the console as well.

You can use the INHBTMSG operand to inhibit one or more specific messages, a range of messages, or a class of messages as shown in the following example:

```
NETA   NTWRK   INHBTMSG=(6,124,177-179,ACT),...
```

The INHBTMSG operand in this example inhibits the display and logging of the following messages:

1. Individually named messages (6, 124)
   - ITP006I
   - ITP124I
2. A range of messages (177-179)
   - ITP177I
   - ITP178I
   - ITP179I
3. A class of messages (ACT)
   - ITP089I
   - ITP091I
   - ITP092I
   - ITP094I
   - ITP107I
   - ITP174I
   - ITP175I

Because you code the INHBTMSG operand within the network definition, you must know which messages you want to inhibit before the run. To inhibit a message during the run, you can use the A (Alter) operator command. Information about the A (Alter) operator command is provided in *WSim User's Guide*. For more information about the INHBTMSG operand and its syntax, refer to *WSim Script Guide and Reference*.

Network logic tests

This section provides a brief introduction to network logic testing. It discusses the coding you can use to create network logic tests and some of the situations in which you might want to use network logic tests. Refer to the *WSim Script Guide and Reference* for complete descriptions of the syntax for the logic test operands.

Refer to Part 2, “Introducing message generation decks,” on page 93 for information about message generation logic tests.

You can code network logic tests after the network definition to test all messages sent and received by a network. Network logic tests are activated when the network is initialized and are active throughout a simulation run. They are processed in the sequence in which you code them before any message generation logic tests.

Network logic testing is performed with network IF statements. You can code network IF statements in your network definition to test data received by WSim from the system under test or sent by WSim to the system under test. You can compare data and use the results to influence the path a device takes through message generation decks.

Note: Network logic tests are not valid for CPI-C transaction program simulations, and are ignored if specified.

What can be tested

Using a network logic test, you can test data from the following sources:

- Locations on a simulated screen
- Locations in the I/O buffer (the data stream)
- Counters
- Data in THs, RHs, and RUs in SNA simulations
- Locations in user areas and save areas.

For any of these data sources, you can specify an exact location for the logic test by using the LOC or LOCTEXT operand, or you can use the SCAN operand to look for data when you are not sure of its exact location. You can specify a starting point using the LOC or LOCTEXT operand and code SCAN=*nnnn* to scan for a specific number of characters. Or, you can code SCAN=YES to scan from the location you specify through the end of the data in that location. You can also specify a variable location by using the LOC or LOCTEXT operand and setting it to the value of a sequence or index counter.

Locations on a Simulated Screen

You can examine data in the screen buffer for a device that has a simulated screen using the LOC operand. Use LOC=B±*value* to specify a relative location in the screen buffer. B+*value* indicates a location starting from the upper left corner of the simulated screen. B-*value* indicates a location starting from the lower right corner of the simulated screen and moving backwards. To specify an exact location, set *value* to an integer 0 - 32766. To specify a variable location, set *value* to the name of a counter.

Use LOC=C±*value* to specify a position relative to the cursor. C+*value* indicates a position ahead of the cursor. C-*value* indicates a position behind the cursor. To specify an exact location, set *value* to an integer 0 - 32766. To specify a variable location, set *value* to the name of a counter.

Use LOC=(*row*, *col*) to specify an absolute position on the screen. You can also use counters for the *row* and *col* values, for example, LOC=(DC1,DC2).

You can also use the $RECALL$ data field option to access the areas on the TEXT or LOCTEXT operands.

Locations in the I/O Buffer (the Data Stream)

If you want to examine data for a device that has no screen buffer or one that uses an application such as CMS or TSO that sends data to the next available line, you can perform a logic test on the data in the I/O buffer. The I/O buffer contains the entire data stream sent to WSim, including all screen formatting characters for screen devices and the TH and RH for SNA devices. It does not include line control characters such as STX, ETX, or EOB.

Use LOC=D+*value* to specify a location in the data stream. To specify an exact location, set *value* to an integer 0 - 32766. To specify a variable location, set *value* to the name of a counter.

You can also use the $RECALL$ data field option to access the areas on the TEXT or LOCTEXT operands.

Counters

You can test sequence and index counters directly in network logic tests by using LOC=*value* or LOCTEXT=*value* to specify the name of a sequence or index counter.

Data in THs, RHs, and RUs in SNA Simulations

Use LOC=TH+*value*, LOC=RH+*value*, or LOC=RU+*value* to examine data in SNA THs, RHs, and RUs. To specify an exact location, set *value* to an integer 0 - 32766. To specify a variable location, set *value* to the name of a counter.

You can also use the $RECALL$ data field option to access the areas on the TEXT or LOCTEXT operands.

Locations in User Areas and Save Areas

You can compare responses returned by the system under test to data that was sent by WSim and saved in user or save areas.

Use LOC=N±*value* to specify a location in a network user area. Use LOC=N*x*±*value* to specify a location in a network-level save area, where *x* is the number of the save area. Use LOC=U±*value* to specify a location in a device user area. Use LOC=*s*±*value*, where *s* is the number of a save area, to specify a location in a device-level save area. To specify an exact location, set *value* to an integer 0 - 32766. To specify a variable location, set *value* to the name of a counter.

You can also use the AREA operand to specify a user area or save area that contains data to be compared. Use AREA=N±*value* to specify a location in the network user area. Use AREA=U±*value* to specify a location in the device user area. Use AREA=*s*+*value* to specify a location in a save area, where *s* is the number of the save area. Specify a length of data in a user area or save area named by the AREA operand by using LENG=*value*. To specify an exact length, set *value* to an integer 1 - 32000. To specify a variable length, set *value* to the name of a counter.

In addition, you can code LOCTEXT=(*data*) when data can include recalls of data from a user or save area.

When tests can be performed

By using the WHEN operand on the network IF statement, you can specify that the logic test is performed on incoming data (WHEN=IN) or outgoing data (WHEN=OUT). You can also use the TYPE operand on the network IF statement to specify that the network logic test is performed only for specific types of devices.

What comparisons can be made

You can use logic tests to compare the following types of data:

- Text in a response
- Text saved from previous responses
- Bits in the TH or RH of an SNA data stream
- User table entries.

Text in a Response
You can compare data to a string of text up to 32000 characters long. Use TEXT=(*text*) or LOCTEXT=(*text*) to specify a text string.

Text Saved from Previous Responses
You can save text data by using the RESP operand on a TEXT statement. You can code the response you are expecting for the TEXT statement on the RESP operand and save this response for comparison to the response that is received. To compare the response you receive with the response you were expecting, use TEXT=RESP in your logic test statement.

Bits in the TH or RH of an SNA Data Stream
You can compare data to bits in the TH or RH of an SNA data stream by coding two hexadecimal digits enclosed in quotation marks, for example TEXT='80'.

User Table Entries
You can compare data with all entries in a user table. Use the user table operand (UTBL) to specify the name or number of a user table. Do not use the TEXT operand.

What actions can be taken

When you compare with a network logic test, use the THEN operand to specify an action to be taken if the comparison is true and the ELSE operand to specify an action to be taken if the comparison is false. You can code either operand or both. Because network logic tests take place during message transmission and receipt rather than message generation, you do not have to code an ELSE operand because no action is required.

Some of the actions available for THEN and ELSE operands include branching to (B) or calling (C) another message path, waiting (WAIT), returning (RETURN) to message generation after the point of the last call, posting (POST) or signaling (SIGNAL) an event, or resetting (RESET) an event. Refer to *WSim Script Guide and Reference* for more information about the statements used to specify actions in network logic tests.

Online response-time statistics

The online Response-Time Statistics feature, RSTATS, provides online response-time calculations for simulated terminals. RSTATS measures the time it takes to enter a command at the terminal and receive a response from the system under test.

Online response time statistics are collected for message-generating resources only, such as DEVs and LUs.

Note: The RSTATS function is not supported for CPI-C transaction program simulations. RSTATS cannot be specified at the APPCLU or TP level. It is ignored for APPCLU and TP if specified at the NTWRK level.

The RSTATS operand

To specify the response-time feature for a particular device, specify the following codes in the network definition:

```
RSTATS=YES
```

You can code this operand on the DEV or LU statements or code it on higher-level statements and default it down to the lower-level statements.

RSTATS=YES specifies that response statistics are accumulated and reported when the operator issues the W (RSTATS) command for the device.

RSTATS=NO specifies that no response statistics are kept for the device.

If you specify RSTATS=NO for a device, you cannot activate RSTATS for that device at any time during the simulation because storage for RSTATS is allocated only at initialization time.

Refer to *WSim User's Guide* for information about how to operate the RSTATS feature.

Paths for message generation decks and STL programs

Use the PATH statement and the PATH operand on the DEV or LU statements to specify the order of message generation decks or STL procedures used by a simulated terminal.

Note: A group of STL procedures that perform a specific task is called an STL program.

The PATH statement specifies the sequence of message generation decks WSim uses for each terminal. Each PATH statement names one or more message generation decks. You can code any number of PATH statements for a network.

For each simulated device, code a PATH operand specifying the names of the PATH statements you have defined in the order you want the terminal to use them. For each device, you can specify any number of paths in any order.

When specifying paths for devices, use the CYCLIC=YES operand to specify that as terminals reference a PATH statement, the first terminal selects the first PATH entry, the second terminal selects the second PATH entry, and so on. If you do not specify CYCLIC=YES, you can use the DIST statement to define a probability to be used in selecting entries from a PATH statement.

You can use the FRSTTXT operand to identify a message generation deck that is used only once at the beginning of message generation. For example, you might want to identify a logon deck that is used once but not repeated as part of a path.

For more information about specifying paths for message generation decks, see Part 2, "Introducing message generation decks," on page 93. For more information about specifying paths for STL programs and for details about the statements and operands you can use to specify paths, see *WSim Script Guide and Reference*.

Random number generation

WSim maintains five random number generator seed values for each active network. These seeds are used to generate random numbers in five functional areas as described as follows:

- CNTRSEED specifies seed values used to set a counter to a random number.
- DELYSEED specifies seed values used to calculate delays or select them from rate tables.
- PATHSEED specifies seed values used to select message generation PATH entries according to a distribution specified on the DIST statement.
- TEXTSEED specifies seed values used to insert random numbers in message generation statements.
- UTBLSEED specifies seed values used to select entries from a user table.

These seeds are maintained separately for each functional area. For example, a random number generated as data on a TEXT statement does not affect the next PATH entry chosen based on a DIST statement. Changes in timings and the external environment can affect which particular random numbers are generated for a particular device, but the introduction into a network of a new functional area that uses random numbers does not change the sequence of random numbers generated in the other functional areas.

Specify the seed values for a network by using the five operands that are listed previously on the NTWRK statement. For more information about these operands, refer to *WSim Script Guide and Reference*.

You can generate random numbers between specific values by using the RN statement in your network definition. The RN statement defines an interval from which a random number is selected and inserted into a data field. You can reference a specific RN in a message generation statement when you want a random number to be generated with a message. In the following example, the RN statement specifies that each of the integers 10 - 50 has an equal chance of being selected when the RN is referenced by a message generation statement.

```
0  RN  LOW=10,HIGH=50
```

For information about referencing RNs in message generation statements, refer to Part 2, “Introducing message generation decks,” on page 93. For information about referencing RNs in STL statements, refer to*WSim Script Guide and Reference*.

How WSim generates random numbers

The random numbers generated by WSim are pseudo-random numbers. The method used to generate these numbers is based on the power residue method discussed in the publication *Random Number Generation and Testing*. When a random number is required during a simulation, WSim multiplies the current seed value for the functional area by 16777219. The low-order 48 bits of this product become the new seed value. This new value is used as the basis for selecting the wanted random number out of the specified range. A binary point is assumed to the left of the 48 bits of the value, thus giving a number between zero and one. This fraction is multiplied by the size of the specified range, and the truncated result is added to the lower limit of the range to produce the final random number.

Terminal scanning and automatic terminal recovery

WSim provides terminal scan and recovery functions that enable the operator to examine and alter the current status of terminals and devices in the network. The scan function examines each simulated terminal and device once every minute to determine whether they are active or inactive. A terminal or device is considered inactive if it meets the following criteria:

- No messages were transmitted or received for the time specified as the scan threshold (second parameter in the SCAN option).
- No interblock, intermessage, or intertransaction delay is active.
- The LOGICAL WAIT indicator is on, the terminal EVENT WAIT indicator is on, the INPUT INHIBITED indicator is on for display devices.
- The device type is not LU3.
- The terminal or device transmitted at least one message.
- The terminal or device is not quiesced.

Code the SCAN operand on the NTWRK definition statement to specify the scan function. Refer to *WSim User's Guide* for information about the operator commands you can use with the scan function.

When WSim finds a terminal or device that is inactive, it writes a message to the operator. Depending upon the third value coded on the SCAN operand, it might attempt to recover the terminal automatically. How that recovery proceeds is based on what you code on the ATRABORT operand for that terminal or device. Code ATRABORT=DECK to specify that the current message generation deck is to be aborted. Code ATRABORT=PATH to specify that the current path is to be aborted. Code ATRABORT=NONE to specify that the current message generation deck is not to be aborted. You can use the ATRDECK operand to specify a recovery message generation deck that is used before normal message generation is resumed.

Refer to *WSim User's Guide* for more information about controlling automatic terminal recovery. Refer to *WSim Script Guide and Reference* for details about coding the ATRABORT and ATRDECK operands.

Tracing messages and Structured Translator Language (STL) programs

This section describes message tracing and STL tracing. With message tracing, you can trace the message generation process during a test. With STL tracing, you can trace the execution of an STL program that WSim uses to generate messages during a test.

Message tracing

With the message trace feature, you can trace the message generation process by providing a printed listing of the steps that WSim follows through the message generation decks. To specify message tracing, code MSGTRACE=YES on the NTWRK, VTAMAPPL, APPCLU, TCPIP, DEV, LU, or TP statement. You can format the output using the Loglist Utility. For information about using the message trace feature to test and debug message generation decks, refer to Part 2, “Introducing message generation decks,” on page 93. For information about using the Loglist Utility to format message trace output, refer to *WSim Utilities Guide*.

Structured Translator Language program tracing

WSim provides an STL trace facility with which you can trace the execution of an STL program. STL is a structured language that you can use to create message traffic for devices simulated by WSim. STL statements are translated into message generation statements by the STL Translator. These statements form message generation decks that WSim uses to generate messages.

To specify the logging of STL trace records, code STLTRACE=YES on the NTWRK, VTAMAPPL, APPCLU, TCPIP, DEV, LU, or TP statement. The following example is an example of a LU statement with STL tracing specified:

```
LU0001  LU  STLTRACE=YES,...
```

You can also activate STL trace using the A (Alter) operator command. See *WSim User's Guide* for a description of this operator command.

Use the Loglist Utility to process the STL trace records. The Loglist Utility produces output that correlates the STL statements with the message generation statements produced by the STL Translator. The output also provides trace data indicating what occurs when the statements are processed.

To enable the Loglist Utility to process the STL trace records, you must include certain coding in your STL program or use a specific execution parameter when running the STL Translator. Refer to *WSim Script Guide and Reference* for more information about STL programs and using the Loglist Utility to process trace records.

User data tables

Use the UTBL statement to create a table from which data can be selected and inserted into messages, or to be referenced from FILE statements to define the simulated file data for FTP client simulations. A single UTBL can have up to 2147483647 entries, and each entry can contain any number of characters. Define the user tables with your network definition. You can define the user table data entries along with the user table definition or include them as a member of the partitioned data set that contains the message generation decks. Specify user table entries in message generation decks by using the MSGUTBL statement. The following example shows sample definitions of UTBLs and MSGUTBLs.

```
**************************************************************
*  UTBL definitions.                                         *
**************************************************************
0        UTBL  (0'F3'124),(04128),(34591),      PART NUMBERS
               (95735),(39782),(89678)
1        UTBL  (GIZMO),(SCREW),(NAIL),(WIDGET),
               (DOODAD),(STAPLE)                DESCRIPTIONS
2        UTBL  CATS                             MEMBER CATS
3        UTBL  DOGS                             MEMBER DOGS

:
**************************************************************
*  MSGUTBL definitions.                                      *
**************************************************************
CATS     MSGUTBL (DOMESTIC SHORT HAIR),(PUMA),(PANTHER)
DOGS     MSGUTBL (MUTT),(GREYHOUND),(SPITZ)
```

You can use the UDIST statement to define a probability distribution that WSim uses to select entries from a UTBL statement. The values you specify on a UDIST statement assign relative weights to the corresponding entries on the UTBL statement. Each weight is a proportion of the total of the weights on the statement. In other words, WSim divides each weight value by the total of the weights to obtain fractional values for each corresponding UTBL entry. One of these fractional values represents the probability that a particular UTBL entry will be chosen on any given selection.

For logic testing, you can reference a user table by number on an IF statement by using the UTBL operand.

Refer to Part 2, "Introducing message generation decks," on page 93 for information about referencing user tables and specifying user table data and distributions in message generation decks. Refer to *WSim Script Guide and Reference* for information about referencing user tables and specifying user table data and distributions in STL programs.

User exit routines

You can write user exit routines to perform additional processing of data and manipulation of network resources. When you specify an exit routine, that routine gains control when certain types of information are sent or received by WSim. You can use the following operands to indicate specific types of exit routines:

- Use the INEXIT operand on the NTWRK statement to specify a message input exit routine.
- Use the OUTEXIT operand on the NTWRK statement to specify a message output exit routine.
- Use the NCTLEXIT operand on the NTWRK statement to specify a network initialization, cancellation, or reset user exit routine.
- Use the UCMDEXIT operand on the NTWRK statement to specify an operator command user exit routine.
- Use the NETEXIT operand on the NTWRK statement to specify a network-level exit routine. You can also use the NETEXIT operand to specify a default exit routine for the INEXIT, OUTEXIT, NCTLEXIT, and UCMDEXIT operands.
- Use the INFOEXIT operand on the NTWRK statement to specify an informational message exit routine.
- Use the UXOCEXIT operand on the NTWRK statement to specify a user exit routine for an operator command issued by way of the user exit interface routine from another user exit routine.
- Use the EXIT message generation statement to specify a message generation exit routine.

Note:

1. For CPI-C transaction program simulations, only the message generation exit is available for use.

Refer to *WSim User Exits* for information about how to use the different types of user exit routines.

Chapter 9. Generating rate tables

Rate tables are used by WSim to calculate terminal delays. Rate tables are placed in a partitioned data set that can be referenced by RATE statements in a network definition. This partitioned data set is defined by the RATEDD DD statement in the JCL or EXEC used to run WSim.

The entries in the partitioned data set each have 1000 two-byte binary numbers that are accessed randomly, multiplied by the UTI value for the active device, and divided by 100 to determine the length of the delay for a terminal in hundredths of seconds. The sampling of entries from one of these tables produces terminal delays that are distributed according to the mean value specified for the table.

WSim provides several pregenerated rate tables. The installation JCL procedure creates the RATEDD data set and copies the supplied tables into it. For each supplied table, the member name is the letter R followed by 2 digits giving the mean of the distribution that the table represents. Thus, table R30 is an exponential distribution with a mean of 30 seconds when UTI=100. The tables that are supplied by WSim are: R10, R25, R30, R50, and R99.

If one of these tables does not meet your requirements, you can generate your own rate tables by defining the tables, putting them in a data set, and using the FORTRAN program, ITPRATEG, supplied in the WSIM.SITPSAMP file on the distribution tape to generate the tables. This appendix explains how to use this program to generate your own rate tables.

Creating input statements

ITPRATEG contains input data statements that specify a name and define the distribution and mean value for each rate table. The format of the input data statements is shown as follow.

Statement Columns	Field Contents
1-5	Upper limit of the sample distribution in hundredths of seconds (minimum 0, maximum 65535)
7-10	Population mean in hundredths of seconds (minimum 1000, maximum 9999)
12-19	Member name for the generated table (1 to 8 characters)

If you specify an upper limit of zero for the sample distribution, 655.35 seconds is assumed. An upper limit of 655.35 seconds is sufficient to allow consideration of at least 99 percent of an exponential distribution with a mean value of 99.99.

If you specify a population mean less than 1000, 10.0 seconds is assumed. The 1000 values in a table are always in the range from 0.01 to the upper limit value and they are generated from an exponential population with the mean that you specify. Specifying a small upper limit can produce a table whose 1000 entries produce a sample average lower than the population mean.

Generating the rate tables

To use ITPRATEG to generate rate tables, compile the FORTRAN source statements and link-edit the resulting object file with the ITPRATEW module (in the WSim object module library). The sample job shown in the following example creates a rate table generator load module and stores it in the WSim load module library under the name ITPRATET.

```
//RATE         JOB
//STEP1        EXEC FORTGCL
//FORT.SYSIN   DD DSN=WSIM.SITPSAMP(ITPRATEG),DISP=OLD,
//             UNIT=3380,VOL=SER=WSIMPK
//LKED.SYSLMOD DD DSN=WSIM.SITPLOAD,DISP=OLD,UNIT=3380,
//             VOL=SER=WSIMPK
//LKED.WSIMLIB DD DSN=WSIM.OBJCPU,DISP=OLD,UNIT=3380,
//             VOL=SER=WSIMPK
//LKED.SYSIN   DD   *
 INCLUDE WSIMLIB(ITPRATEW)
 NAME ITPRATET(R)
/*
//
```

The sample job shown in the following example creates a data set of rate tables with means of 10, 25, 30, 50, and 99, respectively.

```
//RATETAB  JOB  MSGLEVEL=1
//STEP1    EXEC PGM=ITPRATET
//STEPLIB  DD   DSN=WSIM.SITPLOAD,DISP=SHR
//RATEDD   DD   DSN=RATE.TABLES,DISP=(NEW,KEEP),UNIT=3380,
//              VOL=SER=WSIMPK,SPACE=(TRK,(5,5,5))
//FT06F001 DD   SYSOUT=A
//FT05F001 DD   *
    0 1000 R10
    0 2500 R25
    0 3000 R30
    0 5000 R50
    0 9900 R99
/*
```

The rate table generator module is then invoked to create the rate tables. Below describes the JCL and EXEC statements shown in the above examples.

Statement	Function
RATETAB JOB	Initiates the job
STEP1 EXEC	Specifies the name of the rate table generator
STEPLIB DD	Defines the WSim load module library
RATEDD DD	Defines the data set that will contain the new rate tables
FT06F001 DD	Specifies printer output of the rate tables
FT05F001 DD	Defines the input data statement data set

Figure 18 on page 93 shows the output produced when ITPRATEG processes the data input statement for the first rate table defined in the example.

```
WSIM RATE TABLE     R10
     POPULATION MEAN = 10.00     REQUESTED UPPER LIMIT = 655.35     SAMPLE MEAN = 10.07
     LOW VALUE = 0.01            HIGH VALUE =  113.13               VARIANCE =  107.6145     STANDARD DEVIATION =   10.3737
-----------------------------------------------------------------------------------------------------------------------------------
  0.01   0.02   0.03   0.04   0.05   0.06   0.07   0.08   0.09   0.10   0.11   0.12   0.13   0.14   0.15   0.16   0.17   0.18   0.19
  0.20   0.21   0.22   0.23   0.24   0.25   0.26   0.27   0.28   0.29   0.30   0.31   0.33   0.34   0.35   0.36   0.37   0.38   0.39
  0.40   0.41   0.42   0.43   0.44   0.45   0.46   0.47   0.48   0.49   0.50   0.51   0.52   0.53   0.54   0.56   0.57   0.58   0.59
  0.60   0.61   0.62   0.63   0.64   0.65   0.66   0.67   0.68   0.69   0.70   0.71   0.73   0.74   0.75   0.76   0.77   0.78   0.79
  0.80   0.81   0.82   0.83   0.84   0.86   0.87   0.88   0.89   0.90   0.91   0.92   0.93   0.94   0.95   0.97   0.98   0.99   1.00
  1.01   1.02   1.03   1.04   1.05   1.06   1.08   1.09   1.10   1.11   1.12   1.13   1.14   1.15   1.17   1.18   1.19   1.20   1.21
  1.22   1.23   1.24   1.26   1.27   1.28   1.29   1.30   1.31   1.32   1.34   1.35   1.36   1.37   1.38   1.39   1.40   1.42   1.43
  1.44   1.45   1.46   1.47   1.48   1.50   1.51   1.52   1.53   1.54   1.55   1.57   1.58   1.59   1.60   1.61   1.63   1.64   1.65
  1.66   1.67   1.68   1.70   1.71   1.72   1.73   1.74   1.76   1.77   1.78   1.79   1.80   1.82   1.83   1.84   1.85   1.86   1.88
  1.89   1.90   1.91   1.92   1.94   1.95   1.96   1.97   1.98   2.00   2.01   2.02   2.03   2.05   2.06   2.07   2.08   2.09   2.11
  2.12   2.13   2.14   2.16   2.17   2.18   2.19   2.21   2.22   2.23   2.24   2.26   2.27   2.28   2.29   2.31   2.32   2.33   2.34
  2.36   2.37   2.38   2.40   2.41   2.42   2.43   2.45   2.46   2.47   2.48   2.50   2.51   2.52   2.54   2.55   2.56   2.57   2.59
  2.60   2.61   2.63   2.64   2.65   2.67   2.68   2.69   2.70   2.72   2.73   2.74   2.76   2.77   2.78   2.80   2.81   2.82   2.84
  2.85   2.86   2.88   2.89   2.90   2.92   2.93   2.94   2.96   2.97   2.98   3.00   3.01   3.02   3.04   3.05   3.07   3.08   3.09
  3.11   3.12   3.13   3.15   3.16   3.17   3.19   3.20   3.22   3.23   3.24   3.26   3.27   3.28   3.30   3.31   3.33   3.34   3.35
  3.37   3.38   3.40   3.41   3.42   3.44   3.45   3.47   3.48   3.50   3.51   3.52   3.54   3.55   3.57   3.58   3.60   3.61   3.62
  3.64   3.65   3.67   3.68   3.70   3.71   3.73   3.74   3.75   3.77   3.78   3.80   3.81   3.83   3.84   3.86   3.87   3.89   3.90
  3.92   3.93   3.95   3.96   3.97   3.99   4.00   4.02   4.03   4.05   4.06   4.08   4.09   4.11   4.12   4.14   4.16   4.17   4.19
  4.20   4.22   4.23   4.25   4.26   4.28   4.29   4.31   4.32   4.34   4.35   4.37   4.38   4.40   4.42   4.43   4.45   4.46   4.48
  4.49   4.51   4.53   4.54   4.56   4.57   4.59   4.60   4.62   4.64   4.65   4.67   4.68   4.70   4.72   4.73   4.75   4.76   4.78
  4.80   4.81   4.83   4.85   4.86   4.88   4.89   4.91   4.93   4.94   4.96   4.98   4.99   5.01   5.03   5.04   5.06   5.07   5.09
  5.11   5.12   5.14   5.16   5.18   5.19   5.21   5.23   5.24   5.26   5.28   5.29   5.31   5.33   5.34   5.36   5.38   5.40   5.41
  5.43   5.45   5.46   5.48   5.50   5.52   5.53   5.55   5.57   5.59   5.60   5.62   5.64   5.66   5.67   5.69   5.71   5.73   5.74
  5.76   5.78   5.80   5.82   5.83   5.85   5.87   5.89   5.91   5.92   5.94   5.96   5.98   6.00   6.01   6.03   6.05   6.07   6.09
  6.11   6.12   6.14   6.16   6.18   6.20   6.22   6.24   6.25   6.27   6.29   6.31   6.33   6.35   6.37   6.39   6.41   6.42   6.44
  6.46   6.48   6.50   6.52   6.54   6.56   6.58   6.60   6.62   6.64   6.66   6.67   6.69   6.71   6.73   6.75   6.77   6.79   6.81
  6.83   6.85   6.87   6.89   6.91   6.93   6.95   6.97   6.99   7.01   7.03   7.05   7.07   7.09   7.11   7.13   7.15   7.17   7.19
  7.22   7.24   7.26   7.28   7.30   7.32   7.34   7.36   7.38   7.40   7.42   7.44   7.47   7.49   7.51   7.53   7.55   7.57   7.59
  7.61   7.64   7.66   7.68   7.70   7.72   7.74   7.77   7.79   7.81   7.83   7.85   7.87   7.90   7.92   7.94   7.96   7.98   8.01
  8.03   8.05   8.07   8.10   8.12   8.14   8.16   8.19   8.21   8.23   8.26   8.28   8.30   8.32   8.35   8.37   8.39   8.42   8.44
  8.46   8.49   8.51   8.53   8.56   8.58   8.60   8.63   8.65   8.67   8.70   8.72   8.75   8.77   8.79   8.82   8.84   8.87   8.89
  8.92   8.94   8.96   8.99   9.01   9.04   9.06   9.09   9.11   9.14   9.16   9.19   9.21   9.24   9.26   9.29   9.31   9.34   9.36
  9.39   9.42   9.44   9.47   9.49   9.52   9.54   9.57   9.60   9.62   9.65   9.68   9.70   9.73   9.75   9.78   9.81   9.83   9.86
  9.89   9.92   9.94   9.97  10.00  10.02  10.05  10.08  10.11  10.13  10.16  10.19  10.22  10.24  10.27  10.30  10.33  10.36  10.38
 10.41  10.44  10.47  10.50  10.53  10.56  10.58  10.61  10.64  10.67  10.70  10.73  10.76  10.79  10.82  10.85  10.88  10.91  10.94
 10.97  11.00  11.03  11.06  11.09  11.12  11.15  11.18  11.21  11.24  11.27  11.30  11.33  11.36  11.39  11.43  11.46  11.49  11.52
 11.55  11.58  11.62  11.65  11.68  11.71  11.74  11.78  11.81  11.84  11.87  11.91  11.94  11.97  12.01  12.04  12.07  12.11  12.14
 12.17  12.21  12.24  12.28  12.31  12.34  12.38  12.41  12.45  12.48  12.52  12.55  12.59  12.62  12.66  12.69  12.73  12.77  12.80
 12.84  12.87  12.91  12.95  12.98  13.02  13.06  13.09  13.13  13.17  13.20  13.24  13.28  13.32  13.36  13.39  13.43  13.47  13.51
 13.55  13.59  13.63  13.66  13.70  13.74  13.78  13.82  13.86  13.90  13.94  13.98  14.02  14.06  14.11  14.15  14.19  14.23  14.27
 14.31  14.35  14.40  14.44  14.48  14.52  14.57  14.61  14.65  14.70  14.74  14.78  14.83  14.87  14.92  14.96  15.01  15.05  15.10
 15.14  15.19  15.23  15.28  15.32  15.37  15.42  15.46  15.51  15.56  15.61  15.65  15.70  15.75  15.80  15.85  15.90  15.95  15.99
 16.04  16.09  16.14  16.19  16.25  16.30  16.35  16.40  16.45  16.50  16.55  16.61  16.66  16.71  16.77  16.82  16.87  16.93  16.98
 17.04  17.09  17.15  17.20  17.26  17.32  17.37  17.43  17.49  17.54  17.60  17.66  17.72  17.78  17.84  17.90  17.96  18.02  18.08
 18.14  18.20  18.26  18.33  18.39  18.45  18.51  18.58  18.64  18.71  18.77  18.84  18.90  18.97  19.04  19.10  19.17  19.24  19.31
 19.38  19.45  19.52  19.59  19.66  19.73  19.80  19.88  19.95  20.02  20.10  20.17  20.25  20.32  20.40  20.48  20.56  20.63  20.71
 20.79  20.87  20.95  21.04  21.12  21.20  21.29  21.37  21.45  21.54  21.63  21.71  21.80  21.89  21.98  22.07  22.16  22.26  22.35
 22.44  22.54  22.63  22.73  22.83  22.93  23.02  23.13  23.23  23.33  23.43  23.54  23.64  23.75  23.86  23.97  24.08  24.19  24.30
 24.42  24.53  24.65  24.77  24.89  25.01  25.13  25.26  25.38  25.51  25.64  25.77  25.90  26.04  26.17  26.31  26.45  26.59  26.73
 26.88  27.03  27.18  27.33  27.49  27.64  27.80  27.97  28.13  28.30  28.47  28.65  28.82  29.00  29.19  29.37  29.56  29.76  29.96
 30.16  30.36  30.57  30.79  31.01  31.23  31.46  31.70  31.94  32.19  32.44  32.70  32.97  33.24  33.52  33.81  34.11  34.42  34.73
 35.06  35.40  35.75  36.11  36.49  36.88  37.29  37.72  38.16  38.63  39.11  39.63  40.17  40.74  41.34  41.99  42.68  43.42  44.22
 45.09  46.04  47.09  48.27  49.60  51.14  52.96  55.18  58.05  62.09  68.96 113.13
```

Figure 18. WSim rate table output

Part 2. Introducing message generation decks

Chapter 10. Getting started with message generation decks

When you simulate system resources with WSim, you must code a script. A script consists of two parts: a network definition and message generation decks. As discussed in Part 1, “Defining WSim networks,” on page 1, the network definition is a set of network definition statements that define the lines, terminals, and devices you want to simulate. Message generation decks are sets of message generation statements that define the messages that flow from WSim to the system under test. The system under test responds to these messages just as it would to messages generated by actual network resources.

You can think of a message generation deck as a computer program that performs the actions of a terminal operator or a device on your system. For example, you can create a deck that simulates an operator entering commands at a terminal or that randomly selects inventory part numbers and descriptions to be sent to the system under test.

With message generation statements and their associated operands, you can control the message generation process. For example, you can generate text with message generation statements, and you can define logic tests, control events, and set intermessage delays. This book provides information about the following message generation topics:

- Planning message generation decks
- Coding message generation decks with message generation statements
- Integrating decks with the network definition
- Analyzing and debugging scripts with the Preprocessor and the postprocessors provided by WSim.

The following sections in this chapter explain the relationship between message generation decks and the network definition, describe how you can create a deck, and show what a deck looks like.

How do message generation decks relate to network definitions?

The network definition defines the terminals and resources you are simulating, and the options WSim uses for the lines, terminals, and devices that compose the simulated system. Message generation decks define messages sent to the system under test, control the timing of the message generation process, and define the interaction between simulated resources. After you create message generation decks, you code the network definition to associate each deck with one or more simulated devices.

The network definition determines the order in whichWSim processes each deck. Depending on the resources that you simulate, you can design a network definition and decks that WSim always uses together, or you can design generic decks that might be used with many different network definitions.

Creating message generation decks and coding network definitions can be an interdependent process. When you code message generation decks, you must also understand the network definition and the options coded for various devices. To create an efficient and effective simulation, both parts of a script must work together.

For more information about creating network definitions, see Part 1, “Defining WSim networks,” on page 1.

How can you create message generation decks?

WSim uses message generation statements in message generation decks to simulate message traffic. You can use one of the following three methods to create message generation decks:

- Structured Translator Language (STL)
- Message generation statements
- Script generating utilities.

Using the Structured Translator Language (STL)

STL is a high-level, structured programming language that uses constants, variables, expressions, and control structures as elements of its programs. The STL Translator is a utility that translates STL programs into message generation decks.

WSim Script Guide and Reference describes how to create decks with STL. Since STL is similar to other high-level languages, you might find it easier to use than message generation statements.

Using message generation statements

You can create message generation decks by coding the statements described in this book and in the *WSim Script Guide and Reference*. These two books describe the syntax conventions, coding requirements, and usage for message generation statements. You can also use message generation statements to modify decks created by any of the three methods listed in the preceding section. If you create decks with one of the script generating utilities, you need to understand message generation statements to debug the decks.

Using script generating utilities

WSim provides two utilities that you can use to generate decks:

- Interactive Data Capture
- Script Generator Utility

This section provides a brief introduction to each utility. See *WSim Utilities Guide* for detailed information about these utilities.

Interactive Data Capture (IDC) captures 3270 SNA data traffic between a host VTAM application and its associated logical unit (LU). IDC then converts the captured data to message generation decks or STL procedures.

The **Script Generator Utility** uses captured data traffic and network configuration data from live runs of the tested system. You can use existing trace capture programs or write your own to obtain the data. After you capture the data, use the programs provided with the Script Generator Utility to format the data; then, sort the data using any standard sort program. When the data is properly prepared, use ITPSGEN, a utility provided with the Script Generator Utility, to convert it into message generation decks.

111 The **TCP/IP Trace STL Generation Utility** uses data from a TCP/IP trace to
111 generate an STL program. The STL program replicates a client that is
111 communicating with a server that is running on z/OS. WSim also provides the

111 **TCP/IP Trace Utility**, which can establish a TCP/IP data trace to capture the
111 messages that are exchanged between a client and server.

What does a message generation deck look like?

A message generation deck is a collection of message generation statements that creates the messages WSim sends to the system under test. The following example illustrates a simple message generation deck that simulates an operator entering a password:

```
DECK1  MSGTXT
*                  Sample Message Generation Deck
*   This coding illustrates a simple message generation deck.
*
*                       Specifies the beginning of a message generation
*                       deck named DECK1.
*
MSG1   TEXT (PW125)     Defines a message named MSG1 to be sent to the
*                       system under test.  This message simulates an
*                       operator entering a password, PW125.
ENTER1 ENTER            Sets the ENTER AID byte.
       ENDTXT           Specifies the end of DECK1.
```

When WSim processes this deck, it generates and sends a message, that is, a password, to the system under test.

Although this example illustrates a simple deck created with only a few message generation statements, you can create more complicated simulations by adding other message generation statements. As you code a deck, each message generation statement performs a special function that contributes to the message generation process enabling you to accurately simulate message traffic between WSim and the system under test.

Chapter 11. Planning for message generation

This chapter discusses topics you should consider before you create message generation decks with message generation statements:

- Planning considerations
- Steps for developing scripts.

It also provides a checklist to guide you when you create scripts.

In general, the planning information in this chapter also applies to using STL and the script generating utilities. For specific planning information for these methods, however, refer to *WSim Script Guide and Reference* and *WSim Utilities Guide*.

For detailed information about all of the planning considerations that are related to tests, see *WSim User's Guide*.

Planning considerations

Before you create the message generation decks for a simulation, understand the objectives of the test. In other words, you need to know what you are trying to test and why you are testing it. This information determines the content of the decks. The test objectives might be stated in a test plan for your test; if they are not, consult your system planner, or refer to *WSim User's Guide* for more information about objectives for different kinds of tests.

Once you understand the objectives of your test, you can begin planning for the message generation decks. Planning at this stage includes making sure that you:

- Understand what you are testing
- Identify any external conditions that constrain messages
- Design your message generation decks
- Determine documentation procedures
- Determine testing procedures.

The following sections discuss each of these considerations in more detail.

Understanding what you are testing

The first step in creating message generation decks is understanding what you want WSim to simulate and how the simulated network interacts with the real system you are testing. The test plan probably includes general information about what you are testing; you need to supply specific details based on your knowledge of the network and the system you are testing.

Before you can create the message generation deck for this test, however, you need to review the test plan and your knowledge of the system to answer the following types of questions:

- What type of network are you simulating and what are its requirements?
- What types of terminals are you simulating?
- What types of messages do you want these terminals to send to the application?
- What types of messages do you expect to receive at the simulated resources?

- What timing is anticipated for messages to be received? Will they arrive on a set schedule or at unanticipated times, that is, asynchronously?

Determining the answers to these questions clarify the tasks that lie ahead. For example, your network might have specific timing or message load requirements, or you might want to simulate specific terminals or devices with particular restrictions. In addition, you might want to generate the same messages for each terminal, or you might need to have each terminal perform different tasks.

Identifying special requirements

A second step in planning for message generation is identifying what special requirements exist for the system. This section describes several special requirements that are commonly encountered in WSim.

When planning your message generation decks, you must ensure that you provide the information that is needed to connect the simulated network to real products. For example, if you are connecting to an application program, you must identify any special message or logon requirements for the application.

You also need to consider whether there are any special requirements for running the test. When an operator runs simulations from a console, the operator can start and stop various terminals, alter operating parameters, log messages, and monitor network activity, among other tasks. For details about operator capabilities, see *WSim User's Guide*.

You can also incorporate operator commands into your message generation decks so that the simulation can run independently of an operator. For example, you want to include operator commands in your script when you have an established test that you want to use to collect statistics continuously or when you have a long test that you want to run during off hours.

With WSim, you can also code messages in your deck to send to the operator to indicate test progress. Because this can be an important test-monitoring capability, be sure to plan for these messages when designing your message generation decks.

Finally, you need to consider whether there are any special requirements for using specific networks and devices. The type of message generation statements you use depends on what type of terminals and network you are using. For example, some statements are specific to display terminals and are ignored for other terminal types.

If you are using a VTAMAPPL simulation, you must define the VTAMAPPL in your system VTAMLST. Find out all the information that you can before you begin to code.

For more information about coding message generation decks for specific devices, see Chapter 18, “Generating messages for specific types of devices,” on page 219. For detailed information about coding the network definition for specific devices,see Part 1, “Defining WSim networks,” on page 1.

Designing message generation decks

The third step in planning for message generation is to design your message decks. When you design a deck, consider the following questions:

- How you want to structure the decks
- The content of the messages

- How you want to integrate the message generation decks with the network definitions.

The following sections explain each of these considerations in more detail.

Structuring message generation decks

A message generation deck can be as complex or as simple and as long or short as you want. The complexity and length of a deck depend on the objectives of your test. For example, if you want several terminals to run the same transactions in the same order, you decide to create one message generation deck that includes that sequence of transactions. If, however, you want these terminals to execute the transactions in a different order, you decide to create a different message generation deck for each transaction.

When designing message generation decks, define the structure and overall organization of all the decks before you code individual decks. Similarly, define the structure and overall organization of an individual deck before you code individual statements.

Determining message content

The content of the messages that WSim sends and receives depends on what you are testing. For example, if you want WSim to simulate a 3270 terminal accessing a banking application, typical messages sent to the application might include account numbers, money amounts, and check numbers.

As part of designing the message generation decks, you need to determine what the user would type to complete the transaction. This information might be actual data such as account numbers or text strings like “HELLO. HOW ARE YOU?” The data can also represent the type of information that is sent to the system under test when a user presses a specific key on the keyboard, such as the Tab or Enter key.

In WSim, you can create message generation decks that generate message text, use keyboard keys, specify intervals for sending messages, and act based on messages received. To code message content and keyboard activity, you must be familiar with the requirements of the application being tested and the resources being used. For example, if you are simulating panels, you need to know field locations for the application. If you are moving the cursor on the panel, you must know the cursor location and the keys that are used to move the cursor.

WSim provides considerable flexibility in designing messages. For example, you can set up a table that containsdata to simulate a database. You can then use data from the table in messages or compare data values from the table against messages received. Additionally, you can simulate keystrokes and cursor movements on display terminals.WSim also maintains a screen image display buffer so that you can test application panels.

Integrating message generation decks with network definitions

As you develop the structure of the message generation decks, you need to specify which terminals use which decks. All terminals can use the same set of decks or different terminals can use different decks. Message generation decks can be used in random order or in a predefined proportion. You use the PATH statement in the network definition to specify which decks are used for each terminal and the order in which they are processed.

“Selecting decks with the PATH statement” on page 248 describes how the PATH statement affects the message generation process. For detailed information about coding the PATH statement on the network definition, see Part 1, “Defining WSim networks,” on page 1.

Documenting message generation decks

A fourth step in planning for message generation is deciding how you document your message generation decks. Document your decks so that they can be maintained and used by others. You can use the comment header at the beginning of the message generation deck and the comment field in a message generation statement to indicate what the deck does and the logic that it follows. For more information about including comments in your message generation decks, see “Coding comments and the comment field” on page 110.

In addition, you or the test planner can maintain complete documentation of a test by keeping test plans with notations or specific descriptions about how the plan was carried out. Plans must indicate which message generation decks were used for particular tests and how the decks were combined with network definitions. Ideally, you might want to keep copies of your data sets, network definitions, and message generation decks with your test plans so that all documentation is together in one place.

With complete, accurate, and up-to-date test documentation, you can repeat tests easily. It also makes it easier to build upon existing network definitions and message generation decks when you modify tests or design new ones.

Testing scripts

A final step in planning for message generation involves determining the procedures that you will use to test the scripts. To ensure that the message generation decks are performing as anticipated, test your decks as you develop them.

Testing is a two-step process that involves:

1. Syntax testing
2. Function testing.

Syntax testing

To test the syntax of your message generation decks, you can use thePreprocessor. For more information about using the Preprocessor after you complete your script, see “Using the preprocessor” on page 257. For a complete description of how to use the Preprocessor, see *WSim Utilities Guide* .

However, correct syntax does not ensure that the script functions as expected. You also need to test whether your decks act as expected.

Function testing

To test the function of message generation decks, you compare records of actual message traffic with the message traffic you are expecting. To do this, you must combine a network definition and message generation decks into a script, as explained in Chapter 19, “Integrating decks with network definitions,” on page 247. Then, you can run a sample test and request message tracing. Finally, run the Loglist Utility to format these traces.

For more information about trace reports, see *WSim User's Guide*. For complete information about the Loglist Utility, see *WSim Utilities Guide*.

Developing scripts

As discussed in Chapter 10, "Getting started with message generation decks," on page 95, message generation decks relate closely to the network definition. Coordinate the coding of the network definition statements and the message generation decks, especially if you are coding complex scripts.

Before you create the message generation decks for your test, however, spend some time writing test scripts in commonplace English sentences. These test scripts must detail all the steps that you are testing. Writing these test scripts in English helps ensure that you completely understand the steps before you spend time creating message generation decks.

It is best to begin by developing the network definition and message generation decks on a small scale. In the long run, this simplifies development of complex simulations and enables you to correct errors and misunderstandings in a smaller environment before moving to a larger simulation.

In general, begin with a network consisting of a limited number of terminals, and create a simple message generation deck. Then, you can integrate the network definition with the deck to create a script. When you get this script to run correctly, you can gradually add more network definition statements and more message generation decks.

For example, you might begin by defining one terminal of the type you will simulate. Then, write a message generation deck that logs the terminal on to the application and then sends and receives a message. When your script accomplishes this task successfully, add more message generation decks so that the terminal can perform more tasks. After you have one terminal functioning successfully, add more terminals.

Continue testing your script in an iterative fashion, changing either the network definition or message decks each time. This process makes it easier to identify errors if they occur.

Checklist for creating message generation decks

The following checklist presents steps that you can follow to create message generation decks. The first two steps help you identify the types of resources and messages you will be simulating, and special requirements for your system. The third and fourth steps list considerations related to designing and documenting your message generation decks. The remaining steps describe procedures that can help you code and test your scripts efficiently.

Identify resources and messages:

1. Understand what you are testing by reviewing the test plan and your system, and talking to other staff members.
   - What type of network are you simulating and what are its requirements?
   - What types of terminals are you simulating?
   - What types of messages do you expect to receive at the simulated terminals?
   - What timing is anticipated for messages to be received? Will they arrive on a set schedule or at irregular times?
2. Identify special requirements for your system.

- Do the message generation decks or the operator interact with the system? If the operator does, what actions should be taken?
- Does the operator need messages sent to the console to monitor the test's progress?
- Are there specific devices on the network or system under test that needs special identification or protocols?

Design and document your decks:

3. Design your message generation deck.
   - How should the decks be structured?
   - What is the message content of the decks?
   - How should the message generation decks and the network definition be integrated?
4. Document the message generation deck in the header and in comment fields.

   Code and test your scripts:
5. Run a sample script through the Preprocessor to test the syntax of the network definition and message generation statements.
6. Perform a test run using the sample script to make sure the simulation functions as you intended.
7. Continue coding message generation decks one by one and adding network components in small groups.

Part 3. Coding message generation statements

Chapter 12. Basic concepts

To simulate message traffic between WSim and the system under test accurately, you need to understand the message generation statements that make up a message generation deck. This chapter describes the basic concepts you must understand before coding message generation statements:

- Using appropriate syntax conventions
- Understanding WSim coding requirements
- Coding the MSGTXT and ENDTXT statements
- Coding a sample message generation deck with the TEXT, WAIT, and IF message generation statements.

This chapter also introduces ways that you can alter the message generation process by coding delimiters, logic tests, and control statements, which are the three basic types of message generation statements.

Note: Only one terminal can be in message generation at any one time and message generation cannot be interrupted.

Syntax conventions for message generation statements

To code message generation statements, you must use WSim syntax. Each statement can include four fields, some of which are optional:

- Name
- Statement
- Operand
- Comment.

As shown in the following example, you must separate these fields with at least one blank space and code them in the following order as required by WSim:

```
Name    Statement    Operand     Comment
```

You can easily recognize each field by the order in which it appears in the examples shown in this book.

The following example shows a statement coded in the required format:

```
OPMSG1  WTO  (PROCESSING COMPLETE)  Message to operator.
```

This statement contains the following four fields:

Name	OPMSG1, which specifies the name for the statement.
Statement	WTO, which specifies the action to be performed; in this case, write a message to the system operator (WTO specifies write to operator).
Operand	(PROCESSING COMPLETE), which specifies text for the message being sent to the console operator.
Comment	Message to operator, which explains the purpose of the statement.

Note: All of the examples in this book show message generation statements in upper case. This is not necessary. You may code message generation statements in mixed case.

Coding the name field

For most message generation statements, you do not need to code a statement name in the name field. If you want to refer to a statement later in the same message generation deck, however, you will want to code a name for that statement.

WSim syntax requires that you begin the name field in column 1 and code 1 - 8 alphanumeric characters. Because WSim recognizes a blank space as the end of a field, do not use blank spaces in the name field. For example, WSim would not recognize FIX NOW as a statement name even though it has 7 characters.

Note: WSim requires that you enter a number in the name field for certain IF message generation statements. The number you enter determines the order in which WSim evaluates IF statements that perform logic testing on messages sent to and received from the system under test. For more information about the IF statement, see Chapter 16, “Defining logic tests,” on page 165.

Coding the statement field

A statement field contains a valid WSim statement, which can be in mixed case. Unlike the name field, which is optional, you must code the statement field for each message generation statement. The statement field identifies the action to be taken by a simulated resource. To specify a statement name, separate the statement field from the name field by at least one blank space. If you do not specify a name in the name field, you can begin the statement in any column following column 1.

All message generation statements are listed alphabetically in the *WSim Script Guide and Reference*.

Coding the operand field

The operand field specifies the name of one or more operands that qualify a statement's action. If you specify an operand field, separate it from the statement field by at least one blank space. Because WSim recognizes a blank space as the end of a field, do not include blank spaces within or between operands. Separate multiple operands with commas.

For certain message generation statements, you must code only the name of an operand in the operand field. For example, the SCROLL statement might include the operand DOWN, as shown in the following example:

```
MOVE1   SCROLL   DOWN    Resource scrolls down.
```

In WSim, however, some statements do not require an operand in the operand field:

- Some statements do not have an associated operand.
- Some statements require only text in the operand field.
- Some statements require an operand name followed by a value that qualifies the operand.

The following section describes the syntax for coding text in the operand field.

Entering message text

The operand field can include text enclosed within text delimiters. Text delimiters are characters that identify the contents as text. On the TEXT statement, for example, you can code the operand field with a message you want to send to the system under test. You can also code text for the WTO statement to define the message thatWSim sends to the operator console.

The following example shows message text entered on a TEXT statement and a WTO statement.

```
MSG1   TEXT (INVALID PASSWORD)      Message to system under test.
WTOMSG WTO  (PROCESSING COMPLETE)   Message to console.
```

As shown in this example, the default text delimiters, the right and left parentheses, enclose each message. You can change the default text delimiters by coding different characters on the MSGTXT statement's TXTDLM operand. For more information about the MSGTXT statement and the TXTDLM operand, see “Coding the MSGTXT and ENDTXT statements” on page 111.

Note: You can enter uppercase or lowercase text within text delimiters. The examples in this book, however, show uppercase message text to help you distinguish it from text entered in the comment field.

When you code messages in the operand field, enter at least one blank space after the statement field and then enter the text enclosed by text delimiters. You can include blank spaces within the data enclosed by text delimiters.

There are several ways to code a message that extends over more than one line. You can code text through column 71 and then begin the remaining text in column 2 of the following line, as shown in the following example:

```
Column                                                            Column
1                                                                     71
        TEXT          (THIS IS A MESSAGE TO BE CONTINUED TO THE NEXT LI
 NE USING THE FIRST CONTINUATION METHOD.)
```

Note: WSim truncates any data that extends past column 71.

As shown in the next example, you can also enter segments of messages on several lines. You must enclose each segment within text delimiters, and each line containing continued text must end with a comma or a plus sign.WSim concatenates all three message segments into one message when processing the statement.

```
MSG1  TEXT  (ENTER PASSWORD),  Message to system under test.
            ( TO COMPLETE ),
            (LOGON)
```

You can also continue message text on the TEXT statement by coding MORE=YES. When you use this operand, WSim concatenates the text on two successive TEXT statements, enabling you to build messages with variable data in certain fields.

Entering operands and operand values

The operand field may include operand values in addition to the operand. In the following example, the COUNT operand on the MSGTXT statement is coded with the numeric value 20 and the TEXT operand on the IF statement is coded with the text value READY.

```
DECK1 MSGTXT  COUNT=20               WSim processes DECK1 20 times.
1     IF      TEXT=(READY),WHEN=IN,  Defines logic test.
              LOC=RU+0,THEN=CONT
```

To code operands with associated values, first enter the operand name; then, enter an equal sign followed by text or numeric values extending through column 71. Remember WSim truncates any data that extends past column 71. Do not code blank spaces around the equal sign or within the qualifying value.

Note: In the preceding example, the number 1 appears in the IF statement's name field. WSim requires that you enter a number in the name field of IF statements coded with the operands WHEN=IN and WHEN=OUT.

When you code multiple operands on a statement, use a comma to separate operands and to show that additional operands appear on the following line:

```
GO1 CALL  LABEL=MOVE2,    Next statement to be executed.
          NAME=DECK5      Name of deck containing statement named MOVE2.
ADD CALC  LOC=N+5,        Addition calculation.
          VALUE=+500
```

Do not include blanks between operands.WSim uses a blank space to distinguish between fields within a statement.

Coding comments and the comment field

The comment field provides supplemental information about the statement or operand, such as describing the action or procedure to be followed by the simulated resource. Although message generation statements do not require comments, you may want to use comments often to clarify the action or procedure represented by the statement.

The following example shows how to code comments in the comment field:

```
LOGMSG  LOG  DISPLAY  Writes the display buffers to the log data set.
```

You can code comments in the comment field in one of the following situations:

- The statement has no operands
- The statement has operands and at least one of them has been coded.

Note: If you do not code operands for a statement that has allowable operands, do not code a comment in the comment field. WSim attempts to interpret comments entered in this way as an operand.

Comments can also be coded on a separate line when that line begins with an asterisk (*) in column 1:

```
WAIT1   WAIT
* This is a comment for the preceding WAIT statement.
```

You can also use comments to create a header for your script. By entering an asterisk followed by information that describes your coding, you can provide extra documentation for your scripts. The figures in this chapter illustrate message generation decks with headers. When you create these headers, however, be sure to code them after the MSGTXT statement or the Preprocessor will delete them.

Basic message generation statements

When you code a message generation deck, you combine a set of message generation statements according to the syntax and coding conventions discussed in the preceding sections. While coding statements with the appropriate syntax, you place them in a particular sequence depending on the messages you want the simulated resources to generate.

Just as WSim requires specific syntax and coding conventions for message generation statements, you create message generation decks according to specific coding conventions. For example, each message generation deck must begin with an MSGTXT statement and end with an ENDTXT statement. Between these two statements, you place message generation statements that define messages to be generated by the resources WSim simulates.

In addition to the MSGTXT and ENDTXT statements required by WSim, you can code many different statements in each message generation deck. Although many message generation statements are listed in the *WSim Script Guide and Reference*, you can use three basic statements to code simple, yet productive decks:

- TEXT
- WAIT
- IF.

These three message generation statements provide the basis for creating complex simulations. As you learn about each statement and understand how they relate to each other, you can soon write message generation decks that simulate many types of terminal sessions.

This chapter briefly describes the basic message generation statements. For more information about using these statements, see the following chapters:

- Chapter 13, “Generating messages with the TEXT statement,” on page 117
- Chapter 14, “Understanding delimiters,” on page 137
- Chapter 15, “Understanding intermessage delays,” on page 153
- Chapter 16, “Defining logic tests,” on page 165.

Coding the MSGTXT and ENDTXT statements

The following example illustrates the basic format of a message generation deck, including the two message generation statements required in WSim, the MSGTXT and ENDTXT statements. It also illustrates the coding conventions used for examples in this book:

- Vertical dots represent parts of a deck not shown in a particular example.
- Lowercase, italicized words and letters represent variables for which you can enter specific information.

```
name      MSGTXT   operand
*
*                     Sample Message Generation Deck
*   This message generation deck begins with the MSGTXT statement.
*   The vertical dots represent additional statements that make up the
*   body of the deck; the ENDTXT statement marks the end of the deck.
*
*                            WSim requires the name field on the MSGTXT
*                            statement.
*
          .                  All other message generation statements appear
          .                  between the MSGTXT and ENDTXT statements.
          .
*
         ENDTXT              The ENDTXT statement marks the end of the deck.
```

Each message generation deck begins with the MSGTXT statement. Unlike operands on other message generation statements, operands on the MSGTXT statement perform important functions that affect *every* statement in the message generation deck. The following list shows some of the possible operands.

CONCHAR	Specifies the control character you are using as a text delimiter for data field options in the current deck. The default CONCHAR is the dollar sign ($).
COUNT	Specifies the number of times WSim processes a deck during message generation before selecting and processing another deck. The default is 1.
PAD	Specifies the character you are using to pad generated messages to a certain length. The default is the alphabet.
TXTDLM	Specifies the text delimiter you are using to separate messages from other fields in the current deck. The default text delimiters are the left and right parentheses ().

WSim requires that you specify a unique name in the MSGTXT name field. Because WSim references the deck by the name you specify, assign a different name to each message generation deck, even if you are using decks for different resources. The following example illustrates coding for the MSGTXT statement:

```
DECK1  MSGTXT  CONCHAR=#    Specifies that you are separating data field
*                           options with a # symbol.
```

When you code a message generation deck, the ENDTXT statement specifies the end of the deck. The ENDTXT statement causes WSim to stop processing the current message generation deck and begin processing another. The ENDTXT statement has no operands associated with it.

To code the ENDTXT statement, complete the required fields in the appropriate syntax, as shown in the following example:

```
          ENDTXT         Specifies the end of a message generation deck.
```

Note: You cannot nest MSGTXT and ENDTXT statements. Each MSGTXT signals the beginning of a deck; each ENDTXT signals the end of a deck.

For additional information about the MSGTXT statement, its operands, and the ENDTXT statement, see the *WSim Script Guide and Reference*.

Coding the TEXT statement

The TEXT statement defines the messages that simulated resources transmit to the system under test. For example, to simulate an operator logging on to the system under test, you would complete the required TEXT statement fields in the appropriate syntax, as shown in the example below.

```
DECK1   MSGTXT
*                   Sample Message Generation Deck
*   This message generation deck begins with the MSGTXT statement
*   and ends with the ENDTXT statement.  The TEXT statements
*   make up the body of the deck.
*
LOGON1  TEXT  (ID1523)   System user ID to be sent from simulated
*                        resource to system under test.
LOGON2  TEXT  (PW1523)   System password to be sent from simulated
*                        resource to system under test.
        ENDTXT           Sends LOGON2 and specifies end of DECK1.
```

This message generation deck is complete. The MSGTXT and ENDTXT statements mark the beginning and end of the deck as required by WSim, and the TEXT statements define data to be sent to the system under test.

The first text statement, however, does not actually cause WSim to send the message. When WSim processes a TEXT statement, it stores the message in the terminal buffer. This message remains in the buffer until another statement causes

WSim to send the message. In the sample message generation deck, WSim places the first message, ID1523, in the terminal buffer when it processes the first TEXT statement. The second TEXT statement causes WSim to send the first message and interrupts message generation. When the device reenters message generation, WSim places the second message, PW1523, in the terminal buffer. Then WSim processes the ENDTXT statement, which sends the second message to the system under test and ends this pass through message generation for the device.

You can also code a WAIT statement in addition to the TEXT statements in the sample deck. As discussed in the following section, a WAIT statement also causesWSim to send messages to the system under test.

Coding the WAIT statement

The WAIT statement simulates the action of a terminal operator waiting for a reply before entering the next message. When WSim encounters a WAIT statement in a message generation deck, it transmits any message in the terminal buffer and suspends message generation for a specified length of time or until a specified event occurs.

You can use a WAIT statement to prevent a device from reentering message generation until an expected response is received from the system under test. The following example illustrates the sample message generation deck with the addition of a WAIT statement.

```
DECK1   MSGTXT
*                  Sample Message Generation Deck
*   This message generation deck begins with the MSGTXT statement
*   and ends with the ENDTXT statement.  The TEXT and WAIT statements
*   make up the body of the deck.
*
LOGON1  TEXT    (ID1523)  System user ID to be sent from simulated
*                         resource to system under test.
WAIT1   WAIT
*                         Sends LOGON1 to the system under test
*                         and suspends message generation.
LOGON2  TEXT    (PW1523)  System password to be sent from simulated
*                         resource to system under test.
        ENDTXT            Sends LOGON2 to the system under test.
```

The WAIT statement causes WSim to send the message defined by LOGON1 to the system under test and to set a WAIT indicator that delays further processing of the message generation deck. WSim will not process the message defined by LOGON2 until you take an action that resets the WAIT indicator.

To reset the WAIT indicator and resume processing of the message generation deck, you can code an IF statement, as discussed in the following section.

Coding the IF statement

In the sample message generation deck, the WAIT statement sends the first message and stops further processing of the deck. You can reset the WAIT indicator and continue processing the deck by adding an IF statement to the sample deck.

The IF statement causes WSim to perform a specified logic test. For example, you might want to continue processing a deck when you receive a specific message from the system under test. As shown in the example below, you can code an IF statement to reset the WAIT indicator when the terminal receives the message ENTER PASSWORD.

```
DECK1   MSGTXT
*                    Sample Message Generation Deck
*   This message generation deck begins with the MSGTXT statement and
*   ends with the ENDTXT statement.  The TEXT, IF, and WAIT statements
*   make up the body of the deck.
*
LOGON1  TEXT    (ID1523)               System user ID to be sent from simulated
*                                      resource to system under test.
1       IF      TEXT=(ENTER PASSWORD), Sets up a logic test for a reply
                LOC=RU+0,SCAN=YES,     to LOGON1.
                THEN=CONT
WAIT1   WAIT
*                                      Sends LOGON1 to the system under test
*                                      and suspends message generation.
LOGON2  TEXT    (PW1523)               System password to be sent from simulated
*                                      resource to system under test.
        ENDTXT                         Sends LOGON2 and specifies end of DECK1.
```

The operands on the IF statement tell WSim to continue processing the message generation deck when the terminal receives the message ENTER PASSWORD:

TEXT=(ENTER PASSWORD)	Specifies the text data WSim uses in the comparison.
LOC=RU+0	Specifies the starting location at which the comparison takes place.
SCAN=YES	Specifies that WSim scans each data stream received by the terminal sequentially for the data specified by the TEXT operand.
THEN=CONT	Specifies that message generation continues after the terminal receives the specified data and resets the WAIT indicator when the expected message is received.

Note: If the message ENTER PASSWORD is never received after WSim processes WAIT1 in the preceding example, message generation will never continue.

Note that you code the IF statement before the WAIT statement. This is because the WAIT statement suspends message generation. If you code the IF statement following the WAIT statement, WSim stops processing the deck before reading the IF statement. Subsequently, there is no way to reset the WAIT indicator. For more information about the WAIT indicator, see “Understanding the WAIT indicator” on page 144.

When WSim processes the sample deck, it sends “ID1523” to the system under test and waits for the appropriate reply, “ENTER PASSWORD”. After receiving the reply, WSim sends “PW1523” to the system under test.

Figure 19 on page 115 shows the actions that take place when WSim processes each statement on the sample deck and the timing of the messages going to the system under test.

```
Statement        Action
---------        ------
TEXT             Put message into terminal buffer (ID1523).

IF               Activate IF, searching for ENTER PASSWORD.

WAIT             Transmit buffer contents and set WAIT indicator.

Simulated                  ID1523                        VM
Terminal  ─¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦→ System

Simulated              ENTER PASSWORD                    VM
Terminal  ←¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦─ System

                 IF evaluation and action taken:  reset WAIT (CONT).

TEXT             Put text data into terminal buffer (PW1523).

ENDTXT           Transmit buffer contents.

Simulated                  PW1523                        VM
Terminal  ─¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦→ System

                 Continue with next message generation deck.
```

Figure 19. How WSim processes the sample message generation deck

The following section describes how different types of message generation statements are classified.

Classification of message generation statements

To create message generation decks, you code message generation statements that define messages for WSim simulated resources. In the preceding sections, you learned about the message generation statements that WSim requires in each deck, the MSGTXT and ENDTXT statements, and optional statements like the TEXT, WAIT, and IF statements. In addition, WSim provides many other message generation statements that help you code even the most complex message generation deck.

Each statement can be classified into the following three categories:
- Delimiters
- Logic tests
- Control statements.

The following sections introduce each type of statement and discuss how you control the message generation process by coding them in your message generation decks.

Understanding delimiters

When you create a deck, the TEXT statement defines the message you want to send but does not send the message. To send a message stored in the terminal buffer, you can code one of the unconditional delimiters: the WAIT, STOP, or QUIESCE statement. An unconditional delimiter always interrupts the message generation process and sends a message stored in the buffer.

WSim also provides conditional delimiters, statements that act as delimiters under certain message generation conditions or during a specific type of simulation. For example, a TEXT statement acts as a delimiter if WSim already has a message in the terminal buffer and if the MORE operand was not coded on the previous TEXT statement. The CTAB statement acts as a delimiter during 3270 simulation.

For more information about each type of delimiter and about using delimiters to interrupt message generation, see Chapter 14, "Understanding delimiters," on page 137.

Understanding logic tests

The IF message generation statement enables you to alter the message generation process by testing counter values, switch settings, messages sent or received by a simulated terminal, and the status of events. You can also code IF statement logic tests that evaluate messages from the system under test to determine whether the system is operating correctly.

WSim provides two types of logic tests: network-level logic tests and message-level logic tests. You define network-level tests by coding IF statements in the network definition. You define message-level tests by coding IF statements in the message generation deck. Network-level tests remain active throughout the message generation process, unlike message-level tests which may be deactivated. In addition, WSim evaluates network-level tests for all devices in the network while message-level tests apply only to devices that execute the message generation deck that contains the tests.

Chapter 16, "Defining logic tests," on page 165 provides a complete discussion of the IF message generation statement and message-level logic tests. For detailed information about network logic tests, see Part 1, "Defining WSim networks," on page 1.

Understanding control statements

Unlike delimiter and logic test statements, control statements do not interrupt or alter the message generation process. When WSim encounters a control statement, it executes the statement immediately and then continues processing the next statement in the deck. These statements perform varied functions in message generation decks.

See Chapter 17, "Understanding control statements," on page 195 for a complete description of the control statements provided by WSim.

Chapter 13. Generating messages with the TEXT statement

In WSim, you define messages for simulated resources by coding data on the TEXT statement. As discussed in Chapter 12, “Basic concepts,” on page 107, WSim reads the message during message generation and places it in the terminal buffer. When WSim encounters a message generation statement that serves as a delimiter, it sends the message to the system under test.

This chapter describes using the TEXT statement to generate messages in the following ways:

- Manually
- Dynamically with data field options
- Generating random numbers
- Using data from user table entries
- Using data with sequence and index counters
- Using data from save and user areas.

In special situations, you also define data in messages with the CMND and STRIPE message generation statements. For more information about generating messages in such situations, see Chapter 18, “Generating messages for specific types of devices,” on page 219.

Note: If a TEXT statement is specified in a message deck that is part of a transaction program (TP) it is ignored. Transaction program messages are generated using CPI-C verb statements.

Generating messages manually

To generate messages manually for a simulated resource, you enter data in the TEXT statement data field. You can enter either text or hexadecimal characters, specifying each character you want to send in your message. The following sections describe how you code each type of data and how you can combine types of data in one statement.

Entering data

You code message data within text delimiters, which by default are the left and right parentheses. The following example illustrates the coding for text and hexadecimal characters:

```
MSG1  TEXT  (ENTER YOUR PASSWORD)  Message with text.
MSG2  TEXT  ('AB')                 Message with hexadecimal characters.
```

Note: Hexadecimal data must be enclosed within single quotation marks. See “Simulating DBCS data entry for simulated 3270 DBCS terminals” on page 241 for more information about entering DBCS data.

Combining types of data

Although you will usually code messages by entering text in a single format, you can code a message in a combination of both types of data. In the following example, the message “ENTER PASSWORD” appears in text, hexadecimal characters, and a combination of both types of data:

```
MSG5 TEXT (ENTER PASSWORD)                 Message in text.
MSG6 TEXT ('C5D5E3C5D940D7C1E2E2E6D6D9C4') Message in hexadecimal characters.
MSG7 TEXT ('C5D5E3C5D9' PASSWORD)          Message in text and hexadecimal
*                                          characters.
```

In MSG6, notice that the blank space between ENTER and PASSWORD appears as hexadecimal X'40'. If you do not code a blank space in this position, WSim sends the message “ENTERPASSWORD”. To avoid processing errors, remember to enter spaces in your messages, whether you are coding text or hexadecimal characters.

WSim recognizes everything coded within the text delimiters as part of the message.

Generating messages dynamically

When you create a message generation deck, you might not want to code a message manually for every message sent to the system under test. For example, you might have several fields in each message, each field containing one of several possible values. Instead of coding many statements for each possible value, simplify the coding process by generating messages dynamically with data field options. A data field option is a value you code in a data field to retrieve data that WSim can access.

You can code data field options to generate different kinds of messages dynamically for many different simulated resources. For example, you code some data field options only on the TEXT statement; you code others on statements such as the WTO, CMND, and LOG statements. WSim also provides options you code only when you simulate a specific resource.

“Understanding data field option use” on page 119 provides a discussion of the different types of data field options available with WSim. For a complete list of data field options and information about coding requirements, see the *WSim Script Guide and Reference*.

Understanding data field option syntax

To code a data field option, you enclose the option within a special control character that you can define with the CONCHAR operand on the MSGTXT statement. In the following example, the $DATE$ data field option appears within dollar signs, the default value for the CONCHAR operand:

```
MSG1   TEXT   ($DATE$)       Data field option for the current date.
```

When WSim processes the TEXT statement in the preceding example, it inserts the current date into the message as EBCDIC characters. By default, WSim formats the date as MMDDYY: month, day, and year. For example, January 2, 1996 appears as 010296. (Refer to *WSim Script Guide and Reference* for a complete list of format options for this data field option.)

Note that the data field option in the preceding example appears within the default text delimiters, the left and right parentheses. If you change CONCHAR or TXTDLM on the MSGTXT statement, do not code the same character on both operands. For example, if you use parentheses as text delimiters, you cannot use parentheses as control characters for data field options in the same deck.

The following example shows how you change the control character with the MSGTXT statement:

```
DECK1  MSGTXT   CONCHAR=#   Specifies the # symbol as the data field
*                           option control character.
MSG1   TEXT     (#DATE#)    Data field option for the current date.
```

As a convention for the examples in this book, data field options appear enclosed within the default control character, $.

Understanding data field option use

To understand the data field options available in WSim, the options differ from one another in the way they generate messages. Some options refer to information WSim maintains internally, such as the current month and time. As shown in the following example, the $MONTH$ and $YEAR$ data field options insert the 2-digit numbers for the current month and year directly into a message:

```
MSG10  TEXT  ($MONTH$)            Message to be generated.
MSG11  TEXT  (ENTER PASSWORD ),   Message to be generated.
             (FOR $MONTH$/$YEAR$)
```

When WSim processes MSG10 for May, it sends the message “05”. When WSim processes MSG11 in June 2001, it sends the message “ENTER PASSWORD FOR 06/01”. If WSim processes MSG11 again in April 2002, it sends the message “ENTER PASSWORD FOR 04/02”.

As illustrated in the preceding example, the $MONTH$ and $YEAR$ options simplify the message generation process. Even though you code each message only once, WSim inserts variable data that creates a different message for each month and year.

Although options like $MONTH$ can stand alone as the only data coded in a statement's data field, other options require you to code additional data. These data field options indicate to the system under test that the accompanying data originates from a specific resource. For more information about using data field options to generate messages for specific simulated resources, see Chapter 18, “Generating messages for specific types of devices,” on page 219.

The following sections explain how to use some of the data field options available with WSim to perform various tasks:
- Generate random numbers
- Generate data from user tables
- Generate data using counters
- Retrieve data from user or save areas.

For a complete list of the data field options available with WSim, see the *WSim Script Guide and Reference*.

Random numbers

Using the $RNUM$ data field option, you can insert random numbers into messages dynamically instead of selecting and coding a different number for each TEXT statement. You can generate a random number with the $RNUM$ option in the following ways:
- Specify the number of digits for the random number and a range of numbers that the random number falls within by coding the $RNUM$ option on a TEXT statement.
- Specify the range of numbers on an RN network definition statement. Then reference that range with the $RNUM$ option on a TEXT statement.

The following sections describe both methods of coding the $RNUM$ data field option.

Specifying a range of numbers

With the $RNUM$ data field option, you can specify a range of numbers from which WSim generates a random number:

```
MSG1  TEXT  ($RNUM,lo,hi,n$)  Generates a random number n digits long
*                             between a low and high value.
```

In this example, *lo* and *hi* specify the low and high values in a range of numbers. You can code *lo* as an integer from 0 to 2147483646 and *hi* as an integer from 1 to 2147483647. *lo* and *hi* can also be counter specifications whose values are within this range. *n* specifies how many digits the random number can contain and is an integer from 1 to 10. *hi* must be greater than *lo*.

Note: If WSim generates a random number shorter than *n*, WSim pads the number with leading zeroes until it is the specified length. If the generated number is longer than *n*, WSim truncates the number on the left.

The following example illustrates the coding required to generate a 3-digit random number from 0 to 999:

```
MSG1  TEXT  ($RNUM,0,999,3$)  Generates 3-digit random number from 0 to 999.
```

When WSim processes MSG1, it generates a 3-digit number whose value can be from 0 to 999 and sends that number to the system under test as MSG1.

Using the RN network definition statement

With the RN random number statement, you can specify a range of numbers when you code the network definition:

```
NET1   NTWRK
*                              Beginning of NET1.
1      RN     LOW=10,HIGH=50     Specifies range for random numbers.
```

When you reference this range of numbers with the $RNUM$ data field option, WSim generates a random number within the specified range. In the preceding example, note that a number 1 appears in the RN statement's name field. By coding each RN statement with a unique number, you can reference each statement from the message generation deck.

To reference an RN network definition statement, code the $RNUM$ data field option with the following syntax:

```
MSG1  TEXT  ($RNUM,m,n$)
*                         Generates a random number from a range
*                         specified on an RN statement.
```

m specifies the number of an RN statement and is an integer from 0 to 4095. *n* specifies how many digits the random number contains and is an integer from 1 to 5. If WSim generates a random number shorter than *n*, WSim pads the number with leading zeroes until it is the specified length. If the generated number is longer than *n*, WSim truncates the number on the left.

As shown in the following example, code *m* as a 1 to reference RN statement 1 and code *n* as a 2 to generate a random number 2 digits long:

```
NET1   NTWRK
*                               Beginning of NET1.
        .
        .                       Network definition statements.
        .
1       RN     LOW=10,HIGH=50       Specifies range for random numbers.
        .
        .                           Network definition statements.
        .
DECK1  MSGTXT
*                                   Beginning of DECK1.
MSG1   TEXT   ($RNUM,1,2$)          Generates a 2-digit random number from
*                                   the range specified on RN statement 1.
        .
        .                           Message generation statements.
        .
```

When WSim processes the statements in the preceding example, it generates a 2-digit random number from the range 10 to 50 and sends that number as the message generated by MSG1.

For complete information about the syntax and coding conventions WSim requires for the $RNUM$ data field option, see the *WSim Script Guide and Reference*.

User tables

When you create a message generation deck, you may have several data fields in your messages, each of which may have several possible values. To use different combinations of fields and values, code a user table, which is a collection of data that contains possible values for a specific field. For example, you can code a user table that provides a complete list of employee names. When you code a reference to this user table on the TEXT statement data field, WSim selects an entry, that is, an employee name, and inserts it into messages sent to the system under test.

The following sections provide detailed information about defining a user table and selecting table entries with the $UTBL$ data field option.

Defining a user table

In WSim, you can define a user table with one of the following statements:

- The MSGUTBL message generation statement.
- The UTBL network definition statement

Each user table may contain up to 2147483647 entries; each entry may contain any number of characters. However, it may be truncated to 32767 characters during message generation.

To create a table that you can reference from several network definitions, code the ***MSGUTBL message generation statement***. To define a user table with the MSGUTBL statement, you must code a name in the MSGUTBL's name field. Then code table entries by enclosing data within parentheses and separating entries with a comma. To place hexadecimal data in a user table, remember to enclose the digits within single quotes: for example, ("1234").

The following example shows how to code the MSGUTBL statement:

```
PARTNUMS  MSGUTBL  (NUM1),(NUM2),  Defines entries for a user table named
                   (NUM3),(NUM4)   PARTNUMS.
```

When you code the MSGUTBL statement, WSim maintains the set of table entries as a separate member of a partitioned data set along with any message generation decks that refer to the table. The MSGUTBL statement is not a part of any one network definition or message generation deck; you do not code this statement between a deck's MSGTXT and ENDTXT statements. All MSGUTBL statements must follow the network definition statements or be previously stored in the partitioned data set named by the MSGDD DD statement. You do not have to code MSGUTBLs before a script's MSGTXTs; they can be interspersed with them. MSGUTBLs are separate members of the MSGDD data set.

To define a user table with the ***UTBL network definition statement***, you must enter an integer from 0 to 4095 in the statement's name field; WSim requires this integer to identify the user table. Then code table entries by enclosing data within parentheses and separating entries with a comma. To place hexadecimal data in a user table, you must also enclose the data within single quotes: for example, ("1234").

The following example shows how to code the UTBL statement:

```
1    UTBL  (SMITH),(JONES),      Defines entries for user table 1.
           (BAILEY),(ROSS)
```

You can code one table in many networks by coding the same table in several network definitions with several UTBL statements. However, to change an entry in the table, you must change that entry in each network definition.

Note: The entries that you define on the UTBL and MSGUTBL statements are static; you cannot change the entries during a simulation.

The example below illustrates the coding required to define a user table with both the UTBL and MSGUTBL statements. The example also shows the placement of the MSGUTBL statement.

```
NET1     NTWRK
*                       Sample WSim Script
*   This script illustrates the coding required to define user tables
*   with the UTBL and MSGUTBL statements.
*
*                                    Beginning of NET1.
*
          .
          .                          Network definition statements.
          .
1        UTBL      (SMITH),(JONES),  Defines user table 1.
                   (BAILEY),(ROSS)
          .
          .                          Network definition statements.
          .
PARTNUMS MSGUTBL   (NUM1),(NUM2),    Defines user table PARTNUMS.
                   (NUM3),(NUM4)
DECK1    MSGTXT
*                                    Beginning of DECK1.
          .
          .                          Message generation statements.
          .
        ENDTXT                       End of DECK1.
```

In this example, the UTBL statement defines user table 1 within the network definition of NET1; the MSGUTBL statement defines a user table named PARTNUMS. Notice that you code the MSGUTBL statement after the network definition statements.

When you define a user table on the MSGUTBL statement, you can reference the table on a UTBL statement in the network definition. As shown in the following example, code the table's name on the UTBL statement in the operand field.

```
NET1     NTWRK
*                                  Beginning of NET1.
1        UTBL      PARTNUMS        References user table PARTNUMS.
          .
          .                        Network definition statements.
          .
PARTNUMS MSGUTBL   (NUM1),(NUM2),    Defines user table PARTNUMS.
                   (NUM3),(NUM4)
```

As discussed in the following section, you can select table entries from a table defined on a UTBL statement or a MSGUTBL statement. You can also select table entries from a table that is defined on a MSGUTBL statement and then referenced by name on a UTBL statement.

Generating messages with the $UTBL$ data field option

After you have defined a user table and coded the table's entries, you can generate messages by selecting table entries with the $UTBL$ data field option.

```
MSG1  TEXT  ($UTBL,id,sel$)   Coding to insert a user table entry into
*                             a message sent to the system under test.
```

When you code the $UTBL$ option, *id* specifies the name of a MSGUTBL statement or the number of a UTBL statement. *sel* specifies how WSim selects the entry from the user table. As discussed in the *WSim Script Guide and Reference*, WSim provides several values for *sel* that enable you to determine how WSim selects a table entry:

F*n* Specifies fixed selection, that is, WSim selects the same entry each time it processes the $UTBL$ option. *n* is an integer that specifies the index of the entry WSim selects from the user table. If *n* exceeds the number of entries in the table, WSim does not select an entry or include any data in the message being generated.

R Specifies random selection of an entry. WSim generates a random number and then selects the entry with the corresponding index.

R*n* Specifies random selection based on a probability distribution defined by UDIST network definition statement *n*. *n* is an integer from 0 to 4095.

cntr Specifies selection of an entry based on the value of an index counter. For more information about selecting user table entries with index counters, see “Index counters” on page 127.

Note: An index is a number used to identify entries in a user table.

To specify fixed selection of table entries, code the $UTBL$ option as shown in the following example:

```
MSG1  TEXT  ($UTBL,id,Fn$)   Coding for fixed selection of table entries.
```

WSim uses zero indexing for user tables. That is, it indexes entries sequentially beginning with 0. For example, you can specify fixed selection of the fourth entry as shown in the following example:

```
PARTNUMS MSGUTBL   (NUM1),(NUM2),(NUM3),   Defines entries for a user table
                   (NUM4),(NUM5),(NUM6)    named PARTNUMS.
          .
          .                                Message generation statements.
          .
MSG1     TEXT      ($UTBL,PARTNUMS,F3$)    WSim selects the entry indexed as
*                                          number 3 in table PARTNUMS, which
*                                          is NUM4, the fourth table entry.
```

In MSG1, F3 specifies that WSim selects the entry from PARTNUMS with a corresponding index of three, which is the fourth entry, NUM4. Because selection is fixed, WSim selects NUM4 each time it processes MSG1.

To specify random selection of table entries, code the $UTBL$ option as shown in the following example.

```
MSG8  TEXT  ($UTBL,id,R$)  Coding for random selection of table entries.
```

In this example, R indicates random selection. When WSim processes MSG8, it generates a random integer and then selects the table entry with an index number that corresponds to the random integer. In this way, WSim can select a different entry each time it processes the message generation statement.

You can also select table entries randomly based on a probability distribution defined on the UDIST network definition statement.

```
1     UDIST  80,20      Assigns weights for selecting entries.
```

As shown in the preceding example, WSim requires an integer in the UDIST statement's name field. The numbers you code in the UDIST statement's operand field assign weights to corresponding table entries. Each weight represents a fraction that determines the probability that WSim selects a particular entry during message generation. For example, 80 indicates that if WSim processes the statement 100 times on the average, it selects the first entry in a table 80 times or 80% of the time. The 20 indicates that WSim selects the second table entry 20 times or 20% of the time. If there are more than two entries in the user table, WSim selects only the first two entries.

You can code up to a maximum of 2000 weights on a UDIST statement. However, the number of weights must be less than or equal to the number of entries coded on the corresponding UTBL or MSGUTBL statement. For example, you cannot assign 20 different weights to a table with only three entries.

To specify random selection based on a probability distribution, code the $UTBL$ option as shown in the following example.

```
MSG3  TEXT  ($UTBL,id,Rn$)  Coding to select an entry based on a
*                           probability distribution.
```

In this example, *id* specifies the number of a UTBL statement or the name of a MSGUTBL statement that defines the user table which WSim selects an entry; R specifies random selection. *n* is the number of the UDIST statement that defines the probability distribution.

The sample script shown below illustrates a network definition and message generation deck with the coding required to select an entry based on a probability distribution.

```
NET1   NTWRK
*                           Sample WSim Script
*   This script illustrates the coding required to define a table and its
*   entries and then select an entry based on a probability distribution.
*
*                               Beginning of NET1.
*
          .
          .                     Network definition statements.
          .
2      UDIST   50,30,20         Defines a probability distribution.
          .
          .                     Network definition statements.
          .
TBL1   MSGUTBL (1487),(7911),   Entries for user table TBL1.
               (1098)
          .
          .                     Other MSGUTBLs and MSGTXTs.
          .
DECK3  MSGTXT
*                               Beginning of DECK3.
MSG5   TEXT    ($UTBL,TBL1,R2$) Defines random selection of entries from
*                               TBL1 with distribution defined by UDIST 2.
         .
         .                      Message generation statements.
         .
       ENDTXT                   End of DECK3.
```

Every time WSim processes the TEXT statement named MSG5, it selects a table entry from the table defined by MSGUTBL statement TBL1 based on the distribution defined by UDIST statement 2. WSim selects the first table entry 50% of the time, the second table entry 30% of the time, and the third table entry 20% of the time. Since the selection is specified as random, entries will be selected randomly in this proportion.

For complete syntax and coding requirements for the UDIST statement, see the *WSim Script Guide and Reference*.

Sequence and index counters

A counter is a storage location used to hold a numeric value. To help you generate a message dynamically, WSim provides two kinds of counters: sequence counters and index counters. The two types vary slightly based on how you assign values to the counter, how you use the counter to generate messages, and how WSim increments the counter's value.

WSim automatically provides one sequence counter for each network, line, terminal, and device. The names of these counters are the following:

NSEQ	Network sequence counter.
LSEQ	Line sequence counter.
TSEQ	Terminal sequence counter.
DSEQ	Device sequence counter.

Note: These counters are also allocated to VTAMAPPLs, APPCLUs, TCP/IP connections, TPs, and LUs. For more information about how counters are assigned, see Table 10 on page 203.

When you reference a sequence counter on the TEXT statement, WSim inserts the value of the sequence counter directly into the message data stream. For information about how WSim determines the value of a sequence counter, see “Sequence counters” on page 126.

You can specify from 3 to 4095 index counters for each network, line, terminal, and device. The names of these counters are the following:

NC*n*	Network index counter *n*.
LC*n*	Line index counter *n*.
TC*n*	Terminal index counter *n*.

DC*n*	Device index counter *n*.

As illustrated by Table 10 on page 203, these counters are also allocated to VTAMAPPLs, APPCLUs, TCP/IP connections, TPs, and LUs. If you specify any user exits, WSim allocates either the number of index counters specified on the CNTRS operand or the number of counters referenced in the network, whichever is greater. If you do not specify any user exits, WSim allocates as many index counters as are needed and ignores the CNTRS operand. WSim allocates a minimum of three index counters.

Although WSim can insert the value represented by an index counter into the message you are generating, you normally use an index counter's value to select an entry from a user table. For more information about how WSim determines the value of an index counter, see “Index counters” on page 127.

The following sections describe how to generate messages dynamically with sequence and index counters. For information about setting the value of a counter and using counters to position the cursor, see the following sections:

- “Coding the SCANCNTR operand” on page 179 describes the SCANCNTR operand on the IF message generation statement. With this operand, you can set a counter to the offset of the text that caused the test condition to be met.
- “Setting switches and counters” on page 202 provides information about setting the value of a counter with the SET statement.
- “Simulating cursor movement” on page 231 describes how to code the CURSOR and SELECT statements to position the cursor using the value of one or two counters.

Sequence counters

You can reference the sequence counters provided by WSim by coding the following data field options on the TEXT statement:

$NSEQ$	References a network sequence counter.
$LSEQ$	References a line sequence counter.
$TSEQ$	References a terminal sequence counter.
$DSEQ$ or SEQ	References a device sequence counter.

When you code a reference to a sequence counter with one of the above data field options, WSim automatically increments the value of the sequence counter before inserting the counter's value into the data stream. However, in no other situation do the sequence counters get automatically incremented. For example, when you code the $CNTR$ and $CNTRX$ data field options, you can reference the decimal and hexadecimal values, respectively, of any counter without changing the counter's value. When WSim increments a sequence counter's value, the value automatically wraps to 0 after reaching 2147483647.

In addition to the data field option that names a counter, you must also specify the number of digits for the value WSim includes in the data stream:

```
MSG1  TEXT  ($NSEQ,3$)       Increments the value of the network sequence
*                            counter and inserts a 3-digit value into MSG1.
MSG2  TEXT  ($DSEQ,4$)       Increments the value of the device sequence
*                            counter and inserts a 4-digit value into MSG2.
MSG3  TEXT  ($CNTR,NSEQ,5$)  Inserts the value of the network sequence
*                            counter without changing the counter's value,
*                            which is 5 digits long.
```

```
MSG4  TEXT  ($CNTRX,TSEQ,2$) Inserts the hexadecimal value of the terminal
*                            sequence counter without changing the counter's
*                            value, which is 2 bytes long.
```

Each time WSim processes MSG1, it increments the network sequence counter's value by one and inserts the incremented value into the message as a 3-digit number. When processing MSG2, WSim increments the device sequence counter's value by one and inserts the incremented value into the message as a 4-digit number. For MSG3, WSim inserts the network sequence counter's value into the message as a 5-digit number without changing the counter's value.

Note: If the counter's value has fewer digits than the length specified by the data field option, WSim pads the value with leading zeroes on the left. If the counter's value has more digits than the length specified by the data field option, WSim truncates the number on the left.

Index counters

When you generate messages dynamically with data from a user table, you can select a table entry with index counters. By coding the $UTBL$ data field option on the TEXT statement, WSim selects an entry based on the value of the specified index counter:

```
MSG2  TEXT  ($UTBL,id,sel$)  Selects a table entry using the value of an
*                            index counter.
```

As discussed in "User tables" on page 121, *id* identifies a user table with number of a UTBL statement or the name of a MSGUTBL statement. *sel* specifies the entry that WSim selects from that table.

To select a table entry with an index counter, code *sel* with an option that references a specific index counter. Remember that WSim enables you to code from 3 to 4095 index counters for each network, line, terminal, or device. After you reference an index counter, WSim can increment the value represented by an index counter, depending on how you reference the counter. The value of each index counter automatically wraps to 0 after you reference the last entry in a user table.

As shown in the following list, each option for *sel* defines the type of index counter and the number of the appropriate index counter:

CN*n*, SN*n*, and NC*n*	Reference network index counter *n*.
CL*n*, SL*n*, and LC*n*	Reference line index counter *n*.
CT*n*, ST*n*, and TC*n*	Reference terminal index counter *n*.
CD*n*, SD*n*, and DC*n*	Reference device index counter *n*.

Note: When you code the $UTBL$ data field option, you can identify index counters differently than you do in other statements; compare the coding above with the names of index counters introduced in "Sequence and index counters" on page 125.

Each of these options causes WSim to handle the counters differently. To increment the counter sequentially every time WSim references its value, code an S before the letter identifying the index counter. For example, reference the value of terminal index counter 35 and increment the value sequentially with the following coding:

```
MSG8  TEXT  ($UTBL,TABLE1,ST35$)
*                         References the value of terminal index counter
*                         35, selects an entry from the user table, and
*                         increments the value of the counter.
```

To hold the counter's value constant, code the option with a C:

```
MSG8  TEXT  ($UTBL,TABLE4,CL52$)
*                         References the value of line index counter 52,
*                         holds the value constant, and selects an entry
*                         from user table TABLE4.
```

NC*n*, LC*n*, TC*n*, and DC*n* also hold the counter's value constant. When you code these options, however, WSim does not reset the counter to 0 when it is set to a value corresponding to an entry beyond the last entry in a user table.

The sample script shown below illustrates how to generate a message that lists an order for parts. The order consists of values for several fields: an order number, the quantity of parts required, the part number, and a description of the part. Two user tables provide the data for the part number and description; the $RNUM$ option generates a random number for the quantity. The $NSEQ$ option references the network sequence counter to supply the part number.

```
NET1   NTWRK
*                        Sample WSim Script
*   This script illustrates the coding required to generate a message
*   that references user table entries, generates a random number, and
*   references the value of a sequence counter.
*
*                                         Beginning of NET1.
*
        .
        .                                 Network definition statements.
        .
PNUMS  MSGUTBL  (04128),(34591),          Defines part numbers.
                (95735),(39782),(89678)
PNAMES MSGUTBL  (GIZMO),(SCREW),(NAIL),   Defines hardware.
                (WIDGET),(STAPLE)
DECK3  MSGTXT
*                                         Beginning of DECK3.
MSG1   TEXT     (ORDER $NSEQ,5$ QUANTITY ), Generates message.
                ($RNUM,1,10,2$ PART NO. ),
                ($UTBL,PNUMS,CN1$ ),
                ($UTBL,PNAMES,SN1$)
       ENDTXT                             End of DECK3.
```

With the coding in the preceding example, WSim selects an order number, a part number, a quantity for that part number, and a description of the part. The coding for the TEXT statement provides each part of the message:

$NSEQ,5$	Specifies that the value of the network sequence counter be used as an order number 5 digits long.
$RNUM,1,10,2$	Specifies that WSim inserts a random quantity into the message that falls within the range from 1 to 10 and is 2 digits long.
$UTBL,PNUMS,CN1$	Specifies that WSim selects the part number from user table PNUMS with an index that corresponds to the value in network index counter 1. The value of network index counter 1 remains constant.

$UTBL,PNAMES,SN1$	Specifies that WSim selects the part description from user table PNAMES with an index that corresponds to the value of network index counter 1. WSim increments this counter's value after referencing its value.

By incrementing the counter's value each time WSim selects an entry from PNAMES, you can select pairs of entries from the two tables. For example, if the value of network index counter 1 is 3, WSim selects the entry with index number 3 from both tables the first time it processes MSG1. Then WSim increments the counter's value by 1, which changes its value to 4. The second time WSim processes MSG1, it selects the entry with index number 4 from both tables and then increments the counter's value again. Each time WSim processes MSG1 it selects another pair of entries.

The following example shows one of the possible messages for a sample order generated from the preceding network and message generation statements:

```
ORDER 00004 QUANTITY 06 PART NO. 95735 NAIL
```

Although you code MSG1 in the preceding sample deck only one time, WSim can send a different message every time it processes the deck.

You can select table entries sequentially or combine sequential selection with the fixed and random selection of entries discussed in "Generating messages with the $UTBL$ data field option" on page 123. The script shown in the example below illustrates the coding required to combine these three methods of selection in one TEXT statement.

```
NET7  NTWRK
*                        Sample WSim Script
*   This script illustrates the coding required to combine fixed, random,
*   and sequential selection of user table entries.
*
*                                       Beginning of NET7.
*
       .
       .                                Network definition statements.
       .
ACT   MSGUTBL    (ORDER),(UPDATE),      Entries for action table.
                 (PRICE)
NUM   MSGUTBL    (6615),(1976),         Entries for part number table.
                 (5699)
CLR   MSGUTBL    (RED),(YELLOW),        Entries for color table.
                 (GREEN)
DECK1 MSGTXT
*                                       Beginning of DECK1.
MSG1  TEXT       ($UTBL,ACT,F1$ ),      Inserts entries from ACT, NUM, and
                 ($UTBL,NUM,SD2$),      CLR into MSG1; the value of device
                 ( COLOR $UTBL,CLR,R$)  index counter 2 is 0.
       .
       .                                Message generation statements.
       .
      ENDTXT                            End of DECK1
```

When WSim processes the TEXT statement in the preceding example, the following message could result:

```
UPDATE 6615 COLOR GREEN
```

User and save areas

When you code the TEXT statement, you can generate messages for simulated resources by retrieving data from one of the following user or save areas:

- Network user area
- Device user area
- Network save area
- Device save area.

During message generation, you can store and retrieve data in a ***network user area***, which is a work area for devices that generate messages. You can use a network user area to store data used by every device on the network or to pass data from one device to another. As shown in the following example, you can allocate one network user area for each network with up to 32767 bytes of storage by coding the NETUSER operand on the NTWRK statement.

```
NET4  NTWRK  NETUSER=32767   Allocates a 32767-byte network user area.
```

The ***device user area*** defines a storage area that serves as a work area for a specific device. As shown in the following example, you can allocate one device user area for each device with up to 32767 bytes of storage with the USERAREA operand on the DEV network definition statement.

```
DEV1  DEV USERAREA=200   Allocates a 200-byte device user area for DEV1.
```

WSim does not allocate user areas dynamically; you must request each area with the appropriate statement and operand. If you reference a user area that you did not allocate in the network definition, you may have difficulty finding your error. In addition, WSim does not detect the reference as a syntax error.

A ***network or device save area*** stores data used by a specific network or device. You can allocate up to 4095 save areas per network or device dynamically by referencing the area from your message generation deck. In addition, the length of a save area is dependent on the length of data you save rather than a number of bytes you specify. For example, each time you code a message generation statement that places data in a save area, WSim automatically provides you with an area that is equal in length to the data you are saving.

Although network save areas are always allocated dynamically during a simulation, you can statically allocate a device save area by coding the SAVEAREA operand on the DEV network definition statement. As shown in the following example, the SAVEAREA operand enables you to specify the number of save areas and the bytes of storage:

```
DEV1  DEV  SAVEAREA=(15,100)   Allocates 15 device save areas for DEV1,
*                              each 100 bytes long.
```

Note: When you code the SAVEAREA operand, each save area that you allocate is equal in length.

The following sections describe how to code the DATASAVE statement to place data in a save or user area and how to insert saved data into a message.

Placing data in a user or save area with the DATASAVE statement

With the DATASAVE message generation statement, you can place data from the device buffer or data coded manually into a save or user area. Use the following operands on the DATASAVE statement to code the data you want to save and the area WSim uses to save the data:

- AREA
- TEXT
- LOC

- LENG.

The ***AREA operand*** specifies the area that WSim uses to save the data. As shown in the following list, WSim provides several options for the AREA operand. For options with a positive or negative offset (±*value*), code *value* as an integer 0 - 32766 or the name of a counter 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*).

N±*value*	Specifies that WSim saves the data in the network user area, 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.
N*s*	Specifies the network save area WSim uses to save the data, where *s* is an integer from 1 to 4095.
s	Specifies the device save area WSim uses to save the data, where *s* is an integer from 1 to 4095.
U±*value*	Specifies that WSim saves the data in the device user area, 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 outside the user area, WSim does not save any data and writes an informational message to the log data set.

The ***TEXT operand*** specifies the data you want to save. Although you can code any amount of data, WSim truncates data on the right that is longer than the space available in the save or user area and writes an informational message to the log data set.

The following example shows the syntax required for the TEXT and AREA operands:

```
SAVE1  DATASAVE  TEXT=(SAVE A ),  Specifies the data to be saved.
                 (MESSAGE),
                 AREA=N+5    Places the data in the network user
*                            at a position offset 5 bytes from the
*                            beginning of the area.
```

If you do not code the TEXT operand, you must code the LOC and LENG operands. The ***LOC operand*** specifies the location of the data to be saved based on an area in the screen buffer, on the location of the cursor, or on an area of the data stream. The following list shows the values WSim provides for the LOC operand. *value* is an integer 0 - 32766 or the name of a counter 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*).

B±*value*	+*value* specifies that the data to be saved is located at an offset from the beginning of the device buffer, excluding any headers. For display devices, -*value* specifies that the data to be saved is located at an offset back from the end of the screen image buffer. For nondisplay devices, -*value* specifies that the data to be saved is located at an offset back from the end of the data in the device buffer.
C±*value*	+*value* specifies that the data to be saved is located at an offset from the current cursor position. -*value* specifies that the data to be saved is located at an offset back from the current cursor position. Code these values only for display devices.
D+*value*	+*value* specifies that the data to be saved is located at an offset from the beginning of the data stream.

TH+*value*	+*value* specifies that the data to be saved is located at an offset from the beginning of the transmission header.
RH+*value*	+*value* specifies that the data to be saved is located at an offset from the beginning of the request/response header.
RU+*value*	+*value* specifies that the data to be saved is located at an offset from the beginning of the request/response unit.
(*row,col*)	Specifies the row and column location of the screen image data to be saved. *row* and *col* are integers 1 - 4095 or names of valid counters.
*	Specifies that the data is to be saved from the device buffer location coded on the last logic test that WSim executed and then took the specified THEN action. This option is valid only when WSim encounters the DATASAVE statement as a result of a logic test execute function (THEN=E*name-label*).

The ***LENG operand*** specifies how many bytes of data WSim places into the save or user area. Code LENG=*value* where *value* is an integer from 1 to 32767; however, do not specify an integer for *value* that exceeds the space available in the user or save area. If the specified length exceeds the available user or save area space, WSim uses the available space and then writes an informational message to the log data set. If the length specified is greater than the length of the data in the device buffer, WSim only saves the data available.

The following example shows the syntax required for the LOC and LENG operands.

```
SAVE1  DATASAVE  AREA=3,   Places data in device save area 3.
                 LOC=C-5,  Specifies data offset back from the cursor.
                 LENG=30   Specifies that 30 bytes of data be saved.
```

The example below illustrates the coding required to save a specific message and to save data from a screen buffer.

```
NET4   NTWRK     NETUSER=4000       Allocates network user area.
*
*                        Sample WSim Script
*   This script illustrates the network definition and message generation
*   statements required to save a specific message from a screen buffer.
*
DEV1   DEV       USERAREA=200       Allocates a 200-byte device user area
*                                   for DEV1.
       .
       .                            Network definition statements.
       .
DECK8  MSGTXT
*                                   Beginning of DECK8.
       .
       .                            Message generation statements.
       .
SAVE1  DATASAVE  AREA=3,
                 TEXT=(SAVE THIS ), Specifies message to be saved in
                      (MESSAGE)     device save area 3.
SAVE2  DATASAVE  AREA=U+10,         Specifies that 30 bytes of data,
                 LOC=C+0,LENG=30    beginning at the location of the
*                                   cursor, be saved from the screen
*                                   buffer into the device user area
*                                   in a position offset 10 bytes
*                                   from the beginning of the area.
SAVE3  DATASAVE  AREA=1,            Specifies that the text HOWDY be saved
                 TEXT=(HOWDY)       at offset 0 in device save area 1.
SAVE4  DATASAVE  AREA=N2,           Specifies that 8 bytes of data from
                 LOC=B+NC2,LENG=8   an offset equal to the value of network
*                                   index counter 2 in the device buffer
*                                   be saved in network save area 2.
SAVE5  DATASAVE  AREA=N+DSEQ,       Specifies that the text HELLO be saved
                 TEXT=(HELLO)       in the network user area at an offset
*                                   equal to the value of the device
*                                   sequence counter.
        .
        .                           Message generation statements.
        .
       ENDTXT                       End of DECK8.
```

For additional information about coding the NETUSER, USERAREA, and SAVEAREA network definition statements, see the *WSim Script Guide and Reference*. For more information about the DATASAVE message generation statement, see "DATASAVE" on page 196.

Manipulating Data in a Save or User Area with the DATASAVE Statement

You can manipulate the data coded in the TEXT operand of the DATASAVE statement with the B2X, BITAND, BITOR, BITXOR, CENTER, DELETE, DELWORD, INSERT, LEFT, OVERLAY, REVERSE, RIGHT, SPACE, STRIP, STRIPL, STRIPT, SUBWORD, TRANSLATE, X2B, and X2C functions.

To specify the function you want to perform, code the FUNCTION operand on the DATASAVE statement. You may also specify the INSERT, POS, PLENG, PAD, TABLEI, and TABLEO operands, depending on the function you want to perform.

The following example shows how to use the INSERT function.

```
DATASAVE  AREA=1,
          TEXT=(HAVE DAY)       Stores "HAVE DAY" in save area 1.

DATASAVE  AREA=2,
          TEXT=($RECALL,1$),    Specifies the target data into which
*                               it will be inserted.
          FUNCTION=INSERT,
          INSERT=(A NICE),      Specifies the data to be inserted.
          POS=5,                Specifies the position in the TEXT
*                               data after which the INSERT data is
*                               to be inserted.
          PLENG=7               Specifies the length to which the
*                               INSERT data is to be padded.
WTO       ($RECALL,2$)          Writes "HAVE A NICE DAY" to the console.
```

Note: The default pad character is a blank.

Converting data in a save or user area with the DATASAVE statement

You can manipulate and convert DBCS or SBCS data coded in the TEXT operand of the DATASAVE statement with the DBCSADD, DBCSADJ, DBCSDEL, DBCS2SB, SB2DBCS, and SB2MDBCS functions.

To specify the function you want to perform, code the FUNCTION operand on the DATASAVE statement.

In the following examples and discussion, the SO character is represented using a "<", the SI character is represented using a ">", and the first byte of each DBCS character, which is referred to as the ward byte, is represented using a "." character.

The following example shows how to use the function.

```
DATASAVE FUNCTION=DBCSADD,          Add SO/SI to TEXT=(..) data
        TEXT=('42C142C242C3'),      ".A.B.C"
        AREA=1                      "<.A.B.C>"

DATASAVE FUNCTION=DBCSADJ,          Delete SI/SO pairs from
                                    TEXT=(..) data
        TEXT=('0E42C10F0E42C20F'),  "<.A><.B>"
        AREA=1                      "<.A.B>"

DATASAVE FUNCTION=DBCSDEL,          Delete SO/SI from TEXT=(..) data
        TEXT=(0E42C142C20F'),       "<.A.B>"
        AREA=1                      ".A.B."

DATASAVE FUNCTION=DBCS2SB,          Convert DBCS ward 42 EBCDIC TEXT=(..)
                                    to SBCS
        TEXT=('42C142C242C3'),      ".A.B.C"
        AREA=1                      "ABC"

DATASAVE FUNCTION=DBCS2SB,          Convert DBCS ward 42 EBCDIC TEXT=(..)
                                    to SBCS
        TEXT=('0E42C442C50F'),      "<.D.E>"
        AREA=1                      "DE"

DATASAVE FUNCTION=SB2DBCS,          Convert TEXT=(..) to DBCS
        TEXT=(ABC),                 "ABC"
        AREA=1                      ".A.B.C"

DATASAVE FUNCTION=SB2MDBCS,         Convert TEXT=(..) to DBCS Mixed
        TEXT=(ABC),                 "ABC"
        AREA=1                      "<.A.B.C>"
```

See “Simulating DBCS data entry for simulated 3270 DBCS terminals” on page 241 for more information about simulating 3270 DBCS terminals.

Inserting data into a message

After you store data in a save or user area, you can retrieve the data during message generation. Then, you can insert the retrieved data into messages generated for the system under test, helping to automate the message generation process.

When you specify messages on the TEXT message generation statement, you can retrieve stored data with the $RECALL$ data field option. The following list shows the values that you can code on this data field option:

$RECALL,***integer***$
: Retrieves all the data last saved in a device save area. The amount retrieved is the same as the amount last saved. *integer* is an integer 1 - 4095 that specifies the number of the save area.

$RECALL,*area*,*length***$**
: Retrieves data from a network or device save or user area by specifying the area where the data is stored and the length of data to be inserted into the data stream. *area* is a location in a save or user area. *length* is an integer 1 - 32767 or the name of a valid counter. *length* is optional; it specifies the amount of data to recall. If you do not code this value, WSim returns all of the data following the offset.

When you code one of the following options for *area*, *value* for the N, U, N*s*, and *s* options is an integer 0 - 32766 or the name of a counter whose value is less than the length of the save or user area. For the B, C, D, TH, RH, and RU options, *value* is an integer 0 - 32766 or a counter specification whose value is within this range.

N±*value*	Specifies an offset into the network user area from the beginning of the area (+*value*) or the end of the area (-*value*).
N*s***+***value*	Specifies offset *value* in a network save area. *s* is an integer 1 - 4095 that specifies the network save area.
*s***+***value*	Specifies offset *value* in a device save area. *s* is an integer 1 - 4095.
U±*value*	Specifies an offset into the device user area from the beginning of the area (+*value*) or from the end of the area (-*value*).
B±*value*	Specifies an offset into the device buffer from the beginning of the buffer (+*value*) or from the end of the buffer (-*value*). For display devices, +*value* specifies an offset from the beginning of the screen image display buffer and -*value* specifies an offset from the end of the screen image display buffer.
C±*value*	Specifies an offset beyond the current cursor position for a display device (+*value*) or before the current cursor position for a display device (-*value*).
D+*value*	Specifies an offset from the beginning of the data stream.
TH+*value*	Specifies an offset from the beginning of the transmission header.
RH+*value*	Specifies an offset from the beginning of the request/response header.
RU+*value*	Specifies an offset from the beginning of the request/response unit.
(*row,col***)**	Specifies the row and column of the screen image of the display device.

The following example illustrates using the $RECALL$ data field option to retrieve data from save and user areas:

```
DECK8  MSGTXT
*                                Beginning of DECK8.
MSG1   TEXT   ($RECALL,N+0,7$)   Retrieves data 7 bytes long from the
*                                beginning of the network user area (0
*                                offset).
MSG2   TEXT   ($RECALL,2,35$)    Retrieves data 35 bytes long from device
*                                save area 2.
MSG3   TEXT   ($RECALL,N3$)      Retrieves data from network save area 3.
MSG4   TEXT   ($RECALL,2$)       Retrieves data from device save area 2.
MSG4   TEXT   ($RECALL,U+DC3,4$) Retrieves 4 bytes of data from the device
*                                user area starting at an offset equal to
*                                the value of device counter 3.
MSG5   TEXT   ($RECALL,U+DC3$)   Retrieves all data previously saved from
*                                the device user area starting at an offset
*                                equal to the value of device counter 3.
MSG6   TEXT   ($RECALL,B+2$)     Retrieves all data from the device buffer,
*                                starting at an offset of 2.
MSG7   TEXT   ($RECALL,C-0,3$)   Retrieves three bytes of data from the
*                                screen image display buffer, starting at
*                                the current cursor position.
MSG8   TEXT   ($RECALL,D+4$)     Retrieves all data from the data stream
*                                starting at an offset of 4.
MSG9   TEXT   ($RECALL,(5,20),8$)
*                                Retrieves eight bytes of data from the
*                                screen image display buffer, starting at
*                                row 5, column 20.
MSG10  TEXT   ($RECALL,TH+DC3$)  Retrieves all data from the transmission
*                                header, starting at an offset specified in
*                                device counter 3 to the end of the message.
MSG11  TEXT   ($RECALL,RH+0$)    Retrieves all data in the request/response
*                                header until the end of the message.
MSG12  TEXT   ($RECALL,RU+4,2$)  Retrieves two bytes of data from the request
*                                unit, starting at an offset of 4.
       .
```

```
.                          Message generation statements.
.
ENDTXT                     End of DECK8
```

For the complete coding requirements for the RECALL data field option, see the *WSim Script Guide and Reference*.

Summary of message generation with the TEXT statement

In WSim, you define messages for the system under test by coding the TEXT message generation statement. With the TEXT statement, you can code messages by entering data manually in the TEXT statement's data field or you can generate messages dynamically with data field options. This chapter described entering data manually and generating data dynamically in the following ways:

Random numbers
: Code the $RNUM$ data field option to generate a random number and insert the number into a message.

User tables
: Code the $UTBL$ data field option to select an entry from a user table and insert the entry into a message.

Sequence counters
: Code data field options such as $NSEQ$, $LSEQ$, $TSEQ$, and $DSEQ$ to place a sequence counter's value into a message.

Index counters
: Code several options on the $UTBL$ data field option to select entries from a user table and place the entry into a message.

User and save areas
: Code the $RECALL$ data field option to retrieve data previously saved and insert the data into a message.

For information about generating data dynamically with other data field options that are available in WSim, see the *WSim Script Guide and Reference*. When you create a message generation deck, use these data field options to simplify the message generation process.

Chapter 14. Understanding delimiters

In Chapter 13, “Generating messages with the TEXT statement,” on page 117, you learned to generate messages with the TEXT message generation statement. To send messages to the system under test, however, you must code delimiters in your message generation deck. As discussed in Chapter 12, “Basic concepts,” on page 107, delimiters are message generation statements that interrupt the message generation process and send messages from the buffer to the system under test. To use delimiters to perform these functions, you must understand how delimiters affect the message generation process and how to code delimiters in a message generation deck.

This chapter provides the following information about delimiters:
- How delimiters affect message generation
- Conditions required for message generation
- How delimiters are classified as unconditional delimiters, conditional delimiters, and delimiters for specific types of devices
- Interrupting message generation with unconditional delimiters
- Sending messages with conditional delimiters
- Coding scripts with delimiters.

In addition to interrupting message generation with delimiters,WSim enables you to delay message generation. You can use statements and operands to create an intermessage delay, a period of time WSim delays between exiting message generation after processing a delimiter and reentering message generation. For more information about intermessage delays, see Chapter 15, “Understanding intermessage delays,” on page 153.

How delimiters affect the message generation process

There are a number of message generation statements that act as delimiters. When WSim processes a statement that serves as a delimiter, it takes the following actions:
1. Interrupts message generation for the active device.
2. Sends any messages in the device buffer to the system under test.
3. Waits until a message is received, a specified amount of time elapses, or all conditions for message generation have been satisfied for that device.
4. Continues to process the deck.

After WSim processes a delimiter, the following conditions must exist before the device can reenter message generation:
- The WAIT, EVENT WAIT, and QUIESCE indicators must be turned off.

  Note: The device can still enter message generation if end of chain has not been sent even if the QUIESCE indicator is turned on.
- The device cannot be in the console recovery state. For more information about console recovery and the F (Console Recovery) operator command, see *WSim User's Guide*.
- The INPUT INHIBITED indicator for a display device must be turned off.

Note: WSim also maintains an INPUT INHIBITED indicator for 3270, 3643, 5250, and LU2 display terminals. For more information about this indicator, see "INPUT INHIBITED indicator" on page 220.

- An SNA logical unit must be in the correct SNA state for sending data.
- The intermessage delay must have expired. Intermessage delays simulate the time an operator takes to enter data or to read data received from the system under test, as discussed in Chapter 15, "Understanding intermessage delays," on page 153.

As mentioned in the preceding list, a device cannot reenter message generation when a WAIT, EVENT WAIT, or QUIESCE indicator is on. An indicator is a flag that registers whether WSim has processed a WAIT or QUIESCE statement. By default, each indicator is off unless explicitly turned on by the associated message generation statement.

You can use the indicators to prevent a device from entering message generation. You may want to do this for the following reasons:

- You want to interrupt message generation until a certain event occurs.
- The system under test is not ready to receive additional messages.
- You want to simulate delays caused by slow operator response times so that you can maintain accurate synchronization between terminal traffic and system responses.

Note: If you code the SCAN operand on the NTWRK statement, WSim attempts to recover inactive terminals automatically by turning the WAIT and EVENT WAIT indicators off. With the SCAN operand, you specify the amount of time a resource can remain inactive before WSim attempts recovery. To maintain the delays coded in your message generation decks, do not code the SCAN operand for a shorter period of time than your longest delay. If you do code the SCAN operand for a shorter period of time,WSim attempts terminal recovery before the required delay is complete. For more information about the coding required for automatic terminal recovery, see Part 1, "Defining WSim networks," on page 1.

For more information about how the WAIT, EVENT WAIT, and QUIESCE indicators control message generation, see "Interrupting message generation with unconditional delimiters" on page 139.

How delimiters are classified

Message generation statements that act as delimiters can be classified by the way that they affect the message generation process:

- Unconditional delimiters
- Conditional delimiters
- Delimiters for specific types of devices
- CPI-C simulation statement delimiters.

An ***unconditional delimiter*** interrupts the message generation process and sends a message in the buffer to the system under test. Unconditional delimiters interrupt message generation regardless of whether a message exists in the buffer. See "Interrupting message generation with unconditional delimiters" on page 139 for more information about these delimiters.

A ***conditional delimiter*** interrupts the message generation process and sends the message in the buffer to the system under test. Conditional delimiters interrupt message generation only if a message has already been generated for the device or an attention identifier (AID) byte has been set. See "Sending messages with conditional delimiters" on page 148 for more information about these delimiters.

Delimiters for specific types of devices interrupt the message generation process and send messages in the buffer only when you simulate specific types of resources. When the AID byte is set, these become delimiters. For detailed information about these delimiters, see Chapter 18, "Generating messages for specific types of devices," on page 219.

CPI-C simulation statement delimiters cause WSim to relinquish control to VTAM.

The following summarizes the classification of message generation statements that are delimiters.

- Unconditional Delimiters

  STOP
  WAIT
  QUIESCE
- Conditional Delimiters

  CMND
  ENDTXT
  TEXT
- Delimiters for Specific Types of Devices

BTAB	CURSOR	FLDADV	HOME	PRINT	SYSREQ
CHARSET	CURSRSEL	FLDBKSP	INSERT	ROLLDOWN	TAB
CLEAR	DELETE	FLDMINUS	JUMP	ROLLUP	
CLEARPTN	DUP	FLDPLUS	LCLEAR	SCROLL	
CMD1-24	ENTER	FM	NL	SELECT	
COLOR	EREOF	HELP	PA1-3	SEND	
CTAB	ERIN	HIGHLITE	PF1-24	STRIPE	

- CPI-C Simulation Statements that are Delimiters[1]

CMALLC	CMCFMD	CMFLUS	CMRCV[2]	CMSEND
CMCFM	CMDEAL	CMPTR	CMRTS	CMSERR

The following sections describe how to code unconditional and conditional delimiters to send messages and to interrupt the message generation process.

Interrupting message generation with unconditional delimiters

When WSim encounters an unconditional delimiter during message generation, it stops processing the message generation deck and sends messages in the buffer to the system under test. Depending on the delimiter, WSim might or might not resume the message generation process automatically:

1. Only CPI-C statements that result in requests being issued to VTAM are delimiters. If the statement fails as a result of a local error that is detected before issuing a request to VTAM, the statement will not act as a delimiter. The types of errors that are typically detected as local errors are parameter checks and state checks.

2. The CMRCV statement will only be a delimiter if the receive type is receive-and-wait, and no data is available on the receive queue. When this statement acts as a delimiter, message generation stops for the transaction program until either data or status is received from the partner TP.

WAIT	Simulates the action of a terminal operator waiting for a reply before entering the next message. The WAIT statement interrupts message generation until a specified amount of time elapses, a specified condition is met, or a message is received from the system under test. If a message was generated, the WAIT statement also causes WSim to send the message in the buffer to the system under test.
STOP	Enables you to end message generation for a device without setting any indicators. The STOP statement interrupts message generation for a particular deck until WSim can reenter message generation. If a message was generated, the STOP statement also causes WSim to send the message in the buffer to the system under test.
QUIESCE	Simulates the action of an operator leaving the workstation unattended. The QUIESCE statement interrupts message generation until the QUIESCE indicator is turned off and all conditions for message generation were satisfied. If a message was generated, the QUIESCE statement also causes WSim to send the message in the buffer to the system under test. **Note:** Once you start sending an SNA chain of messages, the QUIESCE statement does not prevent reentry to message generation until an end-of-chain message is generated.

The following sections describe each unconditional delimiter and provide examples of the syntax required to code them in a message generation deck. "The WAIT statement," which follows this section, and "The QUIESCE statement" on page 147 also provide information about the WAIT, EVENT WAIT, and QUIESCE indicators.

The WAIT statement

The WAIT statement interrupts message generation for a device. You can use the WAIT statement to cause a device to wait for a specified amount of time, until a specified event occurs, until a specific message is received, or for some combinations of conditions. A WAIT statement that does not specify an EVENT sets the WAIT indicator. The device cannot reenter message generation until the WAIT indicator is reset. For a complete list of the actions that reset the WAIT indicator, see "Understanding the WAIT indicator" on page 144.

If you code a WAIT statement without an operand and with an IF statement logic test, you can cause a device to wait until a specific message is received from the system under test. When you code these statements together, the logic test specifies the message; the WAIT statement interrupts message generation for that device until the specified message is received. In this way, you can simulate an operator waiting for a specific reply before entering the next message.

Note: If you code the WAIT statement without an operand and not with a logic test, WSim interrupts message generation indefinitely. To reenter message generation when this occurs, you may enter the F (Console Recovery) operator command from the console.

For more information about logic tests, see Chapter 16, "Defining logic tests," on page 165. See *WSim User's Guide* for more information about the F operator command.

To interrupt message generation until a specific event takes place or for a specified amount of time, you can code one of the following operands on the WAIT statement:

- EVENT=*event*
- TIME={*integer* | *cntr*}[,UTI=*uti*]

When you code the EVENT operand, WSim interrupts message generation for a device until a specified event occurs. By coding the TIME and UTI operands, you can specify the maximum amount of time WSim interrupts message generation for a device.

The following sections describe the coding required for each operand.

Coding the EVENT operand

To simulate an operator waiting for an event to take place on another terminal, code the EVENT=*event* operand:

```
WAIT1  WAIT  EVENT=GOAHEAD  Turns on the WAIT EVENT indicator and interrupts
*                           processing until the event named GOAHEAD is posted.
```

If you specify the name of an event with the EVENT operand, that event must be posted before WSim continues message generation. You can take one of the following actions to post an event:

- Code POST=*event* on the EVENT message generation statement.
- Issue the A (Alter) operator command from the console using the POST=*event* parameter.
- Code the THEN=POST(*event*) or ELSE=POST(*event*) operands on an IF statement logic test.
- Code the THEN=POST(*event*) operand on an ON statement.

For more information about the EVENT statement, see Chapter 17, “Understanding control statements,” on page 195. For more information about the A (Alter) command, see *WSim User's Guide*. Chapter 16, “Defining logic tests,” on page 165 provides detailed information about posting events with a logic test.

The following example illustrates a WAIT statement that specifies the name of an event and the coding required to post that event.

```
DECK1   MSGTXT
*                   Sample Message Generation Decks
*   This example illustrates a WAIT statement and an EVENT statement
*   coded to simulate one resource waiting for an event to be posted
*   by a second simulated resource.
*
*                            Beginning of DECK1.
         .
         .                   Message generation statements.
         .
WAIT5   WAIT   EVENT=START   Specifies that WSim interrupt message
*                            generation for this device until the event
*                            named START is posted.
         .
         .                   Message generation statements.
         .
        ENDTXT               End of DECK1.
*
DECK2   MSGTXT
*                            Beginning of DECK2.
         .
         .                   Message generation statements.
         .
EVENT8  EVENT  POST=START    Posts the event named START.
         .
         .                   Message generation statements.
         .
        ENDTXT               End of DECK2.
```

After WSim processes WAIT5 in the preceding example, the first simulated resource cannot resume message generation until the event named START is posted by the second simulated resource.

Coding the TIME operand

To simulate a terminal operator waiting a specific amount of time for a reply to a message, code TIME=*integer* or TIME=*cntr* on the WAIT message generation statement. The value you code on the TIME operand specifies the amount of time, in seconds, that WSim interrupts message generation for a device.

The time specified on the TIME operand is the maximum amount of time that WSim interrupts message generation. If a message is received before the specified time elapses, WSim continues message generation without waiting for the time to elapse. When you code certain values on the TIME operand,WSim calculates the amount of time by multiplying a number by the active user time interval (UTI). For more information about the UTI, see Chapter 15, “Understanding intermessage delays,” on page 153.

You can code one of the following values for the TIME operand:

integer	Specifies a fixed number of seconds (this is never multiplied by the UTI) to be the maximum amount of time WSim interrupts message generation. *integer* is an integer 0 - 21474836.
A(*integer*)	Specifies that WSim selects a value from the range of zero to two times *integer*.WSim then multiplies the result by the active UTI value to determine the intermessage delay. *integer* is an integer 0 - 1073741823.
F(*integer*)	Specifies that WSim fixes the amount of time at the value of *integer* multiplied by the active user time interval. *integer* is an integer from 0 to 2147483647. If you specify F0, WSim processes the statement as if you had not coded the TIME operand.
R(*integer*)	Specifies that WSim selects a value randomly from the range defined on an RN statement and multiplies the value by the active user time interval. *integer* specifies the number of the RN statement and is an integer 0 - 255.
R(*integer1*,*integer2*)	Specifies that WSim selects a value randomly from the range specified by *integer1* and *integer2* and multiplies the value by the active UTI value. *integer1* is an integer from 0 to 2147483646 and *integer2* is an integer from 1 to 2147483647 or counter specifications whose values are within this range. *integer1* must be less than *integer2*.
T(*integer*)	Specifies that WSim selects a value randomly from the rate table specified by a RATE statement and multiplies the value by the active UTI value. *integer* specifies the name of the RATE statement and is an integer 0 - 255. For more information about using the RATE statement, see “Coding the DELAY operand” on page 158.
cntr	Specifies that WSim uses the value of a sequence or index counter as the amount of time. The counter's value must be within the range 0 to 2147483647. For any counter value, the maximum amount of time to wait is the value multiplied by the UTI value.

Note: You must code the parentheses for R(*integer1*,*integer2*) when you code the TIME operand. All other parentheses are optional.

When you code TIME=*cntr*, you can specify one of the following sequence or index counters for *cntr*:

NSEQ	References a network sequence counter.
LSEQ	References a line sequence counter.
TSEQ	References a terminal sequence counter.
DSEQ	References a device sequence counter.
NC*n*	References network index counter *n*.
LC*n*	References line index counter *n*.
TC*n*	References terminal index counter *n*.
DC*n*	References device index counter *n*.

The following example illustrates coding for TIME=*integer*.

```
WAIT2  WAIT  TIME=300      Turns on the WAIT indicator and interrupts
*                          processing for a maximum of 300 seconds
*                          or 5 minutes.
```

Coding the UTI and TIME operands

To interrupt message generation for a specific amount of time, you can also code the UTI=*uti* operand and the TIME operand on a WAIT statement. The UTI operand specifies the name of a UTI network definition statement. As discussed in Chapter 15, “Understanding intermessage delays,” on page 153, the UTI statement defines a user time interval, which is a scale factor in hundredths of seconds. When WSim processes the WAIT statement, it multiplies the amount of time coded on the TIME operand by the value coded on the specified UTI statement. If you do not code the UTI operand, WSim uses the UTI that is active for the device to calculate the delay.

The amount of time specified when you code the UTI and TIME operands is the maximum amount of time that WSim interrupts message generation. If a message is received before the specified time elapses, WSim continues message generation without waiting for the time to elapse.

Note: It is invalid to code the UTI operand without the TIME operand. In addition, it is invalid to code the UTI and TIME operands with the EVENT operand. Coding these statements in either manner results in a syntax error.

The following example illustrates a script with the coding required when you code the UTI operand:

```
NET1   NTWRK
*                     Sample WSim Script
*   This example illustrates the coding for a UTI network definition
*   statement and a WAIT message generation statement.  The UTI operand
*   references a user time interval specified on the UTI statement.
*
*                        Beginning of NET1.
       .
       .                 Network definition statements.
       .
UTIA   UTI   100         Specifies a delay of 1 second.
       .
       .                 Network definition statements.
       .
DECK1  MSGTXT
*                        Beginning of DECK1.
       .
       .                 Message generation statements.
```

```
        .
WAIT3  WAIT  TIME=F100,  Multiplies the value of TIME (100) by the
             UTI=UTIA    number of seconds represented by UTIA (1 second)
*                        and interrupts message generation for that number
*                        of seconds (100 seconds) or until a message
*                        is received.
        .
        .                Message generation statements.
        .
       ENDTXT            End of DECK1.
```

For more information about the UTI network definition statement, see "Coding the UTI statement" on page 155.

Understanding the WAIT indicator

During message generation, WSim turns on the WAIT indicator when you code the WAIT statement in one of the following ways:

- Without an operand
- With the TIME operand
- With the TIME and UTI operands.

You can also set the WAIT indicator by coding the following operands on the IF and ON statements:

- THEN=WAIT or ELSE=WAIT on the IF statement
- THEN=WAIT on the ON statement.

For more information about coding the IF statement, see Chapter 16, "Defining logic tests," on page 165. See "ON" on page 212 for more information about the ON statement.

A WAIT indicator that is turned on as a result of a WAIT statement coded without an operand is usually reset by a logic test that is evaluating messages received from the system under test. A WAIT indicator that is turned on as a result of a WAIT statement associated with a TIME value is reset when the specified time period elapses or a message is received from the system under test. If WSim receives a message from the system under test before the specified delay expires, WSim evaluates the message with any active logic tests and message generation continues.

The following example illustrates how WSim processes a WAIT statement and turns on the WAIT indicator.

```
MSG5   TEXT  (PW1)       Simulates an operator entering a password.
WAIT3  WAIT  TIME=60     Sends MSG5 and interrupts processing for 60
*                        seconds or until a message is received from
*                        the system under test.
```

First, MSG5 is sent. After 60 seconds elapse or when a message is received,WSim turns off the WAIT indicator and continues processing the deck.

If you want to restart message generation before the specified time elapses, enter the F (Console Recovery) operator command to place the specified resource in the console recovery mode. For more information about the F command, see *WSim User's Guide*.

The following provides a complete list of the conditions under which WSim resets the WAIT indicator:

- An S (Start) operator command is executed

- BIND is received (SNA LUs)
- An IF statement THEN= or ELSE=CONT (continue), B (branch), C (call), RETURN, QUIESCE, or RELEASE action is taken
- An ON statement THEN=CONT (continue), B (branch), C (call), RETURN, QUIESCE, or RELEASE action is taken
- The time coded on a WAIT statement's TIME operand expires
- A message is received after a WAIT TIME=*nn* statement is processed
- Automatic terminal recovery
- Console recovery mode is entered
- A console recovery subcommand is entered.

Preserving the WAIT indicator over asynchronous IF statements

During a simulation, the following situation sometimes occurs:

1. You want to check for a specific response to each message that you transmit
2. You receive some special sequences that require you to interrupt your normal flow to send a special key (such as to clear the screen)
3. After sending the special key, you want to continue waiting if you do not receive your expected response or to proceed to the next message to be transmitted if you do.

This can occur, for example, when you simulate a VM/CMS session and you receive “MORE”, “HOLDING”, or “VM READ” at some point, but you do not want to proceed until you receive a specific response. This can also occur when you simulate a TSO session, where “***” indicates that you need to clear the screen before proceeding.

The following example, although limited, shows how to avoid this problem. You can extend this example to handle cases where you want to check for responses on only some transmissions or where you want to check specific locations for each response. While these are more complicated and may require moving some network level IF statements into message decks, the general technique still applies.

This specific example checks for the VM “MORE...” or “HOLDING” sequence for a 24x80 3270 display and sends PA2 to clear the screen when this is received. It also checks for “VM READ” and sends ENTER in this case.WSim does not transmit the next regular message, however, until the data specified by the RESP operand for the previous message appears somewhere on the screen.

```
TESTNET  NTWRK  MSGTRACE=YES
* Switch 1 - on = waiting for response; off = response received
* Switch 2 - on = sending special sequence; off = sending normal text
* Note: You may want to use the SCAN operand on the NTWRK to handle
*        the situation where you never receive the expected response.
         .
         .
         .
*        network IFs
         IF   WHEN=OUT,LOC=SW2,THEN=SW2(OFF),ELSE=SW1(ON)
         IF   WHEN=OUT,LOC=SW1,THEN=WAIT
         IF   WHEN=IN,LOC=B+0,TEXT=RESP,SCAN=YES,THEN=SW1(OFF)
         IF   WHEN=IN,LOC=(24,61),TEXT=(MORE...),THEN=CSENDKEYS-PA2
         IF   WHEN=IN,LOC=(24,61),TEXT=(HOLDING),THEN=CSENDKEYS-PA2
         IF   WHEN=IN,LOC=(24,61),TEXT=(VM READ),THEN=CSENDKEYS-ENTER
         IF   WHEN=IN,LOC=SW1,ELSE=CONT
         .
         .
         .
```

```
SENDDATA MSGTXT
         TEXT  (data1),RESP=(respdata1)
         ENTER
         TEXT  (data2),RESP=(respdata2)
         ENTER
         TEXT  (),RESP=(respdata3)
         PF10
         .
         .
         .
         ENDTXT

SENDKEYS MSGTXT
* This example depends upon the fact that no TEXT statements are
* used in these special sequences, since this would reset the RESP
* data.  If TEXT statements were required here, the data would have
* to be put in a save area via a DATASAVE following each message in
* the primary message deck and the network IFs adjusted accordingly.
ENTER    ENTER
         SETSW  SW2=ON
         RETURN
PA2      PA2
         SETSW  SW2=ON
         RETURN
         ENDTXT
```

Understanding the EVENT WAIT indicator

When you specify the EVENT operand on the WAIT statement, WSim turns on the EVENT WAIT indicator and interrupts message generation until a specific event occurs (that is, until the event is posted). Like the WAIT indicator, with the EVENT WAIT indicator, you can simulate an operator waiting for a specific reply from the system under test or for an event to take place at another terminal. In addition, WSim may also turn the EVENT WAIT indicator on as a result of a logic test initiated through an IF message generation statement.

To turn off the EVENT WAIT indicator, you can post an event with the POST operand on the EVENT message generation statement. In the following example, the EVENT WAIT indicator is turned on for a simulated resource; however, as WSim processes a different message generation deck for another resource, that other resource posts the event.

```
DECK1  MSGTXT
*                     Sample Message Generation Decks
*   These decks illustrate the coding required to simulate one
*   resource waiting for an event to be posted by another resource.
*
*                           Beginning of DECK1.
MSG1   TEXT  (PW8)          Places the password into the buffer.
WAIT1  WAIT EVENT=GOAHEAD   Interrupts processing until event GOAHEAD
*                           is posted, turns on the EVENT WAIT indicator,
*                           and sends MSG1.
       .
       .                    Message generation statements.
       .
       ENDTXT               Specifies end of DECK1.

*
DECK2  MSGTXT
*                           Beginning of DECK2.
MSG1   TEXT  (USER5)        Places the user ID into buffer.
EVENT1 EVENT  POST=GOAHEAD  Posts event GOAHEAD.
       .
       .                    Message generation statements.
       .
       ENDTXT               End of DECK2.
```

When WSim processes DECK1 again, event GOAHEAD is already posted. Because the EVENT WAIT indicator was turned off when GOAHEAD was posted,WSim is able to continue processing the rest of DECK1.

As discussed in "Coding the EVENT operand" on page 141, you can also post an event with the A (Alter) command or an IF statement logic test. If you want to restart message generation before the event is posted, you can enter one of the following operator commands at the console:

A (Alter)	Specifies the name of an event for WSim to post when you specify POST=*event*.
F (Console Recovery)	Places the specified resource in the console recovery mode.

For more information about restarting message generation with operator commands, see *WSim User's Guide*.

The STOP statement

To stop message generation for a device and send any messages that are currently in the device buffer, code the STOP message generation statement. When WSim processes this statement, it stops message generation for the active device.

The STOP statement has no associated operands. The following example shows how this operand is coded.

```
STOP1  STOP         Stops message generation for a particular device.
```

As shown in the following example, WSim stops message generation and sends the message in the buffer, if any, when it encounters a STOP statement.

```
DECK1  MSGTXT
*                 Beginning of DECK1.
MSGA   TEXT  (CHANGE)  Places CHANGE into buffer.
STOP1  STOP            Sends MSG3 and stops message generation for the
*                      device.
       ENDTXT          End of DECK1.
```

When WSim processes the STOP statement in DECK1, message generation for that device stops and the message CHANGE is sent to the system under test. The STOP statement stops message generation for the active device for this pass through message generation.

The QUIESCE statement

To simulate a device that is logged on but unattended by its operator, code the QUIESCE message generation statement. When WSim processes a QUIESCE statement, message generation stops only for the active device.

Note: Once you start sending an SNA chain of messages, the QUIESCE statement does not prevent reentry of message generation until an end-of-chain message is generated.

The following example illustrates how to code the QUIESCE statement:

```
QUIESCE5  QUIESCE     Turns on the QUIESCE indicator and stops message
*                     generation for a particular resource.
```

In the following example, the QUIESCE statement sends the message in the buffer and ends message generation:

```
MSG1   TEXT  (PART NO. 1549)   Places PART NO. 1549 into buffer.
       QUIESCE                 Stops message generation for the device,
*                              prevents further message generation for
*                              the device, and sends MSG1.
```

After WSim processes the QUIESCE statement and turns on the QUIESCE indicator, the simulated resource can receive messages from the system under test and can evaluate messages received for any active logic tests. However, the resource cannot generate additional messages until the QUIESCE indicator is reset. While the QUIESCE indicator is on, the simulated resource responds negatively to polling from the system under test and does not generate any messages. To poll a device, the system under test sends a set of characters to a terminal. By responding to these characters, the terminal indicates whether it has a message to send.

WSim can also turn on the QUIESCE indicator as a result of an IF statement logic test or when you issue the QUIESCE operand on the A (Alter) operator command. To turn off the QUIESCE indicator, you can take one of the following actions:

- Issue the A (Alter) operator command with the RELEASE operand.
- Issue an R (Reset) operator command.
- Code THEN=RELEASE or ELSE=RELEASE on a logic test.

Note: For TCP/IP client simulations, the QUIESCE indicator prevents the device from automatically reconnecting.

To learn more about the IF statement, see Chapter 16, “Defining logic tests,” on page 165. For more information about issuing operator commands at the console, see *WSim User's Guide*.

Sending messages with conditional delimiters

When WSim processes a conditional delimiter after a message was generated or an AID byte was set, it interrupts message generation and sends the message in the buffer. When all conditions for message generation were satisfied, the device reenters message generation, and WSim continues processing the message generation deck. In contrast to unconditional delimiters, conditional delimiters act as delimiters only when a message exists in the buffer.

The following statements act as conditional delimiters:

TEXT	Interrupts message generation and sends the message in the buffer to the system under test. When the device reenters message generation,WSim places a new message into the buffer.
CMND	Interrupts message generation and sends the message in the buffer to the system under test. When the device reenters message generation,WSim places an SNA command into the buffer. The CMND statement is valid during SNA simulations only.
ENDTXT	Interrupts message generation, sends the message in the buffer to the system under test, and ends message generation for a particular device.

Note: If a message is generated by a TEXT statement and the cursor is moved with a cursor movement statement such as TAB, the next TEXT statement encountered by WSim does not act as a delimiter. Also, coding MORE=YES on the previous TEXT statement causes the current TEXT statement not to act as a delimiter.

As shown in the following example, a TEXT statement generates a message in the buffer at the current cursor location.

```
DECK1  MSGTXT
*                      Beginning of DECK1.
MSG1   TEXT  ($DATE$)  Places the current date into the buffer.
       ENDTXT          End of DECK1.
```

To send the message from the buffer to the system under test, you must code a delimiter. However, as discussed in “How delimiters are classified” on page 138, the preceding example already contains a conditional delimiter, the ENDTXT statement. When WSim processes MSG1, it places the current date into the buffer. Then WSim processes the ENDTXT statement, which indicates that WSim has reached the end of DECK1. Message generation is stopped at the ENDTXT statement, and MSG1 is sent to the system under test.

When you create message generation decks that are more complex than DECK1, do not use the ENDTXT statement as the only delimiter in the deck. If you code a deck with the ENDTXT statement as the only delimiter,WSim can enter an infinite processing loop within the message generation deck. Instead, you can use a second TEXT statement to send the message generated by the first TEXT statement. As a conditional delimiter, the second TEXT statement sends the message generated by the preceding TEXT statement.

The following example illustrates a series of TEXT statements serving as conditional delimiters:

```
DECK1  MSGTXT
*                        Beginning of DECK1.
MSG1   TEXT  ($DATE$)    Places the current date into the buffer.
MSG2   TEXT  (PW3)       Sends the current date to the system under test and
*                        ends this pass though message generation.  When
*                        WSim reenters message generation, it places PW3
*                        into the buffer.
MSG3   TEXT  (USER3)     Sends PW3 to the system under test and ends this
*                        pass through message generation.  When WSim
*                        reenters message generation, it places USER3
*                        into the buffer.
       ENDTXT            Sends USER3 to the system under test and specifies
*                        the end of DECK1.
```

Steps:

1. In the preceding example, WSim places MSG1 into the buffer and continues to process the message generation deck. When WSim processes MSG2, however, MSG2 acts as a delimiter and interrupts message generation. This pass through message generation ends, and WSim sends MSG1 to the system under test.
2. When all conditions for message generation were met, the device reenters message generation.WSim then places MSG2 into the buffer. For a list of conditions required for message generation, see “How delimiters affect the message generation process” on page 137.
3. In turn, MSG3 interrupts message generation until all required conditions were met. This pass through message generation ends, and WSim sends MSG2 to the system under test. When the device enters message generation again, WSim places MSG3 into the buffer.
4. Because MSG3 is in the buffer when WSim processes the ENDTXT statement, the ENDTXT statement serves as a delimiter.WSim stops message generation for DECK1 and sends MSG3 to the system under test.

Coding a script with delimiters

The following example illustrates how delimiters interact during message generation. In this example, a script provides two message generation decks named DECK1 and DECK2. DECK1 combines a TEXT, WAIT, and QUIESCE statement to simulate an operator logging on to a terminal and leaving the terminal unattended. DECK2 combines a TEXT, EVENT, and a STOP statement to illustrate another operator logging on to a different terminal. The EVENT statement turns off the EVENT WAIT indicator for DECK1.

```
NET1   NTWRK    UTI=100
*                        Sample WSim Script
*   This example illustrates a network definition and message generation decks
*   that simulate two operators logging on from two different terminals.
*
*                               Beginning of NET1.
        .
        .                       Network definition statements.
        .
UTI1   UTI      10              Specifies a user time interval of 0.1 second.
        .
        .                       Network definition statements.
        .
*
DECK1  MSGTXT
*                               Beginning of DECK1, associated with
*                               simulated resource 1.
MSG1   TEXT     (LOGON XY)      Places LOGON XY into buffer.
WAIT1  WAIT     EVENT=LOGONOK   Sends MSG1 and interrupts message generation
*                               until event LOGONOK is posted.
QUI1   QUIESCE                  Quiesces simulated resource and stops message
*                               generation for DECK1.
       ENDTXT                   End of DECK1.
*
DECK2  MSGTXT
*                               Beginning of DECK2.
MSGA   TEXT     (LOGON XY)      Places LOGON XY into buffer.
EVT1   EVENT    POST=LOGONOK    Posts LOGONOK.
STOP1  STOP                     Stops message generation and sends MSGA.
MSGB   TEXT     (HI)            Places message HI into buffer.
       ENDTXT                   Sends MSGB and specifies end of deck.
```

WSim processes DECK1 and DECK2 as follows:

Steps:

1. WSim processes DECK1, placing “LOGON XY” into the buffer. Then WSim processes WAIT1, which interrupts message generation for device 1 and sends MSG1 to the system under test. Device 1 cannot reenter message generation until event LOGONOK is posted.
2. WSim begins processing DECK2 (for a different device, device 2), placing “LOGON XY” into the buffer. Then WSim posts event LOGONOK and processes STOP1. STOP1 ends message generation for device 2 and sends MSGA to the system under test.
3. Because event LOGONOK has now been posted, device 1 is able to reenter message generation.WSim processes the QUIESCE statement, which interrupts message generation. At this point, device 1 receives messages from the system under test but cannot generate messages in return. To the system under test, this resource appears to be unattended by its operator.
4. When the second simulated resource is able to reenter message generation, WSim processes MSGB.

Coding delimiters in your message generation decks enables you to create complex simulations and interactions between resources. The next chapter, Chapter 15, “Understanding intermessage delays,” on page 153, describes additional methods of delaying message generation that enable you to simulate operator think time and the time required to enter data.

Chapter 15. Understanding intermessage delays

As discussed in Chapter 14, "Understanding delimiters," on page 137, you must code delimiters in your message generation deck to interrupt message generation, send messages to the system under test, and simulate interaction between simulated resources. In addition, you can simulate the operator think time and the time required to enter data by coding an intermessage delay. An intermessage delay is a period that WSim waits or delays after it exits message generation and before it reenters message generation.

When you code intermessage delays in your message generation decks, you can simulate operator think time and data entry time. For example, when WSim encounters a delimiter, it exits message generation for that device. When all conditions for message generation are satisfied, that device reenters message generation, and WSim continues processing the message generation deck. However, if you code an intermessage delay, WSim delays a specified amount of time before reentering message generation.

This chapter discusses the statements and operands you can use to specify an intermessage delay and provides information about the following topics:

- Specifying an intermessage delay
- Determining the start of an intermessage delay
- Specifying multiple user time intervals
- Altering user time intervals with the A (Alter) operator command
- Specifying delay values for individual resources
- Specifying intermessage delays for individual messages
- Coding a WSim script with intermessage delays.

Specifying an intermessage delay

WSim calculates an intermessage delay based on the following values that you code on network definition statements and on message generation statements:

- User time interval (UTI)
- Delay value.

A ***user time interval*** (UTI) is a scale factor in hundredths of seconds that can apply to an entire network, to individual network resources, or to an individual message. For example, if you code UTI=50, the user time interval is equal to 0.5 seconds (50 X 0.01 second).

In WSim, you can specify a user time interval on several different network definition and message generation statements. For example, you can specify a user time interval for an entire network by coding the UTI operand on the NTWRK statement:

```
NET1  NTWRK  UTI=integer   Syntax for the UTI operand.
```

WSim uses this user time interval to calculate the intermessage delay for every resource on the simulated network unless it is overridden by an individual UTI.

Note: If you specify UTI=0, WSim reenters message generation without calculating an intermessage delay.

With WSim, you can specify different user time intervals with the following network definition and message generation statements:

- UTI operand on the NTWRK definition statement
- UTI network definition statement
- IUTI operand on the DEV, LU, and TP network definition statements
- SETUTI message generation statement.

To create an intermessage delay, you also specify a ***delay value***, an integer that you can assign to each resource in a network or to each message. The integers that you assign enable you to create individual intermessage delays. If you do not specify a delay value, WSim uses a fixed default delay value of 1.

You can specify a delay value on the following network definition and message generation statements:

- DELAY operand on such statements as the DEV, LU, and TPnetwork definition statements. For a complete list of the network definition statements on which you can code the DELAY operand, see the *WSim Script Guide and Reference*.
- DELAY message generation statement.

WSim calculates an intermessage delay by multiplying the user time interval by the delay value. For example, if the active UTI for the device is 10 with a delay value of 20, WSim determines the intermessage delay using the following calculations:

1. WSim calculates the user time interval as 0.1 seconds (10 X 0.01 seconds).
2. WSim multiplies the user time interval by the delay value (0.1 seconds X 20) to calculate the intermessage delay, which is 2 seconds.

When WSim exits message generation, it waits for this 2-second intermessage delay to expire before reentering message generation.

The following table shows how WSim determines an intermessage delay using different user time intervals and delay values.

Table 9. How user time intervals relate to intermessage delays

Active UTI Value	User Time Interval	Delay Value	Intermessage Delay
UTI=1	0.01 seconds	100	1.0 second
UTI=10	0.1 seconds	5	0.5 second
UTI=100	1 second	20	20 seconds
UTI=1000	10 seconds	10	100 seconds

When you specify an intermessage delay, you can also specify when WSim starts calculating the delay. The following section describes how you specify the start of an intermessage delay with the THKTIME operand.

Determining the start of an intermessage delay with the THKTIME operand

With WSim, you determine the start of an intermessage delay by coding the THKTIME operand on statements such as the DEV and LU (VTAMAPPL) network definition statements. For a complete list of the network definition statements on which you can code the THKTIME operand, see the *WSim Script Guide and Reference*.

When you code this operand, you specify whether WSim starts to calculate the intermessage delay immediately or after all conditions for entering message generation have been satisfied.

To determine when WSim starts an intermessage delay, code one of the following values on the THKTIME operand:

IMMED	WSim calculates the intermessage delay immediately upon exiting message generation, even if the resource is not ready to generate another message. For example, WSim begins the delay for a resource even if a WAIT indicator remains on.
UNLOCK	WSim does not begin the intermessage delay until all conditions for message generation were satisfied and the terminal is ready to generate a message. When you simulate a 3270 display terminal, for example, the delay does not begin until all SNA responses were received, the terminal WAIT indicator is off, and the keyboard is unlocked.

Notes:
- IMMED is the default value for the THKTIME operand.
- When you code a WAIT statement with the TIME operand, it replaces the normal intermessage delay.

The following example shows how to code the THKTIME operand:

```
NET2  NTWRK
*                     Beginning of NET2.
DEV5  THKTIME=UNLOCK  Starts intermessage delays for DEV5 when the
*                     keyboard is unlocked.
DEV6  THKTIME=IMMED   Starts intermessage delays for DEV6 immediately
*                     upon exiting message generation.
       .
       .              Network definition statements.
       .
```

For more information about coding the THKTIME operand when you simulate SNA devices, see Chapter 18, “Generating messages for specific types of devices,” on page 219.

Specifying multiple user time intervals

When you code the UTI network definition statement, you can define multiple user time intervals for individual resources and messages. The following sections describe how to define multiple user time intervals with the UTI statement and how to reference the intervals from the network definition or from a message generation deck.

Coding the UTI statement

To define multiple user time intervals, code multiple UTI network definition statements as shown in the following example:

```
UTI1  UTI     100        Defines 1 second user time interval.
UTI2  UTI     50         Defines 0.5 second user time interval.
UTIA  UTI     1000       Defines 10 second user time interval.
```

You can code any number of UTI statements. By coding several statements, you can specify many different user time intervals and create many different intermessage delays. In this way, you can simulate devices operating at different speeds and change intermessage delays for individual messages.

Referencing multiple user time intervals

After you code network UTI statements, you can reference the individual user time intervals with one of the following operands:

- The IUTI operand on such statements as the SSCP, CNTLR, TERM, DEV, TP, and LU network definition statements
- The UTI operand on the WAIT, DELAY, and SETUTI message generation statements.

For a complete list of the statements on which these operands can be coded, see the *WSim Script Guide and Reference*.

The following sections describe how to code the IUTI operand and the SETUTI message generation statement to reference individual user time intervals. For more information about the DELAY message generation statement, see "Specifying intermessage delays for individual messages" on page 159. See Chapter 16, "Defining logic tests," on page 165 to learn about the UTI operand on the WAIT message generation statement.

Coding the IUTI operand

The IUTI operand specifies the name of a UTI network definition statement. The user time interval defined on the named UTI statement then becomes the individual user time interval for this device, overriding coding on the NTWRK statement.

The following example shows the coding required for this operand.

```
NET1  NTWRK
*                       Beginning of NET1.
       .
       .                Network definition statements.
       .
UTI1  UTI    100        Defines UTI1.
UTI2  UTI    50         Defines UTI2.
UTI3  UTI    10         Defines UTI3.
       .
       .                Network definition statements.
       .
DEV1  DEV    IUTI=UTI3  Specifies that WSim use the user time interval
*                       defined by UTI3 as the active user time interval
*                       for DEV1.
```

Coding the SETUTI statement

The UTI operand on the SETUTI message generation statement also references a UTI statement and changes the active user time interval for a simulated resource. The UTI operand value can be set to "NTWRKUTI", which references the network-level UTI. However, the user time interval specified by the SETUTI statement remains active only until WSim processes another SETUTI statement or the current deck's ENDTXT statement.

The following example illustrates changing the intermessage delay specified on the network definition with the SETUTI statement.

```
NET8    NTWRK   UTI=10
*                        Sample WSim Script
*   This script illustrates the coding required to change the intermessage
*   delay with the SETUTI statement.
*
*                            Beginning of NET8; defines a user time
*                            interval for all resources on NET8.
         .
         .                   Network definition statements.
```

```
        .
UTI1    UTI     500         Specifies a user time interval for UTI1.
UTI2    UTI     1000        Specifies a user time interval for UTI2.
        .
        .                   Network definition statements.
        .
DEV1    DEV     IUTI=UTI1   Specifies a user time interval equal to the value
*                           of UTI1 for DEV1.
        .
        .                   Network definition statements.
        .
DECK1   MSGTXT
*                           Beginning of DECK1 for DEV1.
MSG1    TEXT    (PW1)       Places PW1 into buffer.
SETUTI1 SETUTI  UTI=UTI2    Specifies new user time interval for DEV1.
MSG2    TEXT    (USER1)     Interrupts message generation and sends MSG1.
*                           When DEV1 reenters message generation, WSim
*                           places USER1 into buffer and calculates the
*                           intermessage delay.
        .
        .                   Message generation statements.
        .
         ENDTXT             End of DECK1.
```

When WSim processes DECK1, SETUTI1 defines a new user time interval for DEV1. This statement changes the user time interval to the value of UTI2, which is 1000 or 10 seconds, overriding the active user time interval specified by the DEV network definition statement named DEV1. WSim continues to use this value to calculate intermessage delays for DEV1 until it processes another SETUTI statement or reaches an ENDTXT statement.

Altering user time intervals with the A (Alter) operator command

You can change the value of a user time interval with the A (Alter) operator command. When you enter this command from the console, you specify the applicable network or resource and one of the following operands:

IUTI=*utiname*
: Specifies that WSim change the individual UTI for the specified resource to the value of *utiname*.

U=*integer*
: Specifies that WSim change the UTI specified on the NTWRK statement to an integer 0 - 65535.

UTI=(*utiname*,*integer***)**
: Specifies that WSim change the user time interval represented by *utiname* to an integer 0 - 65535.

In the following example, you code a user time interval of 100 on the NTWRK statement:

```
NET4   NTWRK   UTI=100   Specifies the user time interval for intermessage
*                        delays.
```

To alter the UTI specified in the preceding example, you can enter the A (Alter) operator command from the console, as follows:

```
A  NET4,U=10
```

WSim then changes the UTI specified on the NTWRK statement from a value of 100 to a value of 10.

For more information about the A (Alter) operator command, see *WSim User's Guide*.

Specifying delay values for individual resources

In WSim, you can define specific delay values that WSim uses to calculate intermessage delays. With the DELAY operand, you can specify delay values for individual resources.

For a complete list of the statements on which these operands can be coded, see the *WSim Script Guide and Reference*. The following sections describe how to code the DELAY operand.

Coding the DELAY operand

The DELAY operand enables you to change intermessage delays by coding different delay values for individual resources. When you code one of the following values on the DELAY operand, you can specify a delay value manually, generate a delay value randomly, or select a delay value from a rate table.

A(*integer*)	Specifies that WSim selects a delay value from the range of zero to two times *integer*. *integer* is an integer 0 - 1073741823.
F(*integer*)	Specifies that WSim uses a fixed delay value of *integer*. *integer* is an integer 0 - 2147483647.
R(*integer*)	Specifies that WSim selects a delay value randomly from the range defined on an RN statement. *integer* specifies the number of the RN statement and is an integer 0 - 255.
R(*integer1*,*integer2*)	Specifies that WSim selects a delay value randomly from the range specified by *integer1* and *integer2*. *integer1* is an integer 0 - 2147483646 and *integer2* is an integer 1 - 2147483647 or counter specifications whose values are within these ranges. The value for *integer1* must be less than the value for *integer2*.
T(*integer*)	Specifies that WSim selects a delay value randomly from the rate table specified by a RATE statement. *integer* specifies the number of the RATE statement and is an integer 0 - 255.

Notes:

- Each delay value specified by the DELAY operand must be multiplied by the active UTI to determine the intermessage delay.
- You must code the parentheses for R(*integer1*,*integer2*) when you code the DELAY operand. All other parentheses are optional.
- If you do not code the DELAY operand, the default delay value is F1. WSim then multiplies the default value by the active user time interval to calculate the intermessage delay.

If you specify an intermessage delay based on a rate table value with T(*integer*), you must define the table with a RATE network definition statement.

```
integer  RATE   member  Syntax for the RATE statement.
```

member specifies a rate table member in the partitioned data set defined by the RATEDD data set referenced in the JCL used when you run WSim. *integer* specifies a number that enables you to reference this statement. For more information about the RATE statement, see the *WSim Script Guide and Reference*. For more information about the RATEDD data set used to run WSim, see Part 1, "Defining WSim networks," on page 1.

The following example shows how to code the DELAY operand to specify a randomly selected delay value:

```
DEV1  DEV    DELAY=R(5,50)  Selects a delay value randomly from the
*                           range 5 to 50.
```

Each time WSim calculates a delay for DEV1, it selects a random number between 5 and 50 and multiplies that number by the active user time interval. For example, when the active UTI is 1000, the active user time interval is equal to 10 seconds. If WSim generates a random number 12 when calculating the delay, the new intermessage delay for DEV1 is equal to 12 times 10 seconds or 120 seconds.

Note: A different intermessage delay may be generated each time WSim calculates a delay for DEV1.

For detailed coding information about the DELAY operand, see the *WSim Script Guide and Reference*.

Specifying intermessage delays for individual messages

When you code the following operands on the DELAY message generation statement, you can change the intermessage delay for an individual message:

- UTI=*uti*
- TIME=*integer*|*cntr*.

The UTI operand overrides the active user time interval for a simulated resource by specifying the name of a UTI network definition statement. The user time interval defined by the named UTI statement temporarily becomes the device's active user time interval for that message. WSim calculates the intermessage delay by multiplying this user time interval by the value specified on the TIME operand.

Note: The UTI operand on the DELAY statement is valid only if you also code the TIME operand.

When you code the DELAY statement with the UTI and the TIME operands, WSim calculates a new intermessage delay that takes effect only the next time the device exits from message generation. For example, when WSim sends the first message following a DELAY statement, it calculates the intermessage delay defined by that DELAY statement. After WSim reenters message generation, the previously specified user time interval and delay value are again active.

You can code the following values for the TIME operand:

integer	Specifies a fixed value that WSim uses as the delay value. *integer* is an integer 0 - 2147483647.
A(*integer*)	Specifies that WSim selects a delay value from the range of zero to two times *integer*. *integer* is an integer 0 - 1073741823. *integer* is the average of the delay values selected by WSim.
F(*integer*)	Specifies that WSim fixes the delay value at the value of *integer*. *integer* is an integer 0 - 2147483647.

R(*integer*)	Specifies that WSim selects a delay value randomly from the range defined on an RN statement. *integer* specifies the number of the RN statement and is an integer 0 - 255.
R(*integer1*,*integer2*)	Specifies that WSim selects a delay value randomly from the range specified by *integer1* and *integer2*. *integer1* is an integer 0 - 2147483646 and *integer2* is an integer 1 - 2147483647 or counter specifications whose values are within these ranges. The value for *integer1* must be less than the value for *integer2*.
T(*integer*)	Specifies that WSim selects a delay value randomly from the rate table specified by a RATE statement. *integer* specifies the number of the RATE statement and is an integer 0 - 255. For more information about the RATE statement, see "Coding the DELAY operand" on page 158.
cntr	Specifies that WSim use the value of a sequence or index counter as the delay value.

Notes:
- WSim multiplies the delay value specified by the TIME operand by the active UTI to determine the intermessage delay.
- You must code the parentheses for R(*integer1*,*integer2*) when you code the TIME operand. All other parentheses are optional.

When you code TIME=*cntr*, you can specify one of the following sequence or index counters for *cntr*:

NSEQ	References a network sequence counter.
LSEQ	References a line sequence counter.
TSEQ	References a terminal sequence counter.
DSEQ	References a device sequence counter.
NC*n*	References network index counter *n*.
LC*n*	References line index counter *n*.
TC*n*	References terminal index counter *n*.
DC*n*	References device index counter *n*.

The following example illustrates the coding required to reference a user time interval defined on a UTI statement and calculate an intermessage delay for an individual message.

```
NET8    NTWRK   UTI=10
*                        Sample WSim Script
*   This script illustrates the coding required to change an intermessage
*   delay with the DELAY message generation statement.
*
*                           Beginning of NET8; defines a user time
*                           interval for all resources on NET8.
         .
         .                  Network definition statements.
         .
UTI1    UTI     500         Specifies a user time interval for UTI1.
UTI2    UTI     1000        Specifies a user time interval for UTI2.
         .
         .                  Network definition statements.
         .
DEV1    DEV     IUTI=UTI1   Specifies a user time interval equal to the
```

```
*                          value of UTI1 for DEV1.
        .
        .                  Network definition statements.
        .
DECK1   MSGTXT
*                          Beginning of DECK1 for DEV1.
MSG1    TEXT    (PW25)     Places PW25 into buffer.
DELAY1  DELAY   UTI=UTI2,  Specifies intermessage delay of 300 seconds (5
                TIME=F(30) minutes) to take place after WSim sends MSG1.
MSG2    TEXT    (USER25)   Interrupts message generation and sends MSG1.
*                          Before DEV1 reenters message generation, the
*                          delay calculated for DELAY1 must expire.  When
*                          DEV1 reenters message generation, WSim places
*                          USER25 into the buffer.
        .
        .                  Message generation statements.
        .
        ENDTXT             End of DECK1.
```

In the preceding example, the NTWRK statement and the UTI statements named UTI1 and UTI2 define user time intervals. The IUTI operand on the DEV statement named DEV1 specifies that the active user time interval for DEV1 is the same as the user time interval specified by UTI1. Because the DELAY operand is not coded on the DEV statement, the default delay value F1 is the active delay value for DEV1.

When you code DECK1 for device 1, you want to change the intermessage delay following MSG1. To override the active user time interval specified on the DEV statement and the default delay value, code the DELAY message generation statement named DELAY1. This statement specifies that WSim use the user time interval specified by UTI2 (10 seconds) and a fixed delay value of 30 to calculate the total intermessage delay, resulting in a delay of 300 seconds or 5 minutes.

When WSim processes MSG2, it interrupts message generation and uses DELAY1 as the intermessage delay. After the delay expires and all conditions for message generation were satisfied, DEV1 reenters message generation. WSim then continues processing the deck using the user time interval specified by the DEV statement and the default delay value.

Coding a script with intermessage delays

The following example illustrates a network definition and two message generation decks that show the coding required to define and change intermessage delays. This example demonstrates how to create simulations that represent actual message traffic with the system under test.

```
NET1    NTWRK   UTI=1          Beginning of NET1, specifies user time
*                              interval equal to 0.01 seconds.
*
*                        Sample WSim Script
*   This script illustrates the coding required to define and change
*   intermessage delays with operands on network definition and
*   message generation statements.
*
        .
        .                      Network definition statements.
        .
PATH1   PATH    DECK1          Defines a path for a device to follow
*                              during message generation.
PATH2   PATH    DECK2          Defines a path for a device to follow
*                              during message generation.
UTI1    UTI     100            Defines UTI1.
```

```
UTI2    UTI     500             Defines UTI2.
TCP1    TCPIP
DEV1    DEV     DELAY=R(1,10),  Defines delay for DEV1.
                THKTIME=UNLOCK, Determines the start of the delay for DEV1.
                PATH=(PATH1)    Specifies that DEV1 process the decks
*                               defined by PATH1.
DEV2    DEV     THKTIME=IMMED,  Starts the delay for DEV2 immediately.
                PATH=(PATH2)    Specifies that DEV2 process the decks
*                               defined by PATH2.
         .
         .                      Network definition statements.
         .
*
DECK1   MSGTXT
*                               Beginning of DECK1 for DEV1.
DELAY1  DELAY   UTI=UTI2,       Changes the active intermessage delay for
                TIME=F30        the next message by specifying the user time
*                               interval coded on UTI2 and a new delay value.
*                               WSim then calculates the intermessage delay
*                               specified by DELAY1.
MSG1    TEXT    (PW5)           Places PW5 into the buffer.
MSG2    TEXT    (USER5)         Interrupts message generation and sends MSG1.
*                               When all conditions for message generation
*                               have been met and after the delay expires,
*                               WSim reenters message generation and places
*                               USER5 into the buffer.
         .
         .                      Message generation statements.
         .
        ENDTXT                  End of DECK1.
*
DECK2   MSGTXT
*                               Beginning of DECK2 for DEV2.
SETUTI1 SETUTI  UTI=UTI1        Overrides NTWRK user time interval and
*                               sets DEV2 UTI to UTI1.
         .
         .                      Message generation  statements.
         .
        ENDTXT                  End of DECK2.
```

In this example, the UTI operand on the NTWRK statement defines a user time interval equal to 0.01 seconds for all resources on the network. In addition, a UTI statement named UTI1 provides an alternate user time interval equal to 1 second; UTI2 provides a user time interval equal to 5 seconds.

The network definition also provides two DEV statements, DEV1 and DEV2. The DELAY operand on DEV1 changes the active delay value to a random number 1 - 10. WSim uses this delay value to calculate intermessage delays for device 1. Also on DEV1, THKTIME=UNLOCK specifies that the intermessage delay does not start until DEV1 is ready to generate messages. On DEV2, THKTIME=IMMED specifies that WSim starts intermessage delays for device 2 immediately after ending message generation.

DECK1 is the message generation deck for device 1. WSim calculates the active intermessage delay for this device by multiplying the delay value defined on DEV1 by the user time interval specified on the NTWRK statement. The DELAY statement named DELAY1 changes this intermessage delay, however, during MSG2. Because THKTIME=UNLOCK for this device, WSim calculates the intermessage delay defined by the DELAY statement before it reenters message generation following MSG1: 5 seconds (UTI=UTI2) multiplied by 30 (TIME=F30). This calculation results in an intermessage delay equal to 150 seconds (2.5 minutes). This intermessage delay is active only for MSG1.

DECK2 provides a SETUTI statement that overrides the active user time interval specified on the NTWRK statement. The SETUTI statement specifies that WSim calculate the intermessage delay with the user time interval defined by UTI1, which is equal to 100 or 1 second, and the default delay value, F1. Every time WSim sends a message generated by DECK2, it delays 1 second before sending the next message (1 second X 1).

WSim processes the decks as follows:

Steps:

1. During message generation for DECK1, WSim processes MSG1, placing PW5 into the buffer. Then, WSim processes MSG2. Because this statement is a conditional delimiter, WSim exits message generation for device 1 and sends PW5 to the system under test. WSim then calculates the delay specified by DELAY1.
2. WSim begins processing DECK2. When WSim processes SETUTI1, the user time interval defined by UTI1 becomes the active user time interval for device 2. WSim continues to process DECK2 until it reaches a delimiter and exits message generation for device 2.
3. If all conditions for message generation were satisfied for device 1 and DELAY1 expired, WSim reenters message generation for device 1, places MSG2 (USER5) into the buffer, and resumes message generation. The intermessage delay defined by DELAY1 is no longer active. The user time interval now active for device 1 is calculated using the interval coded on the NTWRK statement and the delay value coded on DEV1.

Chapter 16. Defining logic tests

In WSim, you can create simulations that interact effectively with the system under test by coding the IF message generation statement. As discussed in Chapter 12, “Basic concepts,” on page 107, the IF statement defines a logic test based on messages transmitted or received by simulated resources. With the IF statement, you can also define logic tests that WSim evaluates immediately. These logic tests are not dependent on message traffic with the system under test.

An IF statement logic test performs the following functions in your message generation decks:

- Specifies comparisons that WSim performs on data sent or received by simulated resources
- Specifies comparisons that WSim performs with counters, switches, and save or user areas
- Specifies tests to determine whether events were completed
- Tests the current location of the cursor
- Alters the message generation process depending on the results of specified comparisons
- Sets switches, overrides normal SNA responses, cancels current delays, saves data, and logs messages.

In addition to IF statements in your message generation decks, you can include IF statements in your network definition. The IF statements in your network definition enable you to test messages sent or received for all devices in your network. This chapter describes the differences between network-level and message-level IF statements. It also describes how to code the IF message generation statement to include logic testing in your message generation decks.

Information provided in this chapter includes the following:

- Understanding network-level and message-level logic tests
- Coding IF statement operands
- Understanding logic test processing
- Understanding logic test examples
- Using logic tests to create self-checking scripts.

This chapter provides detailed examples throughout to help you understand how logic tests work and how you code logic tests in your message generation decks.

Understanding logic tests

You can define logic tests with IF statements coded on the network definition and in message generation decks. The following sections describe each type of logic test and provide a list of terms commonly used to describe message-level logic tests.

Network-level logic tests

You can specify network-level logic tests by coding IF network definition statements in the network definition. Unlike message-level logic tests that apply only to messages sent to or from a particular resource, network-level logic tests apply to all resources on a network and enable you to test every message going in or out of a network.

Note: Network-level logic tests are not evaluated for CPI-C simulations.

WSim defines network-level message generation statements as follows:

- You create network-level logic tests by coding IF message generation statements in the network definition just before the PATH statement.
- WSim activates network-level tests when initializing the network; the tests remain active throughout the simulation.
- WSim evaluates network-level tests in the order you code them on the network definition and before evaluating any message-level logic tests.
- WSim does not use the name field on a network logic test; it has no effect on processing order.
- You can code a maximum of 256 network logic tests in one network.
- You can code network logic tests for all resource types, including resources that cannot generate messages.

Because WSim does not require you to code a name for IF network definition statements, entries in the name field are optional:

```
name    IF   operand        Defines a network-level logic test.
```

When you code a network-level IF statement, WSim requires that you specify either the EVENT, LOC, LOCTEXT, or CURSOR operands:

EVENT=*event*
: Specifies the name of an event to be tested.

LOC=*location*
: Specifies the starting location of the data to be tested.

LOCTEXT={*cntr* **|** **(***data***)** **|** *integer***}**
: Specifies the data to be tested.

CURSOR=(*row***,***col***)**
: Specifies the cursor position to be compared with the current cursor position.

For detailed information about other operands you can code on the IF network definition statement, see the *WSim Script Guide and Reference*.

Message-level logic tests

You can specify message-level logic tests by coding IF message generation statements in your message generation decks. Unlike network-level logic tests that test every message going in or out of a network, message-level logic tests enable you to test individual messages sent to or from a particular resource.

WSim defines message-level logic tests as follows:

- You create message-level logic tests by coding IF message generation statements at any position in a message generation deck.
- Depending on the operands you code, message-level tests can apply to specific types of messages.
- WSim activates a message-level test only when processing the message generation deck containing the test.

- You can deactivate a message-level logic test during message generation.
- WSim requires that you code a unique number in the name field of each message-level IF statement where WHEN=IN or WHEN=OUT. The numbers you code control the order in which WSim evaluates these IF statements against messages sent to or received by simulated resources. For more information about numbering these operands, see "Evaluating logic tests" on page 183.
- You can code up to 4095 IF statements numbered 0 to 4094 that specify WHEN=IN and WHEN=OUT.
- You can create unlimited logic tests if you code WHEN=IMMED; WSim does not require you to name these IF statements. For more information about the WHEN operand, see "Coding the WHEN operand" on page 169.

You name a logic test by placing a number in the IF statement name field:

```
integer   IF   operand        Defines message-level logic test.
```

As with network-level logic tests, WSim requires that you code either the EVENT, LOC, LOCTEXT, or CURSOR operands:

EVENT=*event*
: Specifies the name of an event to be tested.

LOC=*location*
: Specifies the starting location of the data to be tested.

LOCTEXT={*cntr* **|** **(***data***)** **|** *integer***}**
: Specifies the data to be tested.

CURSOR=(*row***,***col***)**
: Specifies the cursor position to be compared with the current cursor position.

For information about other operands you can code on the IF message generation statement, see "Coding IF statement operands" on page 168.

Terminology used to describe message-level logic tests

To help clarify the explanations of logic testing provided in this chapter, the following list defines terms used to describe the logic testing process:

Activate
: WSim activates an IF statement as soon as it processes the statement during message generation. When an IF statement is activated, it is ready to perform a comparison.

Deactivate
: WSim deactivates an IF statement when it processes a DEACT statement. Under certain conditions, other message generation statements may also deactivate an IF statement. When an IF statement is deactivated, it can no longer perform a comparison.

Evaluate
: WSim evaluates an IF statement when the comparison is performed. An IF statement is evaluated differently depending on the value you code on the WHEN operand:
  - If you code WHEN=IN, WSim evaluates the IF statement when a message is received from the system under test.
  - If you code WHEN=OUT, WSim evaluates the IF statement when a message is sent to the system under test.

- If you code WHEN=IMMED, WSim evaluates the IF statement as soon as it processes the statement during message generation.

Test condition
The specified state that must exist if the result of the comparison is true.

Met or Not met
When WSim evaluates an IF statement, the test condition is met if the result of the comparison is true. The test condition is not met if the result of the comparison is false.

THEN or ELSE action
If the test condition is met, WSim takes the action specified on the THEN operand. If the test condition is not met, WSim takes the action specified on the ELSE operand. No action is taken under the following conditions:

- The test condition is met, and you did not specify a THEN action.
- The test condition is not met, and you did not specify an ELSE action.
- WSim did not evaluate the logic test. In this case, processing continues with no THEN or ELSE action being taken. See "Conditions under which a logic test is not evaluated" on page 185 for more information.

These terms are used throughout this book to describe logic testing. For clarification, you might want to refer to this list whenever you encounter one of the terms.

Coding IF statement operands

WSim provides operands on the IF message generation statement that help you create many different logic tests. Depending on the operands you code, you can test message traffic with the system under test or test the value of switches and counters. You can also specify actions to be taken after WSim completes the test.

The following provides a complete list of the operands you can code on IF message generation statements to create varied logic tests:

AREA	Specifies a save or user area location that stores the text WSim uses to test messages.
COND	Specifies the condition, such as equal (EQ) or not equal (NE), for which WSim is to make the comparison.
CURSOR	Specifies a cursor position to be compared with the current cursor position.
EVENT	Specifies the name of an event to be tested.
DELAY	Specifies that WSim cancels the active intermessage delay when it takes the specified THEN action. If you code the DELAY operand, you must also code the THEN operand.
DATASAVE	Specifies data to be saved in a save area when WSim tests a message and takes the THEN action. If you code the DATASAVE operand, you must also code the THEN operand.
ELSE	Specifies the action WSim takes if the test condition is not met.
LENG	Specifies the length of the text value in the user or save area named by the AREA operand.
LOC	Specifies the starting location of the data to be tested.
LOCTEXT	Specifies the data to be tested.
LOCLENG	Defines a maximum length to be associated with the LOC operand data.

LOG	Specifies data to be written in a log record when the test is made and the THEN action is taken.
RESP	Specifies that if the THEN action is not taken, WSim does not generate an automatic SNA response for this message.
SCAN	Specifies that WSim scans messages sequentially for data specified by the TEXT, AREA, or UTBL operand.
SCANCNTR	Specifies a counter to be set to the offset of a save or user area, buffer, or data stream where the test condition is met.
SNASCOPE	Specifies which SNA flow WSim tests for the data coded on the TEXT, AREA, or UTBL operands.
STATUS	Specifies that WSim keeps this IF statement active.
TEXT	Specifies the text value for which the test is made.
THEN	Specifies the action to be taken if the test condition is met.
TYPE	Specifies the type of terminal for which this IF statement is to be evaluated.
UTBL	Specifies the name or the number of a user table that contains entries to be compared with the data defined on the LOC operand.
UTBLCNTR	Specifies a counter to be set to the index of the user table entry that caused the logic test to be met.
WHEN	Specifies when WSim evaluates the logic test.

The following sections describe commonly used IF statement operands:
- "Coding the WHEN operand"
- "Coding the TEXT and AREA operands" on page 170
- "Coding the LOC operand" on page 171
- "Coding the LOCTEXT operand" on page 172
- "Coding the THEN and ELSE operands" on page 173
- "Coding the UTBL and UTBLCNTR operands" on page 178
- "Coding the SCANCNTR operand" on page 179.

For detailed information about other IF statement operands and their coding requirements, see the *WSim Script Guide and Reference*.

Coding the WHEN operand

The values that you code on the WHEN operand determine when WSim evaluates a logic test:

OUT	WSim evaluates the IF statement when a message is sent to the system under test.
IN	WSim evaluates the IF statement when a message is received from the system under test. This is the default if you do not code the WHEN operand.
IMMED	WSim evaluates IF statements to test switches, counters, save and user areas, events, or data in the buffer. **Note:** IMMED is the only meaningful value for CPI-C simulations.

Although you can have logic tests active for input and output messages at the same time, WSim evaluates active logic tests only against the appropriate type of message.

The following example illustrates how WSim evaluates IF statements against the appropriate type of message:

```
DECKA MSGTXT
*                                  Beginning of DECKA.
MSG1  TEXT   (ACCOUNT 15409)       Defines MSG1.
0     IF     LOC=U+0,TEXT=(SAVE),  Defines logic test 0 to test data in
             WHEN=IN,THEN=CONT     the user area against text SAVE.
1     IF     LOC=SW2,THEN=CONT,    Defines logic test 1 to test device
             WHEN=IN               switch 2.
2     IF     LOC=NSW32,WHEN=OUT,   Defines logic test to test network
             THEN=CONT,ELSE=WAIT   switch 32.
MSG2  TEXT   (data)                Defines MSG2.
      ENDTXT                       End of DECKA.
```

The following steps describe how WSim processes DECKA:

Steps:

1. WSim processes DECKA until it reaches the TEXT statement named MSG2. MSG2 causes the message generated by MSG1 to be sent. WSim then evaluates the logic test defined by IF statement 2, in which WHEN=OUT.
2. When WSim receives a message from the system under test, it evaluates the logic tests defined by IF statements 0 and 1.

For more information about the WHEN operand, see “Evaluating logic tests” on page 183.

Coding the TEXT and AREA operands

The values that you code on the TEXT and AREA operands specify the text or the location of the text that WSim uses as the basis of the logic test comparison:

- The TEXT operand specifies the exact text against which WSim makes the comparison.
- The AREA operand specifies the save or user area location that contains the text against which WSim makes the comparison.

When you code the TEXT operand, you can specify one of the following values:

RESP	Specifies that the data to be used in the comparison was specified on the RESP operand of the previous TEXT statement. If you did not code the RESP operand on the statement, WSim does not evaluate the logic test.
cntr	Specifies that the value of a counter is to be used in the comparison. You can specify a counter on the TEXT operand only if you also specify a counter on the LOC operand or if you also specify a counter or integer on the LOCTEXT operand. *cntr* is the name of a valid counter.
(*data*)	Specifies the data to be used in the comparison. You can also code data field options within the text delimiters.
'*xx*'	Specifies a test under mask, which compares a byte of data to the mask specified by the 2 hexadecimal digits within single quotation marks. If all of the bits set on in the mask byte are set on in the data byte, WSim takes the specified THEN action. If any of the bits are not set on, WSim takes the specified ELSE action.
integer	Specifies a 1- to 10-digit integer from 0 to 2147483647 to be used in the comparison. WSim compares the counter specified on the LOC or LOCTEXT operand or the integer specified on the LOCTEXT operand against *integer*. You can only code this value when you specify one of the above integers.

Note: WSim does not allow the TEXT operand when you specify a test on a switch with the LOC operand or when you code the AREA or LENG operands.

When you code the AREA operand, you can specify one of the following options, where *value* is an integer 0 - 32766 or the name of a counter 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*).

N±*value*	Specifies that the text is located at an offset from the start (+*value*) or back from the end (-*value*) of the network user area.
N*s*+*value*	Specifies that the text for the comparison is located at offset *value* from the start of a network save area *s*. *s* is an integer from 1 to 4095.
s+*value*	Specifies that the text for the comparison is located at offset *value* from the start of save area *s*. *s* is an integer from 1 to 4095.
U±*value*	Specifies that the text is located at an offset from the start (+*value*) or back from the end (-*value*) of the device user area.

Note: WSim does not recognize the AREA operand when you specify the name of a counter on the LOC operand or when you code the LOCTEXT or TEXT operand.

The following example illustrates how you code logic tests with the TEXT and AREA operands.

```
DECK1  MSGTXT
*                                  Beginning of DECK1
       .
       .                           Message generation statements
       .
5      IF      TEXT=(TEST DATA),   Defines a logic test comparing data at
               LOC=U+0,THEN=CONT   location U+0 against text TEST DATA.
       .
       .                           Message generation statements.
       .
6      IF     AREA=N+5,LENG=20,    Defines a logic test comparing data at
              LOC=B+1,THEN=CONT    location B+1 against 20 bytes of data
*                                  at a location offset 5 bytes into the
*                                  network user area.
       ENDTXT                      End of DECK1.
```

Coding the LOC operand

The LOC operand specifies the starting location at which WSim begins the comparing process for the logic test. When you specify one of the following options, WSim tests data against a counter's value or against the actual data stream, including any headers and control information. *value* is an integer from 0 to 32766 or the name of a counter whose value is within this range.

cntr	Specifies a test on the value of a counter. *cntr* is the name of a valid counter.
D+*value*	Specifies a test on the incoming or outgoing data stream at offset *value*.
RH+*value*	Specifies a test on the request/response header at offset *value*. This option acts the same as D+*value* for non-SNA devices.
RU+*value*	Specifies a test on the request/response unit at offset *value*. This option acts the same as D+*value* for non-SNA devices.
TH+*value*	Specifies a test on the transmission header at offset *value*. This option acts the same as D+*value* for non-SNA devices.

The B, C, and (*row*,*col*) values specify a logic test on the data as it appears in the buffer, excluding headers and control information. *value* is an integer 0 - 32766 or the name of a counter whose value is within this range.

B±*value*	Specifies that the test be at an offset from the start of data in the device buffer (+*value*), excluding headers, or back from the end of the data in the buffer (-*value*) for nondisplay devices. For display devices, -*value* specifies that the test be made at an offset back from the end of the screen image buffer.
C±*value*	Specifies that the test be at an offset from the cursor (+*value*) or at an offset back from the cursor (-*value*).
(*row*,*col*)	Specifies that the test be made at the specified row and column of the screen image of a display device. If specified for a nondisplay device, WSim ignores this value. *row* and *col* are integers 1 - 255 or names of valid counters.

To test whether network, terminal, and device switches are on, you can code the following values:

NSW*n*
NSW*n***&NSW***m***&...**
NSW*n***|NSW***m***|...**
Specifies that WSim tests one or a combination of the 4095 network-level switches. *n* and *m* are switch numbers from 1 to 4095. You can test combinations of switches by using the and (&) or the or (|) logic operator. Do not mix logic operators on one LOC operand.

TSW*n*
TSW*n***&TSW***m***&...**
TSW*n***|TSW***m***|...**
Specifies that WSim tests one or a combination of the 4095 terminal-level switches. *n* and *m* are switch numbers from 1 to 4095. You can test combinations of switches by using the and (&) or the or (|) logic operator. Do not mix logic operators on one LOC operand.

SW*n*
SW*n***&SW***m***&...**
SW*n***|SW***m***|...**
Specifies that WSim tests one or a combination of the 4095 device-level switches. *n* and *m* are switch numbers 1 - 4095. You can test combinations of switches by using the and (&) or the or (|) logic operator. Do not mix logic operators on one LOC operand.

Note: Different level switches might be used together.

The following example shows how to code the LOC operand:

```
0   IF   LOC=C+0,TEXT=(SAVE),      Defines logic test 0.
         TYPE=LU2,THEN=CONT
```

In the preceding example, LOC=C+0 causes WSim to begin comparing data at the first byte starting at the current cursor location. Because TYPE=LU2, WSim evaluates this logic test only for an SNA LU2 terminal.

For SNA terminals, WSim evaluates the following values on SNA response messages and on data transfers: D, TH, RH, RU, N, U, and *s*.

Coding the LOCTEXT operand

The LOCTEXT operand specifies the actual data to be compared. The LOCTEXT operand causes the IF statement to always be evaluated when the conditions specified on the TYPE, WHEN, and SNASCOPE operands are met. See *WSim Script Guide and Reference* for details about the LOCTEXT operand.

The following options are valid for the LOCTEXT operand:

cntr Specifies a counter whose value is to be used in the comparison. *cntr* is the name of a valid counter. See "Coding the LOC operand" on page 171 for a list of valid counters. WSim compares the counter or integer specified on the TEXT operand against *cntr*.

(*data***)** Specifies the data to be used in the comparison. You can also code data field options within the text delimiters.

integer Specifies a 1- to 10-digit integer from 0 to 2147483647 to be used in the comparison. WSim compares the counter or integer specified on the TEXT operand against *integer*.

Below gives an example of coding the LOCTEXT operand.

```
1  IF      WHEN=IMMED,LOCTEXT=($RECALL,B+45,5$),  Check 2 different screen
           COND=EQ,TEXT=($RECALL,B+132,5$),       locations to see if they
           THEN=NSW1(ON)                          are equal.
```

Note: LOCTEXT cannot be coded with LOCLENG, AREA, LENG, LOC, CURSOR, or EVENT.

Coding the THEN and ELSE operands

The values that you code on the THEN and ELSE operands name an action WSim takes when evaluating a logic test. The THEN operand defines the action WSim takes if the specified comparison is successful; the ELSE operand defines the action WSim takes if the specified comparison fails. If the test condition is met, WSim takes the action specified on the THEN operand. No action is taken if you did not code the THEN operand. If the test condition is not met, WSim takes the action specified on the ELSE operand. Again, no action is taken if you did not specify the ELSE operand.

When WSim takes an action following a logic test, indicators, switches, or message generation paths may be altered. When WSim does not take an action, all indicators, switches, and message generation paths remain as they were before WSim evaluated the IF statement.

You can specify actions to be taken following a logic test by coding one of the following values on the THEN and ELSE operands:

ABORT
Specifies that WSim aborts the current message generation deck, deactivates all active logic tests, and selects the next message generation deck for processing as specified by the path selection rules.

B (Branch)
Specifies a branch to another location within the message generation decks. A branch causes WSim to stop processing at one point in a deck and begin processing at another point in the same deck or in another deck. This action also resets the WAIT indicator.

C (Call)
Specifies a call for a label in the named deck or a label in the same deck WSim is processing. Unlike a branch, WSim saves a return pointer allowing message generation to return to the point of the call. This action also resets the WAIT indicator.

CONT (Continue)
: Specifies that message generation continues in the current message generation deck. This action also resets the WAIT indicator.

E (Execute)
: Specifies immediate execution of statements located at the named message generation deck.

IGNORE
: Specifies that WSim takes no action. In addition, WSim does not perform a WAIT, CONT, BRANCH, CALL, RELEASE, QUIESCE, ABORT, IGNORE, or RETURN action for the message being tested, even if the message meets a subsequent logic test's condition.

NSW(ON|OFF)
: Specifies that WSim sets on or clears all 4095 network switches.

NSW*n*(ON|OFF)
: Specifies that WSim sets on or clears network switch *n*. *n* is an integer 1 - 4095.

QUIESCE
: Stops message generation until WSim performs a release action. A quiesced device receives messages but responds negatively to polls and cannot generate messages. This action also resets the WAIT indicator.

QSIGNAL(*event*)
: Specifies that the named event is to be signaled for the active device only.

RELEASE
: Specifies that a quiesced device continue message generation. This action also resets the WAIT indicator.

RESET(*event*)
: Specifies that the named event is no longer posted.

RETURN
: Specifies that WSim returns to message generation after the point of the last call. If you have not issued any CALL statements, WSim writes a message trace (MTRC) record or STL trace (STRC) record to the log data set and ignores the return action. This action also resets the WAIT indicator in either case.

SIGNAL(*event*)
: Specifies that the named event is to be signaled.

SW(ON|OFF)
: Specifies that WSim sets on or clears all 4095 device switches.

SW*n*(ON|OFF)
: Specifies that WSim sets on or clears device switch *n*. *n* is an integer 1 - 4095.

TSW(ON|OFF)
: Specifies that WSim sets on or clears all 4095 terminal switches.

TSW*n*(ON|OFF)
: Specifies that WSim sets on or clears terminal switch *n*. *n* is an integer 1 - 4095.

VERIFY-(*data*)
: Causes WSim to log a VRFY record on the network's log data set.

WAIT Inhibits further entry into message generation.

WAIT (*event*)
Inhibits entry into message generation until the named event is posted.

The WAIT(*event*), POST(*event*), RESET(*event*), SIGNAL(*event*), and QSIGNAL(*event*) actions of the THEN and ELSE operands enable you to control events that you name. See the *WSim Script Guide and Reference* for information about the syntax required for these values. For more information about events, see “Controlling events” on page 209.

The B (Branch), C (Call), and E (Execute) actions enable you to specify the name of a message generation deck, the label of a statement in the current deck, or the name of a deck and the label of a statement in that deck. The following list illustrates these three coding methods using the B (branch) action.

BERROR	Branches to the deck named ERROR.
B-MSG1	Branches to the statement labeled MSG1 in the current deck.
BERROR-MSG1	Branches to the statement named MSG1 in the deck named ERROR.

The following example shows how to code the THEN and ELSE operands.

```
0    IF   LOC=B+5,TEXT=(HELLO),    Defines logic test 0.
          THEN=CONT,ELSE=WAIT
```

In this example, WSim tests incoming messages against the specified text HELLO. If an incoming message matches HELLO, WSim performs THEN=CONT, continuing the message generation process. If an incoming message does not match HELLO, WSim performs ELSE=WAIT and waits for another incoming message.

The following sections provide more information about two actions you can code on the THEN and ELSE operands:
- E (Execute)
- VERIFY-(*data*).

E (Execute)

E (Execute) specifies the location of a deck containing statements that WSim is to execute immediately while the logic testing procedure is active. The following list shows the types of statements WSim can execute as a result of the E (Execute) action.
- BRANCH
- CALC
- DATASAVE
- DEACT
- EVENT
- IF (WHEN=IN or OUT)
- LABEL
- LOG
- MONITOR
- MSGTXT
- ON
- OPCMND
- RESET
- SET
- SETSW

- WTO
- WTOABRHD.

If WSim encounters any other statement type when processing E (Execute), it ends the execute action and begins processing again.

The following example shows how the execute action affects message generation.

```
DECK1  MSGTXT
*                                  Beginning of DECK1.
MSG1   TEXT      (PW512)           Data for MSG1.
0      IF        LOC=B+0,TEXT=(LOGON OK),
                 THEN=EDECK2,SCAN=YES
WAITA  WAIT
       ENDTXT                      End of DECK1.
*
DECK2  MSGTXT
*                                  Beginning of DECK2.
       DATASAVE  LOC=*,AREA=U+0,LENG=10
1      IF        LOC=B+0,TEXT=(RESPONSE 1),
                 THEN=CONT,SCAN=YES
WTO1   WTO       (IF 1 ACTIVATED WAITING), Message to operator console.
                 ( ON RESPONSE 1)
       ENDTXT                      End of DECK2.
```

WSim processes DECK1 and DECK2 as described in the following steps:

Steps:

1. WSim begins processing DECK1, generating MSG1 and activating IF statement 0. When WSim processes the WAIT statement, it sends MSG1 to the system under test. When (and only when) the system under test returns the response LOGON OK and meets the conditions set by IF statement 0, WSim then begins to process DECK2, as specified by THEN=EDECK2.

   Note: If the THEN operand action had specified branch or call rather than execute, WSim would not have processed DECK2 until the next time message generation began for the resource.
2. WSim processes the DATASAVE statement, which specifies LOC=*. This operand specifies that WSim saves the data that satisfied the last IF statement for which it took the THEN action. In the preceding example, the data saved would be LOGON OK regardless of its offset into the received message.
3. After processing the DATASAVE statement, WSim activates IF statement 1 and processes the WTO statement. The WTO statement writes the message "IF 1 ACTIVATED WAITING ON RESPONSE 1" to the operator.
4. IF 1 resets the WAIT indicator if RESPONSE 1 is found in the data.

VERIFY-(data)

VERIFY-(*data*) causes WSim to log a VRFY record on the log data set. Each VRFY record contains information about a logic test, including the type of test, the expected or comparison data, and the actual data in the record. The Loglist Utility recognizes each VRFY record and produces verification reports based on this information.

By examining the Loglist Utility verification reports, you can tell if a logic test failed and, if so, exactly why it failed. If you include an optional description when you code the VERIFY action, WSim identifies specific logic tests in the Loglist Utility output and groups VRFY records into a verification summary report.

The following example shows how to code the VERIFY value on the ELSE operand:

```
0  IF  COND=EQ,                      Coding the verify action on an
       WHEN=IMMED,                   IF statement.
       TEXT=(HELLO),
       LOC=B+39,
       THEN=CONT,
       ELSE=VERIFY-(***FAILURE***)
```

If WSim does not find HELLO at offset 39 in the device buffer when processing IF statement 0, it logs a VRFY log record with the message “***FAILURE***”. When processed by the Loglist Utility, this VRFY record results in the following report:

```
                    VERIFICATION REPORT

DESCRIPTION   LOCATION LENG COND EXPECTED VALUE  ACTUAL VALUE
------------- -------- ---- ---- --------------- ------------
***FAILURE*** B+39        5 EQ   HELLO           BYE
```

From the sample verification report, you can see that WSim found the string “BYE” at the specified location, rather than “HELLO”.

Although the VERIFY action easily verifies data in a buffer as shown in the preceding example, you can use it with any valid combination of IF statement operands. For example, you can verify switch settings, counter settings, cursor locations, the status of events, and data in user or save areas.

For detailed information about coding VERIFY-(*data*), see the *WSim Script Guide and Reference*. For a detailed description of the Loglist Utility verification reports, see *WSim Utilities Guide*.

Coding THEN and ELSE on multiple IF statements

When multiple logic tests are active for a message generation deck, WSim evaluates the tests each time it sends messages to or receives messages from the system under test. Because the actions specified on the THEN and ELSE operands affect message generation, WSim can take only one of the following actions for each message sent or received:

- B (Branch)
- C (Call)
- CONT (Continue)
- IGNORE
- QUIESCE
- RELEASE
- RETURN
- WAIT.

Although WSim can take one of these actions only on a single data transfer, it could take many other actions, such as setting switches, posting events, and signaling events. WSim always performs these actions, even if it has already taken another action for that message.

The actions specified by the B (Branch), C (Call), CONT (Continue), RETURN, RELEASE, QUIESCE, and WAIT values on the THEN and ELSE operands do not take effect immediately upon evaluation of the logic test. These values alter message generation processing the next time WSim begins message generation for a resource. For example, the branch or call actions define a new entry point for message generation, enabling you to alter the message generation path dynamically depending on messages sent and received by a resource.

Each of these actions except WAIT also resets the WAIT indicator for a resource. For example, if the WAIT indicator is on for a device when WSim processes THEN=QUIESCE or ELSE=QUIESCE, it resets the WAIT indicator and turns the QUIESCE indicator on. These actions have no effect on the EVENT WAIT indicator.

The IGNORE action does not change pointers or indicators; it cancels the action specified on subsequent IF statements, such as B (Branch), C (Call), CONT (Continue), QUIESCE, RELEASE, RETURN, and WAIT, for the message being tested. IGNORE does not cancel other actions, such as setting switches, executing other message generation statements, or posting and signaling events.

You can use the IGNORE action to screen out messages WSim should ignore when determining correct responses or actions, as shown in the following example:

```
0  IF  LOC=RU+0,      WSim ignores the message READY when evaluating logic
       TEXT=(READY),  tests that specify the B, C, CONT, QUIESCE, RELEASE,
       THEN=IGNORE    RETURN, or WAIT actions.
```

Coding the UTBL and UTBLCNTR operands

The IF statement enables you to specify two operands that reference entries in a user table:

- UTBL={*integer*|*name*}
- UTBLCNTR=*cntr*.

When you code the UTBL operand instead of the TEXT or AREA operands, WSim compares each user table entry to the data at the location specified by the LOC or LOCTEXT operand.

Note: You cannot code the UTBL operand when you specify the name of a counter or switch on the LOC operand or when you specify the name of a counter or integer on the LOCTEXT operand.

WSim continues to compare the data until it finds an entry that matches the data or scans all of the entries. During the testing process, WSim may take one of the following actions:

- If WSim finds a user table entry that meets the specified condition, WSim performs the THEN action.
- If no entries meet the conditions set by the IF statement, WSim performs the ELSE action.
- If a comparison cannot be made because the length of the data specified by the LOC operand is less than the smallest UTBL entry and the LOCLENG operand is not specified, WSim takes no action.

To code the UTBL operand, you can specify 0 - 255 for *integer* or the name of an MSGUTBL statement.

The UTBLCNTR operand specifies that WSim set a network, line, terminal, or device counter to the index value of the UTBL entry that met the test condition. As discussed in “Generating messages with the $UTBL$ data field option” on page 123, WSim indexes tables entries beginning with zero.

To code the UTBLCNTR operand, you can specify the name of a valid counter for *cntr*.

For more information about the UTBL and UTBLCNTR operands, see “Example of logic testing for a display terminal using WHEN=IMMED” on page 188.

Coding the SCANCNTR operand

The SCANCNTR operand enables you to set a counter to the offset of the data that caused the logic test's condition to be met. When you code SCANCNTR=*cntr* and the condition is met, WSim assigns the value of the offset that satisfied the test condition to the specified counter. If the condition is not met, WSim does not change the value of the counter.

To code the SCANCNTR operand, you can specify the name of a valid counter for *cntr*, as shown in the following example:

```
1  IF      TEXT=(TEST MESSAGE),      Tests data in buffer at a zero offset
           LOC=B+0,SCANCNTR=DC1,     and sets DC1 to the offset.
           THEN=CONT,ELSE=WAIT
```

Notes:

- WSim does not allow the SCANCNTR operand when you code the following:
  - CURSOR and EVENT operands
  - The name of a counter or switch on the LOC operand
  - The name of a counter or integer on the LOCTEXT operand.
- WSim does not require you to code the SCAN operand when you code SCANCNTR. However, you still must code SCAN if you want WSim to scan for the data specified by the AREA, TEXT, or UTBL operands.
- If you specify the same counter on both the SCANCNTR and UTBLCNTR operands, the SCANCNTR operand takes precedence if the test condition is met.

The following list defines the offset returned by the SCANCNTR operand if the test condition is met.

LOC Operand	Offset Returned
B±*value*	The offset from the beginning of the device buffer.
C±*value*	The offset from the beginning of the device buffer.
(*row*,*col*)	The offset from the beginning of the device buffer.
N±*value*	The offset from the beginning of the specified save or user area.
N*s*+*value*	The offset from the beginning of the specified save or user area.
s+*value*	The offset from the beginning of the specified save or user area.
U±*value*	The offset from the beginning of the specified save or user area.
D+*value*	The offset from the beginning of the incoming or outgoing data stream.
TH+*value*	The offset from the beginning of the incoming or outgoing data stream.
RH+*value*	The offset from the beginning of the request/response header (RH) of the incoming or outgoing data stream.
RU+*value*	The offset from the beginning of the request/response unit (RU) of the incoming or outgoing data stream.

For the LOCTEXT operand, the list below defines the offset returned by the SCANCNTR operand if the test condition is met.

LOCTEXT Operand	Offset Returned
(data)	The offset from the beginning of the data specified.

The SCANCNTR operand can help you locate data when an offset is unknown or variable. For example, when the simulated resource interacts with a full-screen application, such as the Interactive System Productivity Facility (ISPF), WSim can display a list of data set names. To select a particular data set when you are uncertain of its location on the screen, you can code a logic test using the SCANCNTR operand, as shown in the following example.

```
DECK1     MSGTXT
*                       Sample Message Generation Deck
*   This message generation deck illustrates using the SCANCNTR operand
*   to code a logic test.
*
*                                Beginning of DECK1.
           .
           .                     Message generation statements.
           .
SCANLOOP  LABEL                  Specifies a label WSim uses to branch or
*                                call to this point in DECK1.
IFA       IF     WHEN=IMMED,     WSim scans the buffer for a data set
                 LOC=B+0,        named MYDATA.  If the test condition is
                 TEXT=(MYDATA),  met, WSim branches to MATCH and sets
                 SCAN=YES,       device counter 1 to the offset.
                 SCANCNTR=DC1,
                 THEN=B-MATCH
NOMATCH   PF8                    If the test condition is not met,
*                                WSim scrolls down, and
STOP1     STOP                   stops generating messages.
BRANCH1   BRANCH LABEL=SCANLOOP  WSim loops back to and begins processing
*                                at the statement labeled SCANLOOP.
MATCH     LABEL                  Specifies a label WSim uses to branch or
*                                call to this statement after it finds MYDATA.
CURSOR1   CURSOR OFFSET=DC1      WSim sets the cursor at an offset equal
*                                to the position of MYDATA.
BKTAB1    BTAB                   WSim tabs backward to an unprotected field,
MSG1      TEXT   (S)             selects MYDATA, and
ENTER1    ENTER                  sends MSG1 to the system under test.
          ENDTXT                 END of DECK1.
```

The following steps describe how WSim processes DECK1:

Steps:

1. WSim processes DECK1, activating the logic test defined by IFA. This logic test specifies that WSim scans the buffer for a data set named MYDATA. If it finds MYDATA, WSim sets device counter 1 to the offset location of MYDATA and branches to the LABEL statement named MATCH. If it does not find MYDATA, WSim scrolls down with the PF8 statement and then processes the STOP statement. The simulated resource appears to be waiting for a new screen.
2. When WSim finds MYDATA, it continues processing DECK1 from the LABEL statement named MATCH. WSim sets the cursor to the offset position of MYDATA and tabs backward to an unprotected field. WSim then selects the data set MYDATA by processing MSG1, ENTER1, and the ENDTXT statement.

Processing logic tests

The following sections discuss how WSim processes logic tests:

- Activating logic tests
- Deactivating logic tests
- Preventing the deactivation of logic tests
- Evaluating logic tests.

Activating logic tests

WSim activates a message-level logic test only when it processes the associated IF statement. When activating a logic test, WSim associates the IF statement with the terminal. WSim does not perform any testing or comparisons when activating a logic test; these actions cannot take place until WSim has something to compare.

Note: Only the last encountered IF statement with the same number is active. For example, when IF number 0 is active and another IF number 0 is encountered, the second IF replaces the first IF as the active IF.

In the following example, WSim generates the message QUERY NAMES and activates the logic test:

```
DECK9   MSGTXT
*                                   Beginning of DECK9.
MSG1    TEXT    (QUERY NAMES)       Defines MSG1.
0       IF      LOC=RU+0,           Tests incoming messages.
                TEXT=(READY),
                THEN=CONT,WHEN=IN
WAIT1   WAIT                        Stops message generation.
        ENDTXT                      End of DECK9.
```

The following steps describe how WSim processes DECK9:

Steps:

1. WSim processes MSG1 and places the message QUERY NAMES into the buffer. Then, WSim activates the logic test defined by IF statement 0. Because the statement specifies WHEN=IN, WSim cannot evaluate the logic test until it receives a message from the system under test.
2. Next, WSim processes the WAIT statement. Because the WAIT statement is a delimiter, WSim turns on the WAIT indicator and sends MSG1 to the system under test. Message generation for the simulated resource then ends. When WSim receives a message from the system under test, it compares the message with the test characters “READY”.
3. If the message from the system under test matches the test characters, WSim turns off the WAIT indicator and continues message generation for this resource. If the message does not match, WSim continues to wait for the next message received from the system under test. When WSim receives another message, it repeats the test process.

Note: Network-level logic tests are always active for all terminals to which they apply.

Deactivating logic tests

Although network-level logic tests remain active throughout a simulation, message-level logic tests remain active only while WSim processes the deck containing the IF statement. In addition, WSim can deactivate a logic test when it processes one of the following statements:

DEACT	Deactivates logic tests that specify WHEN=IN and WHEN=OUT. This statement does not apply to logic tests that specify WHEN=IMMED.
IF	Deactivates any preceding IF statement with the same name.
TEXT	Deactivates all currently active message-level logic tests unless STATUS=HOLD was coded on the IF statement.
CMND	Deactivates all currently active message-level logic tests for SNA devices unless STATUS=HOLD was coded on the IF statement.

In addition, WSim deactivates logic tests in a deck with no outstanding CALL statements when it reenters path selection after processing the deck's ENDTXT statement.

WSim normally deactivates logic tests after processing a TEXT statement, unless you code the IF statement with the STATUS=HOLD operand. The DEACT statement, however, deactivates all specified message generation logic tests, even if they specify STATUS=HOLD. For more information about the STATUS=HOLD operand, see "Preventing the deactivation of logic tests" on page 183.

Note: The DEACT statement does not deactivate network-level IF statements.

To deactivate an IF statement with the DEACT statement, you code the IFS operand:

```
ENDIF   DEACT   IFS=(0)        Deactivates IF statement 0.
```

In this example, the DEACT statement deactivates the IF statement before WSim processes any statement that could deactivate the logic test.

The following example shows two IF statements with the same number. When WSim processes the second IF statement, it deactivates the previous logic test and activates the new test.

```
DECK3 MSGTXT
*                                 Beginning of DECK3.
MSG1  TEXT    (PW23)              Message to be sent.
0     IF      LOC=RU+0,TEXT=(READY), Tests messages received from the
              THEN=CONT,WHEN=IN      system under test.
WAIT1 WAIT                        Stops message generation.
0     IF      LOC=RU+10,          Tests messages received from the
              TEXT=(RUNNING),     system under test.
              THEN=CONT,WHEN=IN
WAIT2 WAIT
*                                 Stops message generation.
      ENDTXT                      End of DECK3.
```

The following steps describe how WSim processes DECK3:

Steps:

1. WSim generates MSG1, activates the logic test defined by the first IF statement, turns on the WAIT indicator, and sends the message. When WSim receives the message READY from the system under test, it turns off the WAIT indicator.
2. When the device reenters message generation, WSim then processes another IF statement with the same label as the first IF statement. WSim activates the second logic test, which deactivates the previous test. At this point, WSim compares all incoming messages against the characters "RUNNING".

Note: When you simulate certain display devices, CLEAR, PA, or PF statements do not deactivate an IF statement. You must code a DEACT statement following these statements to deactivate a logic test. For more information about statements used to simulate display devices, see Chapter 18, “Generating messages for specific types of devices,” on page 219.

Preventing the deactivation of logic tests

The STATUS=HOLD operand on the IF statement enables you to prevent WSim from deactivating a logic test when it processes a TEXT or CMND statement. For example, you might want to send several messages to an application and test for the response ERROR following each message. Because a TEXT or CMND statement deactivates a message-level logic test, however, you must code an IF statement after each TEXT or CMND statement, as in the following example:

```
DECK3 MSGTXT
*                                 Beginning of DECK3.
MSG1  TEXT  (PART 1578)           Message 1.
0     IF    LOC=RU+10,TEXT=(ERROR),   Tests for response ERROR.
            WHEN=IN,THEN=CERRDECK
MSG2  TEXT  (PART 2990)           Message 2.
1     IF    LOC=RU+10,TEXT=(ERROR),   Tests for response ERROR.
            WHEN=IN,THEN=CERRDECK
MSG3  TEXT  (PART 9156)           Message 3.
2     IF    LOC=RU+10,TEXT=(ERROR),   Tests for response ERROR.
            WHEN=IN,THEN=CERRDECK
```

You can simplify the deck illustrated in the preceding example by coding the IF statement STATUS operand as shown in the following example. When you code STATUS=HOLD, WSim does not allow a TEXT statement to deactivate the IF statement:

```
DECK3 MSGTXT
*                                  Beginning of DECK3.
0     IF     LOC=RU+10,TEXT=(ERROR),   Tests for response ERROR to the
             WHEN=IN,THEN=CERRDECK,    following TEXT statements.
             STATUS=HOLD
MSG1  TEXT   (PART 1578)           Message 1.
MSG2  TEXT   (PART 2990)           Message 2.
MSG3  TEXT   (PART 9156)           Message 3.
```

If you code STATUS=HOLD, remember that a DEACT statement or selecting another path entry (from the PATH statement) still deactivates the logic test. Also, the IF statement can be deactivated by another IF statement with the same number.

Note: TEXT and CMND statements will first act as conditional delimiters and then deactivate logic tests. This allows WHEN=OUT logic tests to be evaluated before being deactivated.

Evaluating logic tests

During a simulation, WSim first evaluates all network logic tests in the order you coded them on the network definition. Then WSim evaluates message-level logic tests depending on the value you code for the WHEN operand:

IMMED If you specify WHEN=IMMED on an IF message generation statement, WSim evaluates the logic test immediately during message generation, not when it receives or sends messages from the system under test. Remember that WHEN=IMMED creates a logic test that tests switches, counters, save areas, user areas, events, or the device buffer, rather than message traffic between WSim and the system under test.

IN or OUT If you specify WHEN=IN or OUT, WSim evaluates both network-level and message-level logic tests when it receives a message from or sends a message to the system under test.

WSim alters the processing order for logic tests depending on how you code each IF statement's name field. WSim requires that you code a number for each message-level test containing the operands WHEN=IN or WHEN=OUT. The number you code for these tests can affect the order in which WSim processes each test.

For example, WSim processes each statement in a deck sequentially. However, when you code several logic tests with WHEN=IN, each test requires an incoming message before WSim can evaluate it. After receiving an incoming message, WSim evaluates the outstanding logic tests with WHEN=IN based on the number in the name field. If WSim has already activated IF statements 9, 3, and 25 where WHEN=IN, it evaluates the statements in ascending order, 3, 9, and 25, regardless of which statement WSim activated first.

The following example shows how WSim evaluates logic tests based on the number in the name field:

```
DECK1 MSGTXT
*                                Beginning of DECK1.
5     IF     WHEN=IN,TEXT=(HELLO), Tests for response HELLO.
             THEN=CONT
0     IF     WHEN=IN,TEXT=(BYE),   Tests for response BYE.
             THEN=WAIT
9     IF     WHEN=IN,TEXT=(ERROR), Tests for response ERROR.
             THEN=QUIESCE
WAIT1 WAIT                         Interrupts message generation.
MSG1  TEXT   (PART 7102)           Message 1.
1     IF     WHEN=IN,TEXT=(WHAT),  Tests for response WHAT.
             THEN=WAIT
```

The following steps describe how WSim processes DECK1:

Steps:

1. WSim activates IF statement 5, then IF statement 0, and finally IF statement 9.
2. If WSim receives a message from the system under test, it evaluates the three active IF statements in the following order: IF statement 0, IF statement 5, and IF statement 9. WSim does not evaluate IF statement 1 because it has not yet activated that statement.

When WSim sends a message to the system under test, it evaluates IF statements with WHEN=OUT in the same manner as it does for WHEN=IN. For this reason, remember to assign unique numbers to each IF statement that specifies WHEN=IN or OUT.

To code the name field on an IF statement that specifies WHEN=IN or OUT, enter an integer from 0 to 4094. WSim can maintain a total of 4095 active IF statements that specify either WHEN=IN or WHEN=OUT at any one time.

Note: If you code WHEN=IMMED, WSim does not require an entry in the name field. Therefore, you can code any number of IF statements that specify WHEN=IMMED.

Conditions under which a logic test is not evaluated

As stated earlier, the WHEN and TYPE operands restrict the data transfers for which a logic test is evaluated. The following rules outline all of the conditions for which a logic test is not evaluated for a transmitted or received message:

- The IF statement specifies WHEN=IN for a transmitted message or WHEN=OUT for a received message.
- The terminal type specified by the IF statement TYPE operand does not match the type of terminal associated with the message.
- The SNA flow type specified by the IF statement SNASCOPE operand does not match the flow type.
- A user area or a save area was specified by the IF statement LOC or AREA operand, and the area does not exist for the terminal and the LOCLENG operand is not coded.
- The IF statement LOC operand specifies (*row*,*col*) for a non-display terminal and the LOCLENG operand is not coded.
- The beginning location for the test as specified by the LOC or AREA operand is not within the available data and the LOCLENG operand is not coded.
- The ending location for the test as specified by the LOC and TEXT operands or the AREA and LENG operands is not within the available data and the LOCLENG or LOCTEXT operand is not coded.
- The LOC operand is set to a null value (no data) and the LOCLENG operand is not coded.
- The TEXT operand is set to a null value (no data) and the LOCLENG or LOCTEXT operand is not coded.
- The TEXT=RESP operand is coded but a RESP was not coded on the previous TEXT statement.
- When the value of LOC is greater than the length of the save area and the LOCLENG operand is not coded.
- The IF statement CURSOR operand specified (*row*, *col*) and the value is not a valid screen position for a display device or it is a non-display device.

Logic testing DBCS data

In the following examples and discussion, the SO character is represented using a “<”, the SI character is represented using a “>”, and the first byte of each DBCS character, which is referred to as the ward byte, is represented using a “.” character.

DBCS data can be identified on a simulated 3270 screen by either being in a DBCS field, having SO and SI characters wrapped around the DBCS data, or having the character attributes indicate DBCS data. The DBCS data in a message sent or received may or may not have SO and SI characters included.

Because of the above situation, logic testing DBCS data must be carefully thought out. For example, if the DBCS data “.A.B.C” is located in a DBCS field or identified using character attributes as DBCS data, you need to code the logic test as follows to obtain a true result. Use the DATASAVE DBCSDEL function to delete the SO and SI characters from the literal text DBCS data. DBCS data continues with a “+” or “,” must end with an SI and start with an SO on the next line. The SI/SO pairs at the continuation are ignored when the data is processed. Only the beginning SO and ending SI are considered.

```
   DATASAVE AREA=1,            save area 1 = .A.B.C
            FUNCTION=DBCSDEL,  delete SO and SI characters
            TEXT=(<.A.B.C>)    DBCS data with SO and SI characters

   IF       WHEN=IMMED,        immediate if to
            SCAN=YES,          scan the
            LOC=B+0,           simulated screen image for
            TEXT=($RECALL,1$), DBCS .A.B.C without SO/SI and
            THEN=B-OK          branch to label OK if .A.B.C found

   WTO      (NOT FOUND)

OK LABEL
```

If the DBCS data on the screen contains SO and SI characters such as "<.A.B.C>", the DATASAVE DBCSDEL function to delete the SO and SI characters is not needed. The following logic test obtains a true result in this case.

```
   IF       WHEN=IMMED,        immediate if to
            SCAN=YES,          scan the
            LOC=B+0,           simulated screen image for
            TEXT=(<.A.B.C>),   DBCS .A.B.C with SO/SI and
            THEN=B-OK          branch to label OK if <.A.B.C> found

   WTO      (NOT FOUND)

OK LABEL
```

Logic test examples

The following examples demonstrate how different logic tests affect the message generation process. Each example includes the following information:

- A description of the example, including its purpose
- Output from the Preprocessor showing the message generation decks with statement numbers
- A step-by-step description of how WSim processes the script
- Formatted message generation trace (MTRC) records produced by the Loglist Utility.

The examples in this section illustrate the logic test process with sample MTRC records and message deck listings produced by the Preprocessor. WSim writes MTRC records to the log data set when you code MSGTRACE=YES on the NTWRK statement. The Loglist Utility then formats these records, providing a trace listing of how WSim processes message generation decks and the activity or inactivity of network- and message-level logic tests.

The statement numbers on the MTRC records correspond to the statement numbers on the message deck listings produced by the Preprocessor. The descriptions of each example use these numbers to demonstrate the effect each statement has on the message generation process.

For additional information about MTRC records and the Preprocessor, see *WSim Utilities Guide*.

Example illustrating logic testing

The example below illustrates several different logic tests. The network is designed to send a message and then wait for a return echo message before proceeding with message generation.

```
         TESTNET  NTWRK MSGTRACE=YES
*                           Sample WSim Script
*   This script illustrates the coding required to send a message and
*   wait for a return echo message before proceeding with message
*   generation.
*
*                                               Beginning of TESTNET,
*                                               specifies MTRC records
*                                               be written to the log
*                                               data set.
                  .
                  .                             Network statements.
                  .
         A        IF    LOC=B+0,TEXT=(*),       Network logic test for
                        ELSE=WAIT,WHEN=OUT      messages sent to the
*                                               system under test.
                  .
                  .                             Network statements.
                  .
         TESTMSG  MSGTXT
*                                               Beginning of deck named
*                                               TESTMSG.
* STMT#
00001    MSG1     TEXT  (TEST MSG ONE),
                         RESP=(TEST)            Defines MSG1, specifies
*                                               data to be compared for
*                                               IF statements where
*                                               TEXT=RESP.
00002    0        IF    LOC=B+0,TEXT=RESP,
                        THEN=CONT,STATUS=HOLD   First logic test.
00003    MSG2     TEXT  (TEST MSG TWO)          Defines MSG2, sends MSG1.
00004    1        IF    LOC=B+0,TEXT=(TEST),
                        THEN=CONT               Second logic test.
00005    MSG3     TEXT  (TEST MSG THREE)        Defines MSG3, sends MSG2.
00006    0        IF    LOC=B+5,TEXT=(MESS),
                        THEN=CONT               Third logic test.
00007    MSG4     TEXT  (* LAST TEST MSG)       Defines MSG4, sends MSG3.
00008             ENDTXT                        Sends MSG4, ends processing.
```

In the preceding example, network-level logic test A turns the WAIT indicator on for each message sent to the system under test unless the first character sent is an asterisk (*). The WAIT indicator stops WSim from processing the message generation deck any further.

Each of the following steps provides the MTRC records formatted by the Loglist Utility to illustrate how WSim processes the preceding script.

Steps:

1. After WSim initializes the network, it processes TESTMSG beginning with the first TEXT statement. WSim places the data for MSG1 into the buffer and saves the data defined by the RESP operand. Then, WSim continues message generation, processing IF statement 0 and activating the logic test.

   When WSim encounters the second TEXT statement, message generation stops and WSim sends MSG1. WSim then evaluates network-level IF statement A; because WSim does not find an asterisk (*) in the data, it turns on the WAIT indicator. When WSim receives a message from the system under test, it then evaluates IF statement 0 using the RESP data saved from MSG1, TEST. If the message received does not match TEST, the test fails and no further processing takes place. The WAIT indicator is still on, and WSim continues to test all incoming messages. When an incoming message matches TEST, processing continues as specified by THEN=CONT. This coding enables WSim to resume message generation at the current point in the message generation deck.

```
ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK TESTMSG
ITP448I MSG GEN ENDED:   STMT# 00003 OF DECK TESTMSG
ITP430I OUTPUT IF A ( NETWORK IF  ) NOT MET - ELSE ACTION TAKEN:  WAIT INDICATOR SET
ITP427I INPUT  IF 0   (TESTMSG  00002) MET - THEN ACTION TAKEN:  CONTINUE (RESET WAIT)
```

2. WSim then generates MSG2, activates the logic test defined by IF statement 1, and sends the message. WSim then evaluates network-level IF statement A; because WSim does not find an asterisk (*) in the data, it turns on the WAIT indicator. Because IF statement 0 specifies STATUS=HOLD, that logic test remains active. However, because the previous MSG2 did not specify the RESP operand, WSim does not evaluate the test when it receives the return message. Now, WSim evaluates the logic test defined by IF statement 1, and the THEN action clears the WAIT indicator.

```
ITP447I MSG GEN ENTERED: STMT# 00003 OF DECK TESTMSG
ITP448I MSG GEN ENDED:   STMT# 00005 OF DECK TESTMSG
ITP430I OUTPUT IF A   ( NETWORK IF   ) NOT MET - ELSE ACTION TAKEN:  WAIT INDICATOR SET
ITP424I INPUT  IF 0 (TESTMSG  00002) NOT EVALUATED - END OF TEST NOT WITHIN DATA
ITP427I INPUT  IF 1   (TESTMSG  00004) MET - THEN ACTION TAKEN:  CONTINUE (RESET WAIT)
```

3. WSim generates MSG3, and the new IF statement 0 overrides the first IF statement 0. Processing stops on the fourth TEXT statement, and WSim sends the MSG3. WSim evaluates the message sent against network-level IF statement A; because WSim does not find an asterisk (*) in the data, it turns the WAIT indicator on. When the system under tests returns a message, WSim evaluates the new logic test. Again, WSim finds the data and takes the THEN action.

```
ITP447I MSG GEN ENTERED: STMT# 00005 OF DECK TESTMSG
ITP448I MSG GEN ENDED:   STMT# 00007 OF DECK TESTMSG
ITP430I OUTPUT IF A   ( NETWORK IF   ) NOT MET - ELSE ACTION TAKEN:  WAIT INDICATOR SET
ITP427I INPUT  IF 0   (TESTMSG  00006) MET - THEN ACTION TAKEN:  CONTINUE (RESET WAIT)
```

4. WSim then processes the last TEXT statement. Now, WSim deactivates the message-level logic test defined by IF statement 0, because it did not specify STATUS=HOLD. WSim enters MSG4 into the buffer. WSim processes the ENDTXT statement where message generation stops, and WSim sends MSG4. WSim then evaluates network-level IF statement A. Because the data includes an asterisk (*), WSim does not turn on the WAIT indicator.

```
ITP447I MSG GEN ENTERED: STMT# 00007 OF DECK TESTMSG
ITP448I MSG GEN ENDED:   STMT# 00008 OF DECK TESTMSG
ITP426I OUTPUT IF A  ( NETWORK IF  ) MET - THEN ACTION NOT CODED
```

The next time the terminal is ready to generate a message, WSim processes the deck again.

```
ITP447I MSG GEN ENTERED: STMT# 00008 OF DECK TESTMSG
ITP449I MSG GEN CONTINUES: DECK TESTMSG  STARTED
ITP448I MSG GEN ENDED:   STMT# 00003 OF DECK TESTMSG
ITP430I OUTPUT IF A ( NETWORK IF  ) NOT MET - ELSE ACTION TAKEN:  WAIT INDICATOR SET
ITP427I INPUT  IF 0   (TESTMSG  00002) MET - THEN ACTION TAKEN:  CONTINUE (RESET WAIT)
```

Example of logic testing for a display terminal using WHEN=IMMED

The example shown below illustrates logic testing for a display device when WHEN=IMMED. WSim compares the display buffer at a specific location to all the entries of an input message user table using the $UTBL$ data field option. If the message received matches one of the input message user table entries, WSim sets a device index counter to the index of the matching entry. If the message received does not match any of the input message user table entries, WSim issues a message to the WSim operator and sets a device index counter to the index of the error response message entry. The next message generated includes data from a response message user table, using a device index counter to access the appropriate table entry. Formatted MTRC records produced by the Loglist Utility follow the step-by-step processing descriptions.

```
          LU2EX   NTWRK  MSGTRACE=YES
*                Sample WSim Script
*   This script illustrates logic testing for a display device.
*
          0       UTBL   (MSG1),           Defines input user table; WSim
                         (MSG2),           compares each message received
                         (MSG3),           with the entries in this table.
                         (MSG5)
          1       UTBL   (RESPONSE MSG1),  Defines response message user
                         (RESPONSE MSG2),  table with response messages
                         (RESPONSE MSG3),  as per messages received in
                         (RESPONSE MSG5),  user table above.
                         (WHAT?)           Error response message.
          0       PATH   LOOPER            Defines path for deck LOOPER.
          APPL1   VTAMAPPL
          LU1     LU

          LOOPER  MSGTXT
*                                        Beginning of deck LOOPER.
* STMT #
00001     MSGA    TEXT   (STARTING)      Defines first message.
00002     TOPLOOP ENTER                  Sets ENTER AID byte.
00003     STOP1   STOP                   Stops message generation.
00004     0       IF     LOC=(24,1),     Row 24, column 1 of display.
                         UTBL=0,         Compare to user table entry 0.
                         UTBLCNTR=DC1,   Return user table entry number
                         COND=EQ,        in device index counter.
                         WHEN=IMMED,     Search for equal compare.
                         ELSE=C-ERROR    Call ERROR if not found.
00005     MSGB    TEXT   ($UTBL,1,CD1$)  Pick up next message from the
*                                        response user table using index
*                                        counter 1.
00006     GOTO1   BRANCH LABEL=TOPLOOP   Continue in loop.
00007     ERROR   WTO    (UNEXPECTED ),  Tell the operator.
                         (RESPONSE ),
                         (RECEIVED FROM ),
                         (SYSTEM)
00008     SET1    SET    DC1=5           Set user table entry number
*                                        for WHAT? message.
00009     RTN1    RETURN                 Return to caller.
00010             ENDTXT                 End of deck LOOPER.
```

Steps:

1. After initializing the network, WSim waits for the host application program to establish the SNA session with device LU1.
2. Device LU1 enters message generation, and WSim places the message STARTING into the display buffer. WSim executes the ENTER statement, which sets the AID byte to indicate an operator pressing the ENTER key. WSim then executes the STOP statement, stopping message generation before it evaluates the IF logic test coded with WHEN=IMMED.

   ```
   ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK LOOPER
   ITP448I MSG GEN ENDED:   STMT# 00004 OF DECK LOOPER
   ```
3. Device LU1 receives one or more response messages. The host application program unlocks the keyboard after sending the last response message to device LU1.
4. Device LU1 enters the message generation process. WSim evaluates the IF logic test coded with WHEN=IMMED, which compares the data in the display buffer at row 24 column 1 to the data in each entry of user table 0, the input message user table. If WSim finds a match, it sets device index counter 1 to the index of the user table entry that matches the data in the display buffer.

If WSim does not find a match, WSim calls label ERROR to do the following actions:

- Issue a WTO to the operator
- Set device index counter 1 to the index of the error response message entry in user table 1
- Return control to the next statement after the immediate IF, by the RETURN statement.

WSim executes the TEXT statement and copies the data from the entry specified by device index counter 1 in user table 1 to the display buffer. WSim sends RESPONSE MSG1 when it receives MSG1; WSim sends RESPONSE MSG2 when it receives MSG2, and so on. WSim executes the BRANCH statement, branching up to label TOPLOOP. Execution of the ENTER statement sets the AID byte to indicate an operator pressing the ENTER key. Execution of the STOP statement stops the message generation process before WSim evaluates the immediate IF logic test. The following MTRC entries demonstrate both an expected response and an unexpected response.

Note: Steps 3 and 4 are performed until WSim stops the network.

Expected response received

WSim generates the following MTRC records when an expected response is received:

```
ITP447I MSG GEN ENTERED: STMT# 00004 OF DECK LOOPER
ITP426I IMMEDIATE IF   (LOOPER   00004) MET - THEN ACTION NOT CODED
ITP450I BRANCH FROM STMT# 00006 OF DECK LOOPER TO TOPLOOP AT 00002 OF DECK LOOPER
ITP448I MSG GEN ENDED:   STMT# 00004 OF DECK LOOPER
```

Unexpected response received

WSim generates the following MTRC records when an unexpected response is received:

```
ITP447I MSG GEN ENTERED: STMT# 00004 OF DECK LOOPER
ITP430I IMMMEDIATE IF   (LOOPER  00004) NOT MET - ELSE ACTION TAKEN:
        CALL FROM 00005 OF LOOPER TO ERROR AT 00007 OF LOOPER
ITP452I RETURN FROM STMT# 00009 OF DECK LOOPER TO STMT# 00005 OF DECK LOOPER
ITP450I BRANCH FROM STMT# 00006 OF DECK LOOPER TO TOPLOOP AT 00002 OF DECK LOOPER
ITP448I MSG GEN ENDED:   STMT# 00004 OF DECK LOOPER
```

Note: To see an example of how to preserve the WAIT indicator over asynchronous IF statements, see “Preserving the WAIT indicator over asynchronous IF statements” on page 145.

Using logic tests to create self-checking scripts

How do you know that WSim runs your simulation correctly regardless of whether you simulate one terminal or thousands? Are the simulated terminals actually sending messages and getting the correct responses in return? If not, you may have a problem, either in the message generation decks or in the application itself. This section describes why you need self-checking scripts and how you can use logic tests to write self-checking scripts that verify whether a terminal session is proceeding as expected.

The concept of self-checking scripts applies to message generation decks written manually and those created by the script generating utilities. Test and debug your decks when you create them. However, this is not always enough to ensure that

WSim can handle unexpected situations when running a complex simulation. For example, the following list indicates problem situations that can occur during an important simulation run.

- Logon failed. This might be caused by new or expired passwords, too many users, applications unavailable, routes unavailable, and so on.
- A message from the operator is sent to the simulated VM panel. This causes the panel to enter the HOLDING state instead of MORE ..., and subsequent transactions would not be accepted.
- The system has changed since the last WSim run. This can be caused by a problem related to a data set, a different response message, or authorization modifications.
- A timing difference exists due to a heavily loaded host processor. Some decks work fine for many months. However, if the system does not respond fast enough in one instance, the terminal can send another message causing the session to be out of synchronization.

Although these are just a few simple, common examples of problems that can arise during simulation, other problems are also possible. You can create self-checking scripts to try to guard against these unexpected problems.

Not everyone needs to write self-checking scripts. However, consider the consequences that might arise if you do not use self-checking scripts. What happens if you do not notice a simulation problem in a timely manner? If you are running short and simple simulations, you can probably detect errors yourself easily. However, if you waste an 8-hour standal-one shift due to terminals getting out of synchronization, then self-checking scripts could have well been worth the effort of preparing them. Self-checking scripts can assist you in debugging these scripts by doing the following:

- Detecting errors sooner than otherwise would be possible
- Detecting errors that otherwise would not be detected.

Determining whether you need self-checking scripts

To help you determine whether you need self-checking scripts to verify your simulations, you should consider alternative ways of determining whether your simulations have processed correctly. The following list describes indicators you can use to track a simulation and provides considerations that may limit their effectiveness.

- The Log Compare Utility. This utility compares 3270 display (DSPY) records between two log data sets and provides reports when a difference is detected. Although the Log Compare Utility can help you determine whether an application has changed over a period, it requires data from two separate simulations.
- The Loglist Utility. This utility tells exactly how the simulation ran. Its primary drawback is that it is run after the simulation is over. In addition, some log lists can be much too long to read in complete detail. Therefore, problems can still go undetected.
- Display Monitor Facility. With this facility, you can see the display image or data stream of a terminal during a simulation run. This facility can be used effectively to verify simulations involving these terminal types, but might be impractical as a verification tool if you are simulating large numbers of terminals.

- Message Rates. WSim can display send and receive message rates during the simulation. If they are high and remain stable, that might mean that everything is fine. However, you cannot be sure that some terminals are not out of synchronization.
- Query Command Output Data. You can use the Q command to query a terminal online; however, it is unlikely that you will query your terminal at the exact instance that a problem occurs.
- Terminal "Hang" Situations. If you even notice a terminal hang situation in the first place, it can take much more effort to find the cause of the problem and correct it in a timely fashion.
- Operating System Messages. The operating system console can warn you about some problems, but it cannot tell you which types.
- Application Program Abnormal Ending. This is a sure sign of a problem, but more subtle problems can also exist without causing abnormal endings.

Creating self-checking scripts

Self-checking scripts verify whether the expected responses are received from the system under test. The action taken when WSim finds a problem depends on the cause of the problem, such as incorrectly written scripts, bugs existing in the application, or temporary system problems that require a particular sequence of terminal recovery actions.

After sending a message, self-checking scripts use IF statement logic tests to scan messages received by the terminal for the expected response. If the response is correct, the terminal proceeds further. If not, the discrepancy should be noted and fixed. This sequence is demonstrated in Figure 20.

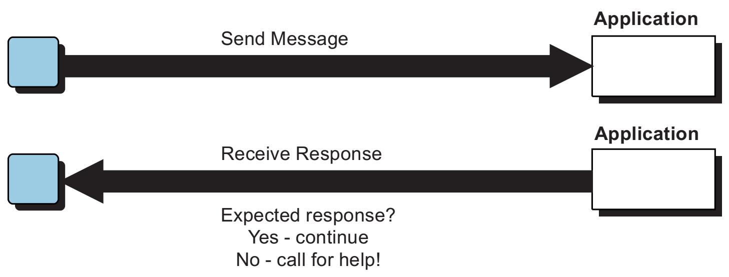


Figure 20. Self-checking script logic

You can code simple, message-level self-checking scripts, or more complex, network-level self-checking scripts to recover from errors automatically. As discussed in the following sections, both types of scripts can save time and find unexpected problems with your application systems.

Message-Level self-checking scripts

The following example shows a message generation deck that does not include self-checking logic tests:

```
DECK5  MSGTXT
*                         Beginning of DECK5.
MSG1   TEXT  (MESSAGE 1)   Defines MSG1.
MSG2   TEXT  (MESSAGE 2)   Defines MSG2.
       ENDTXT              End of DECK5.
```

The following example shows the same message generation deck with self-checking logic built-in to make the system wait for the expected response:

```
DECK5  MSGTXT
*                                Beginning of DECK5.
MSG1   TEXT  (MESSAGE 1)         Defines MSG1.
0      IF    LOC=B+0,
       TEXT=(REPLY 1),
       THEN=CONT
WAIT1  WAIT
MSG2   TEXT  (MESSAGE 2)         Defines MSG2.
0      IF    LOC=B+0,            Defines basic logic test.
             TEXT=(REPLY 2),
             THEN=CONT
WAIT2  WAIT
       ENDTXT                    End of DECK5.
```

This example demonstrates the most basic form of a self-checking script, using TEXT-IF-WAIT sequences. However, two drawbacks exist:

- The size of the message generation deck can grow by a factor of three.
- The terminal merely "hangs" if WSim does not receive the expected response. WSim does not take an error action to alert you or to correct the situation.

As it turns out, these drawbacks are not always serious. Therefore, this basic method helps you to write self-checking scripts.

Network-Level self-checking scripts

Although the message-level IF statement is a fundamental part of self-checking scripts, you can code network-level IF statements to enhance the self-checking qualities of your scripts further and to reduce the size of the TEXT-IF-WAIT blocks in your message generation decks, as shown in the following example

```
NET1  NTWRK
*                Sample WSim Script
*   This script illustrates using network IF statements to reduce the
*   size of message generation decks.
*
*                                    Beginning of NET1.
SEND  IF   LOC=D+0,TEXT=(ANYTHING),
           THEN=WAIT,ELSE=WAIT,
           WHEN=OUT
RECV  IF   LOC=D+0,TEXT=RESP,
           SCAN=YES,THEN=CONT,
           ELSE=CERROR,WHEN=IN
       .
       .                             Network definition statements.
       .
DECK1 MSGTXT
*                                    Beginning of DECK1.
MSG1  TEXT  (MESSAGE 1),             Defines MSG1.
            RESP=(REPLY 1)
MSG2  TEXT  (MESSAGE 2),             Defines MSG2.
            RESP=(REPLY 2)
       .
       .                             Message generation statements.
       .
      ENDTXT                         End of DECK1.

ERROR MSGTXT
*                                    Beginning of ERROR deck.
      WTO   (EXPECTED RESPONSE ),
            (NOT RECEIVED. ),
            (PLEASE HELP!)           Message to operator.
WAIT1 WAIT
      ENDTXT                         End of ERROR deck.
```

WSim evaluates network-level IF statements SEND and RECV for every message sent and received by all terminals in the network. The following steps describe the sequence of events in message generation for the preceding example:

Steps:

1. When WSim sends MSG1, IF statement SEND puts the terminal into a wait state. The RESP=(REPLY 1) operand on the TEXT statement indicates the expected response to this request.
2. The terminal waits until WSim receives a message.
3. IF statement RECV checks the message received and determines if the expected response REPLY 1 exists anywhere in the data stream. If so, THEN=CONT resets the WAIT indicator and enables the terminal to continue and send MSG2.
4. If the expected response did not exist in the message received, IF statement RECV causes WSim to take the ELSE=CERROR action. CERROR specifies that WSim calls (C) the message generation deck named ERROR. This message generation deck notifies the operator that WSim did not receive the expected response. In fact, you could have set up this message generation deck to perform various error recovery actions, relieving the operator of manual intervention.

When you code IF statements along with message generation statements such as TEXT, WAIT and WTO, you can create many types of self-checking scripts. The type of self-checking script you decide to code may depend on the considerations discussed in the following sections.

Positioning statements to check your scripts

You can enter coding for self-checking scripts at the following positions in a script:

- After you code each message to be generated
- After a complete function.

If you want to check your scripts after each message generated, you can code the TEXT-IF-WAIT block of statements. This type of self-checking script allows the most thorough types of checking to be done on a message-by-message basis.

To check your scripts after a complete function, you can place one IF statement logic test at the end of each message generation deck or logical function that WSim processes. In this manner, you can check your progress at larger intervals.

Checking for unexpected responses

If you want to check for unexpected responses from the system under test, you can add WAIT statements and WTO statements to your decks. You can also code IF statement logic tests that specify actions to be taken when an unexpected response is received:

- With the WAIT statement, you can simulate a terminal waiting for an expected message. However, you cannot specify an action to be taken when an unexpected response is received.
- When you code the WTO statement followed by a WAIT statement, you can notify the operator about unexpected responses. Then it is up to the operator to diagnose and correct the problem while the terminal waits.
- If you code the IF statement logic test, you can create message generation decks that automatically respond to unexpected responses, determine the nature of the error, and possibly correct the problem. The actions you can specify with the IF statement offer built-in corrective capabilities.

Chapter 17. Understanding control statements

As discussed in Chapter 12, “Basic concepts,” on page 107, the message generation statements provided with WSim are classified as delimiters, logic tests, and control statements. Chapter 14, “Understanding delimiters,” on page 137 describes how delimiters affect the message generation process; Chapter 16, “Defining logic tests,” on page 165 provides information about coding logic tests. This chapter discusses control statements, which are message generation statements that control the message generation process through specific actions and events.

Below lists the message generation statements classified as control statements:

BRANCH	DEACT	LABEL	OPCMND	SET
CALC	DELAY	LOG	RESET	SETSW
CALL	ERROR	MONITOR	RESP	TH
CANCEL	EVENT	MSGTXT	RETURN	WTO
DATASAVE	EXIT	ON	RH	WTOABRHD
CM*xxxx*[3]				

This chapter provides the following information about control statements and how they affect the message generation process:

- Coding control statements
- Using control statements for specific devices
- Monitoring and automating the message generation process
- Altering sequential processing
- Controlling switches and counters
- Controlling events.

As discussed in the following sections, with control statements, you can alter message generation processing and simulate the diverse actions of specific devices.

Coding control statements

This section illustrates how you code control statements in a message generation deck and helps you understand how control statements affect the message generation process. The descriptions and coding examples in this section show the range of actions that can be specified with control statements.

MSGTXT

The MSGTXT statement specifies the beginning of each message generation deck and the control character and text delimiter you use when coding the deck. As shown in the following example, the MSGTXT statement also enables you to control the message generation process when you code the COUNT operand:

```
DECK1   MSGTXT  COUNT=30     WSim processes DECK1 30 times.
```

3. CMxxxx is a generalized notation used to indicate a category of statements that are used during CPI-C transaction program simulations. All CPI-C simulation statements are control statements. Refer to *WSim Script Guide and Reference* for a list and description of all CPI-C simulation statements.

With the COUNT operand, you can control how many times WSim processes a deck in succession before going to the next deck specified on the PATH statement. In the preceding example, WSim processes DECK1 a total of 30 times.

Note: There are other MSGTXT operands for specific devices. For detailed information about these operands, see the *WSim Script Guide and Reference*.

DATASAVE

With the DATASAVE statement, you can save data specified manually or data in a buffer by placing it in a save or user area. With the TEXT operand of the DATASAVE statement, you can save data specified manually in hexadecimal or EBCDIC format or data generated with data field options.

The following example illustrates two DATASAVE statements:

```
SAVE1  DATASAVE  AREA=3,LOC=B+80,   Specifies the data to be saved and
                 LENG=100           the area it is to be saved in.
SAVE2  DATASAVE  AREA=U+0,          Uses a data field option to generate
                 TEXT=($ID,8$)      data to be saved.
```

In the preceding example, SAVE1 saves 100 bytes of data at an offset of 80 into the buffer and places it in device save area number 3. SAVE2 generates the 8-character name of the terminal associated with this deck and places the name in the device user area.

If you code the TEXT operand on the DATASAVE statement without specifying any data, WSim performs a clear function on the area defined by the AREA operand. For dynamic save areas, the storage associated with the save area is released for other uses.

```
SAVEA  DATASAVE  AREA=5,TEXT=()   Specifies that WSim perform a clear
*                                 function on device save area 5.
```

For more information about the clear function, see the *WSim Script Guide and Reference*.

You can retrieve data saved in a save or user area with the $RECALL$ data field option, as discussed in “Inserting data into a message” on page 134.

CALC

With The CALC statement, you can control the value maintained at a specific location in a save or a user area. With this statement, you can change the value in one of the following ways:

- Set a new value
- Add to an existing value
- Subtract from an existing value.

To use the CALC statement, you must first save numeric data with the DATASAVE statement, as discussed in “DATASAVE.” Then code the CALC statement to perform a calculation on the data:

```
ADD1  CALC  LOC=N+5,VALUE=+500   Addition calculation.
```

In the preceding example, LOC=N+5 specifies an offset into the network user area where WSim performs the calculation. VALUE=+500 specifies that WSim add 500 to the value at the location specified by the LOC operand.

To retrieve the new data resulting from this calculation, use the $RECALL$ data field option as discussed in “Inserting data into a message” on page 134.

Note: The CALC statement operates only on data in hexadecimal EBCDIC format. For example, use the hexadecimal EBCDIC characters X'F1F2F3' to represent the decimal number 123 in the save or user area before using the CALC statement to add to or subtract from the value.

DELAY

The DELAY statement specifies a delay value that WSim uses to calculate a temporary intermessage delay. The value coded on the DELAY statement overrides the delay value specified by the DELAY operand on statements such as the TP, DEV, and LU network definition statements. However, the new delay value remains active only for the message currently being generated. In this manner, you can alter the intermessage delay for individual messages.

The following example shows how to code the DELAY statement:

```
DELAY1  DELAY  TIME=100  Specifies a new delay value for the current
*                        message.
```

When WSim processes DELAY1, it calculates a new intermessage delay by multiplying the active user time interval (UTI) for this device by the new delay value (100).

For more information about intermessage delays and user time intervals, see Chapter 15, “Understanding intermessage delays,” on page 153. For a complete list of the network definition statements on which you can code the DELAY operand, see the *WSim Script Guide and Reference*.

EXIT

The EXIT statement invokes a user exit routine during message generation, enabling the exit routine to control message generation with return codes. Code this statement as shown in the following example:

```
EXIT6  EXIT  MODULE=name  Specifies the exit module that gains control
*                         when WSim processes the statement.
```

For more information about user exit routines, see *WSim User Exits*. For more information about the EXIT statement, see *WSim Script Guide and Reference*.

CMxxxx

CM*xxxx* statements are CPI-C verb statements that areused during CPI-C transaction program simulations. Some CPI-C simulation statements are also delimiters. See “How delimiters affect the message generation process” on page 137 for a list of the CPI-C delimiters. Refer to *WSim Script Guide and Reference* for a list and description of all CPI-C simulation statements.

Using control statements for specific types of devices

When you simulate specific types of devices, you can control the message generation process with specific control statements. The following sections describe these statements and provide coding examples.

RESET

With RESET statement, you can simulate the action of a RESET key on a display device. Code this statement as shown in the following example:

```
RESET5   RESET             Simulates action of the RESET key.
```

WSim recognizes the RESET statement only for 3270 simulations.

ERROR

The ERROR statement performs the following functions during message generation:

- Provides logical error simulation for 3270 devices
- Generates the SNA sense bytes for an LU2 device.

The ERROR statement specifies the status and sense values to be entered. As shown in the following example, you must code the value for the STATUS operand as 4 hexadecimal digits enclosed within single quotation marks:

```
ERROR1  ERROR   STATUS='xxxx'  Specifies the status and sense values to
*                              be entered.
```

WSim recognizes the ERROR statement only for 3270 simulations.

RESP

The RESP statement provides for logical error simulation by SNA devices. It overrides the normal SNA response with an exception response that contains a sense value you specify with the SENSE operand.

As shown in the following example, you must code the value for the SENSE operand as 8 hexadecimal digits enclosed within single quotation marks:

```
RESP8   RESP  SENSE='xxxxxxxx'   Specifies sense value to be included with
*                                the exception response.
```

TH

During SNA simulations, you can use the TH statement to perform the following functions:

- Modify the SNA transmission header (TH) built by WSim for messages generated with a TEXT statement
- Build a transmission header for a command defined by a CMND statement.

The following example illustrates coding for the TH statement:

```
TH2  TH  SNF=55  Sets the transmission header sequence number field.
```

WSim recognizes the TH statement only during SNA simulations.

RH

During SNA simulations, you can use the RH statement to perform the following functions:

- Modify the SNA request or response header (RH) built by WSim for messages generated with a TEXT statement
- Build a response header for a command defined by a CMND statement
- Specify chaining of transmitted messages.

The following example illustrates coding for the RH statement:

```
RH5  RH   CHAIN=FIRST  Specifies that chaining control flags be set in
*                      the RH.
```

WSim recognizes the RH statement only during SNA simulations.

Monitoring and automating message generation

In addition to the control statements for specific types of devices, WSim provides control statements that monitor and automate the message generation process. The following sections describe these control statements and provide coding examples.

OPCMND

With OPCMND statement, you can automate message generation by specifying an operator command that WSim issues during the simulation. When WSim processes the OPCMND statement, it queues the operator command for execution. See *WSim User's Guide* for valid operator commands.

As shown in the following example, the data you enter must be in EBCDIC format, and it must represent a valid operator command:

```
OPC1  OPCMND  (A $NETID$,U=100)       Changes the network UTI to 100.
OPC2  OPCMND  (A $NETID$,),           Alters the message generation
              (PATH=((0,1,2,3,4,5),   path for all network devices.
                   (,6,7,8,9,10)))
OPC3  OPCMND  (ZEND)                  Shutdown WSim.
```

When WSim processes OPC1 and OPC2 during a simulation, it issues the A (Alter) operator command. OPC1 changes the current network UTI value to 100, and OPC2 alters the message generation path for all devices in the network.

The $NETID$ data field option coded in the preceding example dynamically inserts the name field from the current NTWRK statement into the data. See the *WSim Script Guide and Reference* for complete information about $NETID$ and other available data field options.

When WSim processes OPC3 during a simulation, it issues the ZEND command to stop WSim processing. The ZEND command writes the message log buffers to the log data set and terminates WSim. See *WSim User's Guide* for more information on the ZEND command.

MONITOR

The MONITOR statement causes the Display Monitor Facility to display the simulated 3270 display image on the display monitoring device:

```
MNTR23  MONITOR             Displays the 3270 display image.
```

WSim recognizes this statement only during 3270 simulations.

For more information about the Display Monitor Facility, see *WSim Utilities Guide*.

WTO and WTOABRHD

With the WTO and WTOABRHD statements, you can write informational messages about the simulation to the operator console during message generation. The WTO and WTOABRHD statement data fields contain data in the same format as the TEXT statement data field; you can enter data in hexadecimal format, text strings, or data generated with data field options. WTOABRHD uses only an abbreviated header before the data.

The following example shows how to write a message to the operator console:

```
WTO1  WTO  ($ID,8$ BEGINNING ),   Message to the operator console.
           (EXECUTION $TOD,8$)
```

When WSim processes WTO1, it sends the name of the active terminal, the text "BEGINNING EXECUTION", and the current time of day to the operator console.

LOG

With LOG statement, you can write data to the log data set. In addition, you can monitor message generation by writing messages for a terminal to the log data set during message generation. WSim marks these messages as LOG records in the log header.

You can log varied information depending on the LOG statement operands you code:

- The LOG statement data field specifies data entered in hexadecimal format, text strings, or data generated with data field options for logging.
- The AREA operand enables you to log the contents of save or user areas.
- The DISPLAY operand enables WSim to write the display buffers to the log data set for formatting by the Loglist Utility.

The following example shows how to code two different LOG statements:

```
LOG1  LOG  ($ID,8$ BEGINNING ),    Logs terminal name, text BEGINNING
           (EXECUTION $TOD,8$ )    EXECUTION, and time of day.
LOG2  LOG  DISPLAY                 Logs the device display image.
```

LOG1 sends the name of the active terminal, the text "BEGINNING EXECUTION", and the current time of day to the log data set. LOG2 writes the contents of the simulated display image to the log data set.

For more information about formatting the log data set and controlling message logging, see *WSim Utilities Guide*.

Altering sequential processing

WSim usually processes each message generation deck as a complete unit, processing each statement sequentially before selecting the next deck for processing. Throughout the message generation process, WSim maintains a statement pointer for each terminal, which indicates the current position being processed in the deck. Each time a terminal enters message generation, the message generation routine uses the saved statement pointer to determine where to resume processing within the deck. When you modify the statement pointer, you alter the sequential processing of message generation statements.

You can modify the position of the pointer by coding the BRANCH and CALL statements and similar actions on IF statements: B (branch) and C (call). You can code the LABEL and RETURN control statements along with CALL and BRANCH. The LABEL statement establishes a label in a deck that WSim uses as the location to go to during a branch or call operation; the RETURN statement returns processing to the point where the last call was issued.

The following sections describe the BRANCH, CALL, LABEL, and RETURN statements, including a coding example for each statement. For more information about the IF statement B (branch) and C (call) actions, see "Coding the THEN and ELSE operands" on page 173.

BRANCH

The BRANCH statement unconditionally changes the sequence of message generation by specifying either the name of a message generation deck or the name of a deck and a statement label that identifies the next statement to be executed. If you want to branch to another statement in your present deck, you can enter a statement label without specifying the name of the deck.

The following example illustrates the BRANCH statement:

```
BNCH5   BRANCH   NAME=DECK8    WSim branches to and starts processing
*                              DECK8.
BNCH6   BRANCH   NAME=DECK8,   WSim branches to and starts processing
                 LABEL=MSG1    DECK8 beginning with statement MSG1.
BNCH7   BRANCH   LABEL=MSG1    WSim branches to statement MSG1 in the
*                              current deck and continues message
*                              generation.
```

Notes:

- The BRANCH statement does not provide a way for WSim to return to the statement where the branch took place.
- You cannot branch to IF statements that specify WHEN=IN or OUT.

CALL

The CALL statement alters message generation in the same way as the BRANCH statement does; however, it causes WSim to return to the statement where the call took place when WSim processes a RETURN or an ENDTXT statement.

The following example illustrates the CALL statement:

```
CALL5  CALL  NAME=DECK4   WSim begins processing DECK4.
CALL6  CALL  NAME=DECK4,  WSim begins processing DECK4 at statement MSG8.
             LABEL=MSG8
CALL7  CALL  LABEL=MSG8   WSim begins processing the current deck at
*                         statement MSG8.
```

Message generation decks that WSim enters with a branch or call operation are extensions of the message generation deck from which you issued the branch or call. This means that WSim does not use the COUNT operand on the new message generation deck, and any logic tests activated in the calling deck remain active.

For more information about the COUNT operand on the MSGTXT statement, see "MSGTXT" on page 195.

Note: You cannot call IF statements that specify WHEN=IN or OUT.

LABEL

As shown in the following example, you use the LABEL statement to establish a label that WSim uses as the object of a branch or call operation:

```
name  LABEL                Specifies a name for WSim to use  during branch
*                          and call operations.
```

Note: You can also branch or call to the labels you code in a message generation statement's name field, except for IF statements that specify WHEN=IN or WHEN=OUT. You cannot branch or call to IF statements coded with these operands.

RETURN

When you code the RETURN statement, WSim restarts message generation at the point where the last CALL statement appeared or the last C (call) action on a logic test was issued.

The following example illustrates the RETURN statement:

```
RETURN1  RETURN            WSim returns to the point of the last CALL
*                          statement or action.
```

Notes:

- WSim ignores the RETURN statement if no CALL statement or action is active.
- The ENDTXT statement acts as a RETURN statement if a CALL statement is active when it is processed by WSim.

Setting switches and counters

WSim maintains counters and switches for each network, terminal, and device. The following sections describe setting counters and switches with the SETSW and SET control statements. For more information about switches and counters, see "Sequence and index counters" on page 125 and the *WSim Script Guide and Reference*.

SETSW

With the SETSW statement, you can set or clear any or all of the switches you defined for a network or a terminal. WSim provides 4095 switches at the network level that any terminal in the network can access. In addition, each terminal capable of message generation has 4095 switches that can be used by that terminal only. You use these switches to indicate certain conditions and then test the switches with IF statement logic tests.

The following example shows how to set the fourth device switch-on for a terminal:

```
SETSW1  SETSW  SW4=ON    Sets the 4th device switch on.
```

Note: The only values that you can code for a switch are ON and OFF.

SET

You can alter the values of sequence and index counters with the SET message generation statement. With SET statement, you can set a counter to the following values:

- A specific integer, a random number, or the value of another counter
- A value resulting from multiplying or dividing the values of two counters or the value of a counter and a constant
- The remainder that results from dividing the values of two counters or the value of a counter and a constant
- Hexadecimal data saved in a user or save area
- The length of data in a user or save area
- The numeric value of EBCDIC data saved in a user or save area
- The index of the last UTBL item
- The cursor's row, column, or offset
- The number of rows or columns on a display screen.

You can also use counters to position the cursor with the CURSOR and SELECT message generation statements. For more information about these statements, see Chapter 18, "Generating messages for specific types of devices," on page 219. In addition, you can set a counter to the offset of the text that satisfies the logic test when you code a logic test with the SCANCNTR operand. For more information about this operand, see "Coding the SCANCNTR operand" on page 179.

To code the SET statement, enter the name of the counter and an option that specifies how WSim sets the counter's value:

```
SETCNTR   SET   cntr=option  Coding required for the SET statement.
```

You can code the name of a valid counter for *cntr*. Because a simulated VTAM application and its logical units do not have lines or terminals associated with them, WSim provides counters for those resources differently. The following list presents the names you can code for *cntr* and the counters that WSim provides for VTAMAPPLs and LUs:

NSEQ	Network sequence counter.
LSEQ	Line sequence counter. WSim provides one line sequence counter for each VTAMAPPL.
TSEQ	Terminal sequence counter. WSim provides one terminal sequence counter for each VTAMAPPL.
DSEQ	Device sequence counter. WSim provides one device sequence counter for each VTAMAPPL LU half-session.
NC*n*	Network index counter *n*.
LC*n*	Line index counter *n*. WSim provides 3 - 4095 line index counters for each VTAMAPPL.
TC*n*	Terminal index counter *n*. WSim provides 3 - 4095 terminal index counters for each VTAMAPPL.
DC*n*	Device index counter *n*. WSim provides 3 - 4095 device index counters for each VTAMAPPL LU half-session.

Table 10 illustrates which counters WSim allocates for each resource simulated by WSim:

Table 10. How counters are allocated for resources

Simulation Type	LSEQ LC1 - LC*n*	TSEQ TC1 - TC*n*	DSEQ DC1 - DC*n*
CPI-C TP			
APPCLU	X		
TP		X Note 1	X Note 2
VTAM Application			
VTAMAPPL	X	X	
LU			X
TCP/IP Connection			
TCPIP	X	X	
DEV			X

Notes:

1. This set of counters is available to all instances of a given CPI-C transaction program.
2. This set of counters is unique for each CPI-C transaction program instance.

You can code several different values for *option*, some of which are described in following sections. For a complete list of the values you can code, see the *WSim Script Guide and Reference*.

As shown in the following example, you must code at least one operand on the SET statement:

```
SET1  SET  TSEQ=+5  Adds 5 to the value of the terminal sequence counter.
```

With SET statement, you can alter multiple counters in a single statement. As shown in the following example, you can code one SET statement to increment the device sequence counter by 10, set the line sequence counter to the same value as network counter 5, and set device index counter 2 to a random number 10 - 100:

```
SETCNTR  SET  DSEQ=+10,LSEQ=NC5,DC2=(10,100)  Sets counters.
```

The following sections describe how to set the value of a counter with the SET statement.

Setting a counter to a specific value or random number

With the following options, you can set a counter to specific values or random numbers:

integer Sets the counter to the specified integer. *integer* is an integer from 0 to 2147483647.

```
SET1  SET  NC2=3           Sets network index counter 2 to 3.
```

cntr Sets the counter to the specified counter.

```
SET1  SET  DC3=NSEQ        Sets device index counter 3 to the value in
*                          the network sequence counter.
```

(*lo*,*hi***)** Sets the counter to a random number between *lo* and *hi*, where *lo* and *hi* are integers between 0 and 2147483647 or counter specifications whose values are within this range. *lo* must be less than *hi*.

```
SET1  SET  LC1=(1,100)     Sets line index counter 1 to a random number
*                          from 1 to 100.
```

RN*n* Sets the counter to a random number in the range specified by the RN statement with label *n*, where *n* is an integer between 0 and 255.

```
1     RN   LOW=1,HIGH=50   Random number between 1 and 50.
⋮
SET1  SET  TC5=RN1         Sets terminal index counter 5 to a random
*                          number specified by RN statement 1.
```

Setting a counter with arithmetic operations

With the following options, you can set a counter to the result of a multiplication, addition, subtraction, or division procedure:

**integer* Sets the counter to the product of its own value and the value specified by *integer*. *integer* is an integer from 1 to 2147483647. WSim wraps the value of the counter after reaching 2147483647.

```
SET1  SET  TC1=2,  Sets terminal index counter 1 to 2.
           TC1=*5  WSim multiplies the value of terminal index
*                  counter 1 (2) by 5 and sets the value of
*                  terminal index counter 1 to the product (10).
```

+integer Sets the counter to the sum of its own value and the value specified by *integer*. *integer* is an integer from 1 to 2147483647. WSim wraps the value of the counter after reaching 2147483647.

```
SET1  SET  LC1=8   Sets line index counter 1 to 8.
SET2  SET  LC1=+5  WSim adds 5 to the value of terminal index
*                  counter 1 (8) and sets the value of
*                  terminal index counter 1 to the sum (13).
```

-integer Sets the counter to the difference of its own value and the value specified by *integer*. *integer* is an integer from 1 to 2147483647. WSim wraps the value of the counter to 2147483647 after reaching 0.

```
SET1  SET  DC3=5,  Sets device index counter 3 to 5.
           DC3=-1  WSim subtracts 1 from the value of device
*                  counter 3 (5) and sets the value of
*                  device index counter 3 to the difference (4).
```

/integer Divides the value of the counter by *integer* and sets the counter to the quotient. *integer* is an integer from 1 to 2147483647.

```
SET1  SET  NSEQ=11  Sets the network sequence counter to 11.
SET2  SET  NSEQ=/2  WSim divides the value of the network
*                   sequence counter (11) by 2 and sets the
*                   network sequence counter to the quotient (5).
```

//integer Divides the value of the counter by *integer* and sets the counter's value to the remainder. *integer* is an integer from 1 to 2147483647.

```
SET1  SET  TC8=33   Sets the terminal index counter 8 to 33.
SET2  SET  TC8=//9  WSim divides the value of terminal index
*                   counter 8 (33) by 9 and sets the value of
*                   terminal index counter 8 to the
*                   remainder (6).
```

**scntr* Sets the counter to the product of its own value and the value of *scntr*. WSim wraps the value of the counter after reaching 2147483647.

```
SET1  SET  TC1=4,     Sets terminal index counter 1 to 4.
           NSEQ=10    Sets the network sequence counter to 10.
SET2  SET  TC1=*NSEQ  WSim multiplies the value of terminal index
*                     counter 1 (4) by the value of the network
*                     sequence counter (10) and sets the value of
*                     terminal index counter 1 to the product (40).
```

+scntr Sets the counter to the sum of its own value and the value of *scntr*. WSim wraps the value of the counter after reaching 2147483647.

```
SET1  SET  LC1=2,     Sets line index counter 1 to 2.
           NSEQ=15    Sets the network sequence counter to 15.
SET2  SET  LC1=+NSEQ  WSim adds the value of line index
*                     counter 1 (2) to the value of the network
*                     sequence counter (15) and sets the value of
*                     line index counter 1 to the sum (17).
```

-scntr Sets the counter to the difference between its own value and the value of *scntr*. WSim wraps the value of the counter to 2147483647 after reaching 0.

```
SET1  SET  DC1=20,    Sets device index counter 1 to 20.
           DSEQ=9     Sets the network sequence counter to 9.
SET2  SET  DC1=-DSEQ  WSim subtracts the value of device sequence
*                     counter (9) from the value of the device
*                     index counter (20) and sets the value of
*                     device index counter 1 to the difference (11).
```

/scntr Divides the value of the counter by the value of *scntr* and sets the counter's value to the quotient. If you attempt to divide by 0, the counter's value does not change, and WSim logs message ITP4681 indicating the reason for the failure to set the counter's value.

```
SET1  SET  NSEQ=13,    Sets the network sequence counter to 13.
           LSEQ=3      Sets the line sequence counter to 3.
SET2  SET  NSEQ=/LSEQ  WSim divides value of the network sequence
*                      counter (13) by the value of LSEQ (3) and
*                      sets the network sequence counter to the
*                      quotient (4).
```

//scntr Divides the value of the counter by the value of *scntr* and sets the counter's value to the remainder. If you attempt to divide by 0, the counter's value does not change, and WSim logs message ITP4681 indicating the reason for the failure to set the counter's value.

```
SET1  SET  TC2=25,    Sets terminal index counter 2 to 25.
           NC9=7      Sets network index counter 9 to 7.
SET2  SET  TC2=//NC9  WSim divides the value of terminal index
*                     counter 2 (25) by the value of network index
*                     counter 9 (7) and sets the value of terminal
*                     index counter 2 to the remainder (4).
```

Setting a counter to hexadecimal data

To set the value of a counter to hexadecimal data in a save or user area, you can code CNTR=(X,*loc*,*leng*) on the SET statement. With this operand, WSim places 1 - 4 bytes of data into the counter without translating the data.

You can code the following values on this operand:

X Specifies that the data in the save or user area is hexadecimal data.

loc Specifies the location of the data, including the save or user area and the offset, if any. If the specified location does not exist, WSim logs message ITP468I and does not set the counter's value.

You code one of the following values for *loc*:

B±*value*
Specifies that the data is at an offset from the start of the data in the device buffer, excluding headers, (+*value*) or back from the end of the data in the buffer (-*value*) for nondisplay devices. For display devices, B-*value* specifies that the data is located at an offset back from the end of the data in the screen image buffer. *value* is an integer 0 - 32766 or the name of a valid counter.

C±*value*
Specifies that the data is at an offset from the current cursor position (+*value*) or back from the current cursor position (-*value*). Code this value only for display devices. *value* is an integer 0 - 32766 or the name of a valid counter.

N±*value*
Specifies that the data is at an offset from the start (+*value*) or the back from the end (-*value*) of the network user area. *value* is an integer 0 - 32766 or the name of a valid counter.

N*s***+***value*
Specifies that the data is located at offset *value* from the start of network save area *s*. *s* is an integer 1 - 4095. *value* is an integer from 0 to 32766 or the name of a valid counter.

(*row,col***)**
: Specifies the row and column location of the data. *row* and *col* may be integers 1 - 255 or the names of valid counters.

*s***+***value*
: Specifies that the data is at offset *value* from the start of save area *s*. *s* is an integer 1 - 4095. *value* is an integer from 0 to 32766 or the name of a valid counter.

U±*value*
: Specifies that the data is at an offset from the start (+*value*) or back from the end (-*value*) of the device user area. *value* is an integer 0 - 32766 or the name of a valid counter.

Note: If you specify a counter as the offset *value*, WSim evaluates that counter when it processes the SET statement and uses the counter's value as the offset. If the value is greater than 31999 for save and user areas or greater than 32766 for all other locations, WSim logs message ITP468 indicating that the value is invalid as an offset and that the counter's value was not set.

leng
: Specifies the length of the data. *leng* is 1 - 4, indicating that the data is 1 - 4 bytes in length. If you specify 2 or more and fewer bytes of data exists at the specified location, WSim sets the value to only the existing bytes of data.

In the following example, the network user area has a 1-byte field representing the length of the data, followed by a device name. This can be saved with the following hexadecimal data.

```
DATASAVE AREA=N+0,TEXT=('08C4C5E5C9C3C5F0F1')
```

You can set a counter's value equal to the first byte by coding the SET statement as shown in the following example:

```
SET5  SET  DC1=(X,N+0,1)  WSim sets the value of device counter 1 to the
*                         hexadecimal data located in the network user
*                         area at a 0 offset and 1 byte in length.
```

Setting a counter to the length of user or save area data

To set the value of a counter to the length of data in a user or save area, you can code the CNTR=LENG(N]U]N*s*]*s*) operand on the SET statement.

You can code one of the following values for the LENG operand:

N
: Sets the counter to the current length of the network user area.

U
: Sets the counter to the current length of the device user area.

N*s*
: Sets the counter to the current length of the data in network save area *s*, where *s* can be an integer 1 - 4095.

s
: Sets the counter to the current length of the data in save area *s*, where *s* can be an integer 1 - 4095.

Setting a counter to EBCDIC data

To set the value of a counter to EBCDIC data, you can code CNTR=(E,*loc*,*leng*) on the SET statement. When you code this operand, WSim sets the counter to the numeric value of a 1- to 10-byte EBCDIC number in a user or save area.

You can code the following values on this operand:

E Specifies that the data located in the save or user area is EBCDIC data.

loc Specifies the location of the data, including the save or user area and the offset, if any. If the specified location does not exist, WSim logs message ITP468I and does not set the counter's value. For a list of the values you can code for *loc*, see “Setting a counter to hexadecimal data” on page 206.

leng Specifies the length of the numeric field. WSim treats leading, nonnumeric characters such as blanks or alphabetic characters as zeros. WSim truncates trailing, nonnumeric characters. If WSim does not find numeric characters within the specified text or if the numeric field's value is greater than 2147483647, it logs message ITP468I and does not change the counter's value. If *leng* is longer than the data at the specified location, WSim uses the available data.

Table 11 illustrates the relationship between *leng*, the EBCDIC data, and the value WSim assigns to the counter:

Table 11. How EBCDIC data is assigned to a counter

leng	**EBCDIC Data**	**Counter Value**	**Explanation**
5	“12345”	12345	All numeric data transferred
5	“0345”	345	All numeric data transferred
4	“ 234 ”	234	Leading blank treated as a zero
5	“$2.45”	2	Translation ends at first nonnumeric character
1	“$234”	None	No numeric data within limits of *leng*
10	“2147483700”	None	Numeric data too large
2	“234”	23	Numeric data longer than *leng*
5	“123”	123	Numeric data shorter than *leng*

Setting a counter to the cursor's row, column, or offset

When you simulate display devices, with the following options, you can set the value of a counter to the row number, column number, or offset of the cursor:

CROW Sets the counter to the current row number of the cursor. The value returned by CROW is the actual row number on the panel, regardless of whether you are simulating a display device with multiple partitions.

CCOL Sets the counter to the current column number of the cursor. The value returned by CCOL is the actual column number on the panel, regardless of whether you are simulating a display device with multiple partitions.

COFF Sets the counter to the current offset of the cursor. When you simulate a 3290 or 8775 display device with multiple partitions defined, the offset returned by COFF represents the cursor offset from the beginning of the presentation space of the currently active partition.

Note: These options are valid only for display devices. If you specify them for nondisplay devices, WSim does not alter the value of the counter. In addition, WSim logs message ITP468I indicating that the reason for the failure was a cursor reference for a nondisplay device.

The following statements set three counters to a current cursor position of row 17, column 46:

```
SET1  SET  DC1=CROW   Sets device counter 1 to the current row
*                     position (17).
SET2  SET  NSEQ=CCOL  Sets the network sequence counter to the current
*                     column position (46).
SET3  SET  LC9=COFF   Sets line index counter 9 to the current offset
*                     (1325), assuming the display is 80 columns wide.
```

Setting a counter to the number of rows or columns in a display

You can set the value of a counter to the number of rows or columns in a display screen by coding the NUMROWS or NUMCOLS operand on the SET statement. Coding NUMROWS sets the counter to the number of rows on the simulated display device; coding NUMCOLS sets the counter to the number of columns.

Note: These options are valid only for display devices. If you specify them for nondisplay devices, WSim does not alter the value of the counter. In addition, WSim logs message ITP468I indicating that the reason for the failure was a cursor reference for a nondisplay device.

Setting a counter to the index of the last item in a user table

You can also set the value of a counter to the index of the last item in a user table with the UTBLMAX(*utblname*) operand on the SET statement. *utblname* specifies the name of an MSGUTBL or the label of a UTBL network definition statement.

Controlling events

As discussed in "The WAIT statement" on page 140, the EVENT=*event* operand on the WAIT statement names an event that WSim must post before further messages can be generated. To create and control the events that synchronize communication between simulated resources, you can code the following control statements:

- EVENT
- CANCEL
- ON
- DEACT
- DATASAVE.

In addition, with the WAIT/POST facility and the ON/SIGNAL facility, you can simulate two or more terminals interacting in a single transaction or a master terminal controlling a network. Both facilities use events to control and synchronize the communication between terminals.

The following sections describe each statement and facility; "Using events to synchronize multiple devices" on page 216 provides examples of using events to create complex simulations.

EVENT

The EVENT statement performs a post, reset, or signal action for the named event, depending on the operand you code. In addition, you can specify a *tag*, which is a name that identifies one or more EVENT statements that may be referenced by a CANCEL statement.

POST=*event*
: Specifies the name of an event to be posted.

QSIGNAL=*event*
: Specifies the name of an event to be signaled. This operand signals the named event only for the resource that issued the QSIGNAL.

RESET=*event*
: Specifies the name of an event to be reset to not posted.

SIGNAL=*event*
: Specifies the name of an event to be signaled.

EVENTTAG=*tag*
: Specifies a name that can be referenced by a CANCEL statement to cancel the action specified by the POST, QSIGNAL, RESET, or SIGNAL operands. For more information about the EVENTTAG operand and canceling an event, see "CANCEL" on page 211.

TIME=*value*
: Specifies the number of seconds WSim delays the action specified by this EVENT statement. *value* is an integer 1 - 21474836 or the name of a valid counter.

event is the name of an event that you specify manually or dynamically with data you retrieve from a save or user area. For more information about generating event names with user and save areas, see "Coding variable event names with the DATASAVE statement" on page 212. When you code EVENT=*event* on a WAIT statement, WSim inhibits message generation until the named event is posted. To post an event, code the POST operand on the EVENT statement, as shown in the following example:

```
EVENT7  EVENT  POST=READY  Specifies that WSim post event READY.
```

When you code the RESET operand on the EVENT statement, you can specify the name of an event that is to be reset:

```
EVENT8  EVENT  RESET=READY,   Specifies that WSim reset event READY to
*              TIME=15        not posted following a 15-second delay.
```

Note: The TIME operand specifies a number of seconds WSim delays before taking the action specified on the EVENT statement. If you do not code the TIME operand, the specified action occurs immediately.

When WSim signals an event, it satisfies all ON conditions for the named event throughout the network. Thus, one signal operation can affect many ON statements and many different terminals.

The following example shows how to code the SIGNAL operand:

```
EVENT5  EVENT  SIGNAL=READY    Signals event READY for all network ON
*                              statements that specify event READY.
```

The QSIGNAL, or qualified signal, operand on the EVENT statement performs the same action as the SIGNAL operand. With the QSIGNAL operand, however, WSim signals the event only for the terminal that issued the QSIGNAL. No other terminals in the network is affected. When you code the QSIGNAL operand, you can use identical event names for all devices in a network. In this way, all network devices can use one common message generation deck without unique event names.

The following example shows how to code a qualified signal action using the QSIGNAL operand on the EVENT statement:

```
EVENT1  EVENT  QSIGNAL=READY   Signals event READY for the device issuing
*                              the EVENT statement.
```

Note: You can use the SIGNAL operand of the A (Alter) operator command to signal an event in a network. You cannot use the A (Alter) operator command to issue a qualified signal, since the QSIGNAL operand must be associated with a device and not a network.

CANCEL

When you code the EVENTTAG operand on both the CANCEL and EVENT statements, you can control events by canceling several event actions with one statement. The EVENTTAG=*tag* operand on the CANCEL statement enables you to cancel POST, SIGNAL, RESET, and QSIGNAL actions on all EVENT statements named *tag*. However, the EVENT statements must be coded with TIME=*value*, and the specified amount of time cannot have elapsed.

In the following example, EVENT1 specifies that WSim is to signal event SNOWFALL after a 5-second delay. In addition, the EVENTTAG operand specifies the tag STORM. CANCEL1 cancels the event actions on any EVENT statement tagged STORM, if the specified delay has not elapsed.

```
EVENT1   EVENT   SIGNAL=SNOWFALL,      Signal event SNOWFALL following
                 TIME=5,EVENTTAG=STORM  a 5-second delay.
         .
         .                             Message generation statements.
         .
CANCEL1  CANCEL  EVENTTAG=STORM         Cancels event SNOWFALL if the
*                                      5-second delay has not expired.
```

If the 5-second delay has not elapsed when WSim processes CANCEL1, it cancels the signal action specified by EVENT1.

The following example illustrates how the CANCEL statement selectively cancels EVENT statements depending on their tag and whether the TIME operand has been coded:

```
EVENT1   EVENT   POST=RAIN,TIME=15,     Posts event RAIN following a
                 EVENTTAG=WEATHER       15-second delay.
EVENT2   EVENT   RESET=SNOW,TIME=10,    Resets event SNOW following a
                 EVENTTAG=WEATHER       10-second delay.
EVENT3   EVENT   SIGNAL=SLEET,TIME=25,  Signals event SLEET following
                 EVENTTAG=STORM         a 25-second delay.
EVENT4   EVENT   QSIGNAL=HAIL           Signals event HAIL immediately
*                                       for the active device only.
         .
         .                              Message generation statements.
         .
CANCEL5  CANCEL  EVENTTAG=WEATHER       Cancels all EVENT statements
*                                       with the TAG WEATHER and coded
*                                       with TIME=value.
```

In the preceding example, CANCEL5 cancels the event actions specified by EVENT1 and EVENT2, if the specified delays have not elapsed. CANCEL5 does not cancel the action specified by EVENT3 because it is tagged STORM instead of WEATHER. In addition, CANCEL5 does not cancel the action specified by EVENT4 because the TIME and the EVENTTAG operands were not specified.

For more information about coding the EVENTTAG operand on the event statement, see “EVENT” on page 209.

ON

With the ON statement, you specify the name of an event and an action to be taken when the named event is signaled by an EVENT statement. The ON statement's EVENT operand specifies the name of the event; the THEN operand specifies the action to be taken.

Note: You must code both the EVENT and THEN operands when you code the ON statement.

The actions specified on the THEN operand are identical to actions you can specify on the IF statement THEN and ELSE operands. For more information about these actions, see "Coding the THEN and ELSE operands" on page 173.

The following example illustrates how you code the ON statement:

```
ON3  ON  EVENT=ERROR,THEN=BDECK5    Specifies an event and an action.
```

ON3 specifies that when event ERROR is signaled, any device that processes this ON statement will branch to and begin processing DECK5. If a device encounters an ON statement that specifies the same event and action as another ON statement active in the same terminal, the device ignores the new statement.

Note: To generate names for events with user and save areas, see "Coding variable event names with the DATASAVE statement"

For complete information about coding the ON statement, see the *WSim Script Guide and Reference*.

DEACT

The DEACT statement controls message generation by deactivating IF statement logic tests or ON statements, depending on the operand you code:

ONEVENTS=(*event*,*event*...)|ALL
: Selectively or globally deactivates ON conditions before normal deactivation, that is, after WSim completes the action specified by the ON statement.

IFS=(*num*,*num*...)|ALL
: Selectively or globally deactivates message generation logic tests before the time that WSim normally deactivates such tests.

To explicitly deactivate an ON statement, code the ONEVENTS operand on the DEACT statement, as shown in the following example:

```
DEACT1  DEACT  ONEVENTS=(READY)  Explicitly deactivates ON statements
*                                that specify event READY.
```

In the preceding example, the DEACT statement deactivates all ON statements that specify event READY before WSim would normally deactivate the ON statements.

For information about deactivating logic tests with the DEACT statement, see "Deactivating logic tests" on page 181.

Coding variable event names with the DATASAVE statement

In addition to coding event names directly on the EVENT and ON statement, you can generate or change event names dynamically during the simulation. To generate event names dynamically, you retrieve an event name from a save area or user area. In this way, the device can store or change the name in the save area or user area at any time, and any device in the network can change names in the network save and user areas.

For all message generation statements that specify an event name, you can code a save or user area with one of the following operand values. *value* is an integer 0 - 32766 or the name of a valid counter.

N±*value*
: Specifies an event name referenced at an offset from the start (+*value*) or back from the end (-*value*) of the network user area.

N*s***+***value*
: Specifies an event name referenced at an offset from the start of a network save area. *s* is an integer from 1 to 4095.

*s***+***value*
: Specifies an event name referenced at an offset from the start of the device save area. *s* is an integer from 1 to 4095.

U±*value*
: Specifies an event name referenced at an offset from the start (+*value*) or back from the end (-*value*) of a device user area.

When you define an event name in this manner, WSim takes the first 8 bytes of data beginning at the specified offset as the event name. If the area contains less than 8 bytes of data, WSim pads the name on the right with blanks. If the area does not exist or contains no data, WSim uses a name consisting of 8 blanks. WSim does not check the validity of the event name, which means you can generate a name that cannot be expressed as EBCDIC characters.

The following example shows how you can save and access a variable event name:

```
SAVE1  DATASAVE  AREA=N+0,      Saves name in network user area.
                 TEXT=(EVENT1)
        .
        .                       Message generation statements.
        .
WAIT1  WAIT      EVENT=N+0      Waits until EVENT1 is posted.
```

In the preceding example, WSim places the name EVENT1 into the network user area. When WSim processes the WAIT statement, the terminal waits until EVENT1 is posted.

In the following example, WSim places the text EVENT2 into the device save area 3. When WSim processes ON1, it retrieves EVENT2 from the device save area 3 and uses it as the event name.

```
SAVE1  DATASAVE  AREA=3,        Saves event name.
                 TEXT=(EVENT2)
        .
        .                       Message generation statements.
        .
ON1    ON        EVENT=3+0,     Specifies event name from device save
                 THEN=B-LABEL   area 3 and an action to be taken when
*                               the event is signaled.
```

For more information about the DATASAVE statement, see “DATASAVE” on page 196.

Controlling communications with events

In WSim, you can control communication among devices by managing events with the WAIT/POST and the ON/SIGNAL facilities. The WAIT/POST facility allows a terminal to wait until an event is posted. With the ON/SIGNAL facility, a terminal activates an asynchronous condition that WSim deactivates by signaling an event.

These two facilities are independent of one another. For example, posting an event has no affect on outstanding ON/SIGNAL conditions, even if the named event is identical for both conditions.

The following sections provide complete descriptions of each facility.

WAIT/POST facility

With the WAIT/POST facility, you direct a simulated resource to wait until an event is posted. As discussed in “EVENT” on page 209, events may be posted by the same terminal waiting on the event, by another terminal, or by the operator. The following example shows how to code the EVENT statement to post an event:

```
EVENT1  EVENT  POST=READY    Posts event READY.
```

The WAIT/POST facility involves WAIT and POST actions. For example, WSim can process a WAIT statement that specifies an event name during message generation or logic testing. After processing the statement, WSim cannot generate any further messages for that resource until the named event is posted. You can also cause a terminal to wait on an event by coding the EVENT operand on the WAIT message generation statement. When WSim processes this statement and operand, it does not generate any further messages for that terminal until the named event is posted.

At times, more than one terminal might be waiting on the same event or a single terminal might be waiting on multiple events. In fact, the meshing of events and terminals waiting on those events can be simulated in any combination you want.

Before a terminal can generate any further messages, all events on which it is waiting must be posted. When one event is posted, that terminal is no longer waiting on that event. After WSim posts all events that a terminal is waiting on, the wait is over, even if one event is reset before the others are posted.

After WSim posts an event, the event remains posted until explicitly reset. If WSim processes a WAIT statement when the named event is already posted, the terminal does not have to wait until the event is again posted. Note, however, that the WAIT statement still serves as a delimiter for message generation regardless of the completion of the event specified. If the event is already posted, the terminal can continue generating messages.

You can test for the completion of an event using a logic test. For example, test an event with an immediate logic test and then generate a message if the event is posted or issue a wait on that event if it is not posted.

WSim maintains ON conditions, that is, the condition of waiting on events to be signaled with an EVENT statement, independently from the standard conditions set by the WAIT indicators. For example, actions that turn off the WAIT indicator, such as taking a THEN=CONT action on a logic test, do not turn off the EVENT WAIT indicator. It must be turned off by posting the event. When WSim turns both of these indicators on, you must turn both off before WSim can generate additional messages. You can also enter commands from the console or use automatic terminal recovery to turn off the EVENT WAIT indicator and the WAIT indicator.

Note: You can use the A (Alter) operator command to post and reset named events in a network; see *WSim User's Guide* for more information about this command. For more information about the WAIT and WAIT EVENT indicators, see "Interrupting message generation with unconditional delimiters" on page 139.

ON/SIGNAL facility

When you use the ON/SIGNAL facility, you activate an asynchronous condition that is satisfied when WSim signals an event. WSim performs the action associated with the condition immediately after the condition is signaled. As discussed in "EVENT" on page 209, you signal an event with the SIGNAL operand on the event statement:

```
EVENT5  EVENT  SIGNAL=READY   Signals event READY.
```

Using the ON/SIGNAL facility, you simulate communication among terminals in different ways from the WAIT/POST facility. This facility uses the ON statement to set up the action to be taken when WSim signals an event. As discussed in "ON" on page 212, you can use any of the actions specified on the THEN and ELSE operands of the IF statement. After signaling the event, WSim takes the specified action, and the ON condition is no longer active. If you want the ON condition to remain active, code a second, identical ON statement in a deck that WSim processes after taking an action specified by the first statement.

When WSim signals an event, it does not affect any EVENT WAIT conditions that may have specified the same event name, nor does posting an event activate an ON condition. WSim maintains the two facilities separately even though both can specify the same event names. An ON condition must be active when WSim signals an event in order to be affected by the signal. Earlier signals of the same event have no effect on an ON condition activated after these signals.

After signaling an event, WSim triggers all active ON conditions specifying that event and queues their actions for execution. If multiple ON conditions specifying the same event are active for a single terminal, WSim can take multiple actions for a terminal as a result of a single signal. After WSim queues all actions, it executes these actions immediately. Since the actions can include additional signals, WSim adds actions specified by ON conditions for these signals to the queue and executes them in turn. WSim completes all actions in the queue before any further processing occurs.

For example, if terminal A signals an event for which terminal B has an active ON statement, WSim takes the action specified by terminal B's ON statement before continuing message generation for terminal A. Some of the possible actions that can result from an event being signaled are a call or branch, which may change statement pointers; an execute action, which may specify executing a block of message generation statements; or possibly the signaling of another event.

For more information about the execute action, which is specified by E (Execute) on THEN and ELSE operands on an IF statement, see "E (Execute)" on page 175.

Note: WSim deactivates ON conditions automatically after completing the specified action. You can also explicitly deactivate an ON statement before it is signaled with the ONEVENTS operand of the DEACT statement. For more information about the DEACT statement, see "DEACT" on page 212.

Using events to synchronize multiple devices

The following example demonstrates how you can use events to synchronize the message generation process across multiple devices. In this example, two message generation decks are processed by different devices: device LU1 processes DECKLU1 and device LU2 processes DECKLU2. When WSim processes these decks, device LU1 selects an order number from a data base and stores the number in the network user area at offset zero. Device LU2 then processes that order number.

```
DECKLU1 MSGTXT
*              Sample Message Generation Decks
*   These message generation decks use events to synchronize
*   message generation for multiple devices.
*
*                                         Beginning of deck for LU1.
         .
         .                                Message generation statements.
         .
SAVE1   DATASAVE  AREA=N+0,               Generates random number for a
                  TEXT=($RNUM,1,9999,4)  part number and places it into
*                                         the network user area.
EVENT1  EVENT     POST=LU2EVENT           Posts event LU2EVENT.
WAIT1   WAIT      EVENT=LU1EVENT          Interrupts message generation
*                                         for LU1 until event LU1EVENT is
*                                         posted.
EVENT2  EVENT     RESET=LU1EVENT          Resets event LU1EVENT.
        ENDTXT                            End of DECKLU1.
*
DECKLU2 MSGTXT
*                                         Beginning of deck for LU2.
WAITA   WAIT      EVENT=LU2EVENT          Interrupts message generation
*                                         for LU2 until event LU2EVENT is
*                                         posted.
EVENTA  EVENT     RESET=LU2EVENT          Resets event LU2EVENT.
MSG1    TEXT      (ORDER PART ),          Generates message with data
                  (NUMBER ),              retrieved from network user area.
                  ($RECALL,N+0$)
EVENTB  EVENT     POST=LU1EVENT           Posts event LU1EVENT.
         .
         .                                Message generation statements.
         .
        ENDTXT
```

The following steps describe how WSim processes these decks during message generation.

Steps:

1. When WSim processes DECKLU1, it places the part number into the network user area at a zero offset, and then processes EVENT1, posting event LU2EVENT. WSim then issues a wait for LU1EVENT and interrupts message generation for LU1.
2. The first time LU2 enters message generation, WSim issues a wait for LU2EVENT, interrupting message generation for LU2 until LU1 posts this event.
3. Because LU1 posted event LU2EVENT, LU2 can reenter message generation when all conditions for message generation have been satisfied. WSim processes EVENTA, which resets event LU2EVENT. The next time WSim processes DECKLU2, LU2 can wait for event LU2EVENT again. WSim continues processing DECKLU2, retrieving the part number from the network

user area and generating MSG1. After processing the part number, WSim posts event LU1EVENT, which enables LU1 to reenter message generation.

4. Because LU2 posted event LU1EVENT, LU1 can reenter message generation when all conditions for message generation have been satisfied. WSim processes EVENT2, which resets event LU1EVENT. The next time WSim processes DECKLU1, LU1 can wait on event LU1EVENT again. LU1 starts the process again by saving another part number, and the preceding steps are repeated.

Chapter 18. Generating messages for specific types of devices

This chapter discusses message generation requirements for specific types of devices. WithWSim, you can simulate a number of different devices attached to your simulated network. So that WSim can represent the activities of a real network effectively, you should code message generation decks that accurately reflect the type of data the real terminal sends. By following the special coding requirements for the devices discussed in this chapter, you can create message generation decks that provide appropriate simulations of the data sent by these devices.

The first part of this chapter discusses the factors you should consider when simulating display devices. A sample message generation deck is provided to help you understand how message generation proceeds for a display device. The chapter then discusses the factors you should consider when simulating SNA devices and the data field options you can use to indicate certain types of information for specific devices.

The remaining sections in this chapter discuss individual coding requirements for the following devices:

- IBM 3270 Information Display System
- IBM 5250 Display System

For other device-specific considerations, particularly for CPI-C, FTP, and Simple TCP and UDP simulations, refer to Part 1, “Defining WSim networks,” on page 1.

Generating messages for display terminals

In general, WSim does not perform any operations on the data stream of a terminal after completing the line control processing required to transmit or receive a message. However, for 3270 and 5250 (LU7) display terminals, WSim performs full buffer simulation by maintaining in storage an image of the display screen for the terminal. For received messages, WSim interprets the commands and data stream orders for these terminals and automatically updates the screen image as required. For transmitted messages, WSim automatically builds the correct data stream from the buffer image.

WSim provides several message generation statements that are valid for display terminals only. Most of these statements represent keys that an operator strikes at a terminal. These key simulation statements are listed in “How delimiters are classified” on page 138 under the item “Delimiters for Specific Simulated Devices”.

Message generation for a display terminal continues until the attention identifier (AID) is set and a subsequent delimiter statement is encountered. When one of the key simulation statements is encountered during message generation and the AID has not been set for the terminal, the statement is processed as a control statement and message generation continues until a delimiter statement is encountered.

If the AID has already been set when a key simulation statement is encountered, the key simulation statement itself acts as a delimiter that interrupts message generation. It is not processed until the next pass through message generation for the terminal.

WSim maintains the position of the cursor for a display terminal. When you specify data to be entered in the terminal buffer with a TEXT statement, the data is entered by WSim at the current cursor position. You can move the cursor without entering data by using the cursor positioning statements such as BTAB, CTAB, CURSOR, TAB, and HOME. If you move the cursor in this manner, WSim assumes that more data is to be entered into the buffer; data from a subsequent TEXT statement is placed in the buffer without sending any data already in the buffer to the system under test.

If you code two consecutive TEXT statements for a display terminal, WSim assumes that an ENTER statement (for 3270 and 5250) is between them. The first TEXT statement puts data into the terminal buffer, the implied ENTER or SEND sets the AID, and the second TEXT statement acts as a delimiter and ends the current pass through message generation.

INPUT INHIBITED indicator

For the 3270 and LU2 display terminals, WSim maintains an INPUT INHIBITED indicator. Like the WAIT, QUIESCE, and EVENT WAIT indicators, the INPUT INHIBITED indicator must be in the Reset (off) state before you can generate a message for one of these terminals. Whenever a message is generated, the INPUT INHIBITED indicator is set. WSim resets the INPUT INHIBITED indicator when any of the following conditions occur:

- A keyboard unlock command sequence is received.
- The intermessage delay expires.
- An S (Start) operator command is executed.
- A BIND for LU2 terminal is received.
- An end bracket for LU2 terminal running an LU2 session is received.
- A RESET statement is executed.
- Automatic terminal recovery is performed.
- Console recovery mode is entered.
- A console recovery subcommand is entered.

Simulating the enter and tab keys

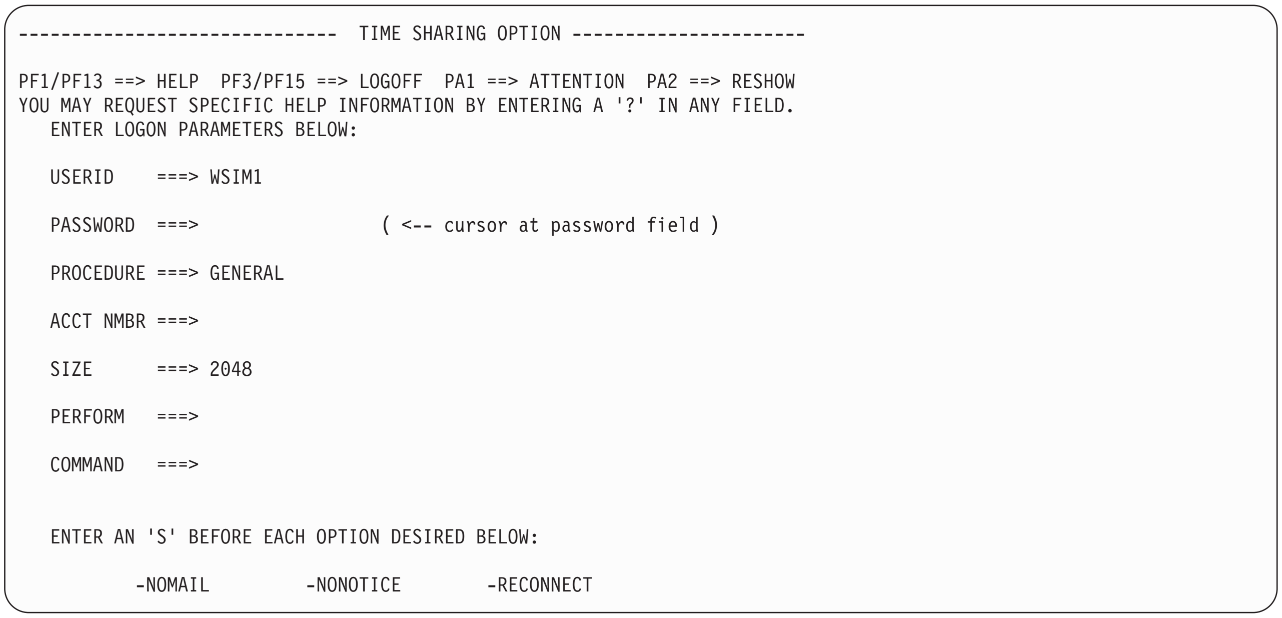
Use the ENTER and TAB statements to simulate the actions of the Enter and Tab keys on display devices.

ENTER Use the ENTER statement to simulate the actions of the Enter key. Although WSim automatically generates an Enter AID, it does so only if no other AID byte has been set at the time. For example, program function (PF) keys, and program attention (PA) keys can also be used to set AID bytes. Use the ENTER statement to simplify the understanding and readability of message generation decks. It is generally coded following the TEXT statement that is to be entered.

TAB Use the TAB statement to simulate the action of the Tab key. The TAB statement is useful for tabbing from one input field on the panel to the next on full-screen menus. Its most important use is to concatenate two or more individual TEXT statements in a message generation deck so that the first TEXT is not transmitted when the second TEXT is encountered. This statement is commonly used in IMS, CICS, and TSO simulations.

To visualize the functions of the Enter and Tab statements, consider entering data in the TSO logon screen shown in Figure 21. The message generation deck fills in all applicable fields before it simulates pressing the Enter key. The following example shows the message generation deck that simulates these actions. The TEXT statements are concatenated with the TAB statement, which also moves the cursor to the next unprotected field. Without the TAB statements, the TEXT statements would be transmitted one at a time, causing TSO to continue to prompt the user for required fields.

When all necessary fields were entered, the screen looks like the one shown in Figure 22 on page 222. Then the deck simulates pressing the Enter key to transmit the data.



```
------------------------------  TIME SHARING OPTION ----------------------

PF1/PF13 ==> HELP  PF3/PF15 ==> LOGOFF  PA1 ==> ATTENTION  PA2 ==> RESHOW
YOU MAY REQUEST SPECIFIC HELP INFORMATION BY ENTERING A '?' IN ANY FIELD.
   ENTER LOGON PARAMETERS BELOW:

   USERID    ===> WSIM1

   PASSWORD  ===>                 ( <-- cursor at password field )

   PROCEDURE ===> GENERAL

   ACCT NMBR ===>

   SIZE      ===> 2048

   PERFORM   ===>

   COMMAND   ===>


   ENTER AN 'S' BEFORE EACH OPTION DESIRED BELOW:

           -NOMAIL         -NONOTICE        -RECONNECT
```

Figure 21. TSO logon panel before data entry

```
TSOLOG  MSGTXT
        TEXT   (PASSWD)     Enter password.
        TAB                 Tab to procedure field.
        TEXT   (WSIM)       Enter procedure name.
        EREOF               Erase last part of "GENERAL."
        TAB                 Tab to account field.
        TAB                 Tab to size field.
        TEXT   (4096)       Enter size.
        ENTER               Now press the Enter key.
        ENDTXT              Delimiter - transmit data.
```

Note that in Figure 21, the cursor is at the start of the PASSWORD field before entering message generation. The TAB statements move the cursor from one field to the next after the data is typed. Note that if the cursor automatically skips to the next field while you are typing (such as if an 8-character password is entered), a TAB is not necessary after that TEXT statement.

Figure 22 on page 222 shows the same panel after data entry.

```
------------------------------  TIME SHARING OPTION ----------------------

PF1/PF13 ==> HELP  PF3/PF15 ==> LOGOFF  PA1 ==> ATTENTION  PA2 ==> RESHOW
YOU MAY REQUEST SPECIFIC HELP INFORMATION BY ENTERING A '?' IN ANY FIELD.
   ENTER LOGON PARAMETERS BELOW:

   USERID    ===> WSIM1

   PASSWORD  ===>                ( password is invisible )

   PROCEDURE ===> WSIM

   ACCT NMBR ===>

   SIZE      ===> 4096

   PERFORM   ===>

   COMMAND   ===>


   ENTER AN 'S' BEFORE EACH OPTION DESIRED BELOW:

           -NOMAIL         -NONOTICE        -RECONNECT
```

Figure 22. TSO logon panel after data entry

Following message generation for a display terminal

The following example illustrates the message generation process for a 3270 synchronous data link control (SDLC) terminal using sample message generation trace (MTRC) records and a listing produced by the Preprocessor. Each step WSim takes during message generation is described using the corresponding numbers on the MTRC records and the message generation decks on the Preprocessor listing.

For more information about MTRC records and the Preprocessor, see *WSim Utilities Guide*.

```
NET3270  NTWRK INIT=SEC,UTI=100,MSGTRACE=YES
*
*        Sample 3270 Network Definition
*
1        PATH  ECHO3270
010021   LINE  PATH=(1),FRSTTXT=LOGON,TYPE=3270
TERM211  TERM  ADDR=C1
DEV2111  DEV

STMT#
*
*   LOGON message deck:
*
         LOGON    MSGTXT
00001             DELAY TIME=30
00002             TEXT  (LOGON APPLID((NEWAPPL)))
00003             SYSREQ
00004             ENDTXT

*
*   ECHO message deck:
*
         ECHO3270 MSGTXT
00001             TEXT   (SEND FIRST TEST MESSAGE)
00002             TEXT   (THIS MESSAGE WILL )
00003             CURSOR ROW=1,COLUMN=19
00004             TEXT   (BE CONCATENATED TOGETHER)
```

```
00005           TEXT   (THIS TEXT IS SENT SEPARATELY)
00006           PF1
00007           TEXT   (SHOW TAB KEY )
00008           TAB
00009           TEXT   (PROCESSING)
00010           CURSOR
00011           ENTER
00012           ENDTXT
```

Steps:

1. During initial start, the LOGON message generation deck is chosen from the FRSTTXT operand. The DELAY statement is processed and processing stops at the TEXT statement. No message is generated.
   ```
   ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK LOGON
   ITP448I MSG GEN ENDED:   STMT# 00002 OF DECK LOGON
   ```
2. When the delay has expired and a poll is received, message generation continues at the TEXT statement. The data is placed in the device buffer and processing continues to the next delimiter. The SYSREQ statement is encountered. For 3270 SDLC, this statement sets the terminal AID to the same value that a real terminal would set for the TESTREQ key. Since the AID was not set before the SYSREQ statement, processing continues to the ENDTXT statement. Message generation stops and the message is transmitted.
   ```
   ITP447I MSG GEN ENTERED: STMT# 00002 OF DECK LOGON
   ITP448I MSG GEN ENDED:   STMT# 00004 OF DECK LOGON
   ```
3. Message generation continues with the selection of another message generation deck (ECHO3270) from the specified PATH. The first TEXT statement is processed and message generation stops at the second TEXT statement. The first message is now transmitted. Since no 3270 key has been processed, the message is automatically sent as if the Enter key has been pressed by the terminal operator. The same processing would occur if an ENTER statement were placed between the two TEXT statements.
   ```
   ITP447I MSG GEN ENTERED: STMT# 00004 OF DECK LOGON
   ITP449I MSG GEN CONTINUES: DECK ECHO3270 STARTED
   ITP448I MSG GEN ENDED:   STMT# 00002 OF DECK ECHO3270
   ```
4. Message generation continues at the second TEXT statement. The data is placed in the buffer at the current cursor position (assumed in this case to be the first buffer position). The cursor is set to the position indicated by the CURSOR statement. This CURSOR statement indicates that more data is to be entered into the buffer before a message is transmitted. Instead of stopping message generation at the following TEXT statement, that TEXT statement is also processed and the data is entered into the buffer. Processing continues to the fourth TEXT statement in the deck, then stops. The message in the buffer is transmitted. In this example, the message transmitted reads: THIS MESSAGE WILL BE CONCATENATED TOGETHER.
   ```
   ITP447I MSG GEN ENTERED: STMT# 00002 OF DECK ECHO3270
   ITP448I MSG GEN ENDED:   STMT# 00005 OF DECK ECHO3270
   ```
5. When message generation continues, the next TEXT statement (THIS TEXT IS SENT SEPARATELY) is processed, and the data is entered into the buffer. The PF1 key is then processed, and the device AID is set as it would be if the terminal operator pressed the PF1 key. The processing routine then detects the next TEXT statement and processing stops. The message is transmitted.
   ```
   ITP447I MSG GEN ENTERED: STMT# 00005 OF DECK ECHO3270
   ITP448I MSG GEN ENDED:   STMT# 00007 OF DECK ECHO3270
   ```
6. When message generation continues, the next TEXT statement (SHOW TAB KEY) is the current deck position. The data is placed in the device buffer and processing continues. The TAB statement is processed much the same as the

CURSOR statement; the current cursor position changes, and the generation routine assumes that more data is to be entered at the new position.

When the next TEXT statement is processed, that data is also entered in the buffer and processing continues to the next delimiter. The CURSOR statement sets the cursor back to the first buffer position. When the ENTER statement is encountered, the AID byte is set and processing continues. When the ENDTXT is encountered, the previously generated message is transmitted with the Enter key AID.

```
ITP447I MSG GEN ENTERED: STMT# 00007 OF DECK ECHO3270
ITP448I MSG GEN ENDED:   STMT# 00012 OF DECK ECHO3270
```

7. When the message generation routine is entered again, another message generation deck is chosen from the PATH statement. Since the PATH statement contains only the deck named ECHO3270, processing continues through the ECHO3270 deck again.

```
ITP447I MSG GEN ENTERED: STMT# 00012 OF DECK ECHO3270
ITP449I MSG GEN CONTINUES: DECK ECHO3270 STARTED
ITP448I MSG GEN ENDED:   STMT# 00002 OF DECK ECHO3270
```

Generating messages for SNA terminals

This section discusses how you can modify SNA messages by altering the default SNA headers built by WSim, controlling chaining of messages, and specifying SNA or user commands. It discusses how to code message generation decks to initiate sessions for SNA terminals and how to generate unsolicited SNA data from simulated physical units, such as communication controllers and terminal control units.

Modifying SNA messages

To modify SNA messages transmitted by your simulated devices, you can use the TH statement to alter the SNA transmission header, the RH statement to control the chaining of messages, and the CMND statement to send SNA or user commands to the system under test.

TH statement

Use the TH statement to modify the SNA transmission header for a message generated by a TEXT statement or a command generated by a CMND statement. You can use the TH statement to set the sequence number field in the transmission header. Code the TH statement following the message generation deck statement that builds the message.

RH statement

Use the RH statement to control the chaining of a transmitted message, as shown in the following example:

```
CHAINS  MSGTXT
        TEXT     (FIRST DATA)
        RH       CHAIN=FIRST,EXC=ON
        TEXT     (MORE DATA)
        TEXT     (EVEN MORE DATA)
        TEXT     (LAST DATA)
        RH       CHAIN=LAST,EXC=OFF
        TEXT     (ONLY DATA)
        ENDTXT
```

When the message generation deck is processed for a logical unit, the first message sent is a first-in-chain RU. The next two RUs transmitted for the logical unit are marked as middle-in-chain. If you omit the RH statement and a first-in-chain RU has been sent for a logical unit, WSim automatically sends subsequent RUs as middle-in-chain. The next message is transmitted as a last-in-chain RU that requests a definite response from the test system. WSim does not generate another message for the logical unit until the response is received. The last message generated from this message generation deck is sent as an only-in-chain RU.

When CHAINING=AUTO is specified in the DEV or LU definition for a non-display SNA device, WSim automatically performs the chaining operation on the message generation text data using the maximum RU size value from the BIND. WSim can use BUFSIZE operand values up to 32767 during the message generation process.

The following example shows how to use the automatic chaining support. In the example, assume that CHAINING=AUTO, BUFSIZE=32767, and the maximum RU size value in the BIND is 256.

```
AUTOC   MSGTXT
TEXT1   TEXT  (MESSAGE 1 ),LENG=5000
TEXT2   TEXT  (MESSAGE 2A),LENG=5000
RH1     RH    CHAIN=FIRST
TEXT3   TEXT  (MESSAGE 2B),LENG=5000
RH2     RH    CHAIN=MIDDLE
TEXT4   TEXT  (MESSAGE 2C),LENG=32767
RH3     RH    CHAIN=LAST
        ENDTXT
```

In this example, WSim processes the message generation deck as follows:

Steps:

1. The first TEXT statement builds a 5000-byte message into the terminal output buffer. The message is passed to any output logic test as an only-in-chain RU. The automatic chaining process takes the message and divides it into 256-byte chain elements. The chain is sent as a single first-in-chain element, multiple middle-in-chain elements, and a single last-in-chain element. When used with the RH statement, the automatic chaining process can generate a large chain.
2. The next three TEXT and RH statements generate a large chain using multiple passes through the message generation process. The second TEXT statement builds a 5000-byte message into the terminal output buffer. The first RH statement sets the first-in-chain indicator. The message is passed to the logic test as a first-in-chain RU. The automatic chaining process takes the message and divides it into 256-byte chain elements. A single first-in-chain element is sent, followed by multiple middle-in-chain elements.
3. The third TEXT statement builds a 5000-byte message into the terminal output buffer. The second RH statement sets the middle-in-chain indicator. The message is passed to the logic test as a middle-in-chain RU. The automatic chaining process takes the message and divides it into 256-byte chain elements. Multiple middle-in-chain elements are sent.
4. The fourth TEXT statement builds a 32767-byte message into the terminal output buffer. The third RH statement sets the last-in-chain indicator. The message is passed to the logic test as a last-in-chain RU. The automatic chaining process takes the message and divides it into 256-byte chain elements. Multiple middle-in-chain elements are sent, followed by a single last-in-chain element.

CMND statement

Use the CMND statement to specify an SNA or user command to be sent by a logical unit to the system under test. WSim builds default SNA headers for the command, but you can modify the headers with the TH and RH statements. The following example shows how to use the CMND statement to generate an INIT-SELF command.

```
CMNDDECK  MSGTXT
          CMND    COMMAND=INIT,
                  MODE=LOGLU0,
                  RESOURCE=APPL1,
                  DATA=($ID,8$/PASSWORD),
                  URC=($ID,8$)
          ENDTXT
```

The following example demonstrates how to use the CMND statement to build a SIGNAL command:

```
CMND  COMMAND=SIGNAL,SENSE='00010000'    SNA attention key (SIGNAL).
```

The CMND statement can be used to generate asynchronous SNA data flows. When the CMND statement is the next statement in a message generation deck, the CMND statement is executed, even if the normal requirements for entering message generation have not been met. The following example demonstrates how to use this asynchronous message generation capability:

```
SLU      MSGTXT
*  Send SNA signal when a chain of data is received which
*  does not allow the terminal to send the next message.
*
0        IF     LOC=RH+0,TEXT='80',THEN=SW1(OFF),  SNA request and
                ELSE=SW1(ON)
1        IF     LOC=RH+0,TEXT='40',THEN=SW1(OFF)   FM ...
2        IF     LOC=RH+0,TEXT='20',THEN=SW1(OFF)   ... data and
3        IF     LOC=RH+0,TEXT='01',ELSE=SW1(OFF)   end of chain and
4        IF     LOC=RH+2,TEXT='40',THEN=SW1(OFF)   not end bracket and
5        IF     LOC=RH+2,TEXT='20',THEN=SW1(OFF)   not change direction.
6        IF     LOC=SW1,THEN=C-SIGNAL              Generate SNA signal.
WAITING  WAIT
         BRANCH LABEL=WAITING
SIGNAL   CMND   COMMAND=SIGNAL,SENSE='00010000'    SNA attention key.
         WTO    (SIGNAL SENT)
         RETURN
         ENDTXT
```

In this example, if the simulated device receives a chain of data that does not allow the device to transmit a following message (for example, change direction is not received and end bracket is not received), the device sends an asynchronous SIGNAL command.

Two operands of the CMND statement (MODE and RESOURCE) accept variable names that can be dynamically generated or changed during the simulation. These variable names can be stored in and retrieved from save and user areas. Any device in the network can change the names in the network user area. You can use the DATASAVE statement to alter the contents of a save area or user area.

You can specify the name of the save or user area and reference an offset with one of the following operand values:

- N±*value*
- N*s*+*value*
- *s*+*value*
- U±*value*

N indicates the network user area, N*s* is the number of a network save area, *s* is the number of a device save area, and U indicates a device user area. *value* is the offset into the area. When you define a name in this manner, the first 8 bytes of data, beginning at the specified offset, are taken as the name. If the area does not exist or if no data is present, a name of eight blanks is used. No validity checking is performed on the name; therefore, you can use a name that cannot be expressed as EBCDIC characters.

For the network and device user area, the name must be padded with blanks in your coding if the length is less than eight.

The following example demonstrates how you can use DATASAVE to place WSIMLU2 into the network user area. A session can then be established for WSIMLU2.

```
SAVE1  DATASAVE  AREA=N+0,TEXT=(WSIMLU2 )
CMND1  CMND      COMMAND=INIT,RESOURCE=N+0
```

In the following example, LU11 is placed into the device user area. A session can then be established for LU11.

```
SAVE5   DATASAVE  AREA=U+0,TEXT=(LU11    )
CMND5   CMND      COMMAND=INIT,RESOURCE=U+0
```

In the following example, the first name from user table 1 is placed into the network user area.

```
SAVE3   DATASAVE  AREA=N+0,TEXT=($UTBL,1,0$)
CMND3   CMND      COMMAND=INIT2,RESOURCE=N+0
```

In this user table, the *luname* should be coded as follows:

```
1 UTBL   (LU11    )
```

In the following example, WSIMLU2 is placed into device save area 1. A session can then be established for WSIMLU2.

```
SAVE8   DATASAVE  AREA=1+0,TEXT=(WSIMLU2 )
CMND8   CMND      COMMAND=INIT,RESOURCE=1+0
```

The following example demonstrates that the first name from user table 1 is placed into device save area 2. A session is then established.

```
SAVE9   DATASAVE  AREA=2,TEXT=($UTBL,1,0$)
CMND9   CMND      COMMAND=INIT2,RESOURCE=2+0
```

Simulating errors in SNA devices

You can simulate SNA terminal errors by using the following statements or operand:

- RESP statement
- RESP operand on the IF statement
- TH statement
- RH statement
- CMND statement.

RESP statement

Use the RESP statement to cause WSim to send an exception response to the next request received instead of the response that would normally be sent. The RU portion of the response is composed of the SNA and user sense bytes specified by the SENSE operand, as shown in the following example.

```
TSTRESP  MSGTXT
         RESP    SENSE='10030000'
         TEXT    (MSG1)
         ENDTXT
```

In this example, when the first request is received from the system under test after transmitting MSG1, the SNA terminal responds with sense data specifying that the function is not supported.

RESP operand on the IF statement

Normally, WSim sends an SNA response to a received request. However, you can use the RESP=NO operand on an IF statement to prevent WSim from sending this response when the request is tested. Instead, WSim builds the TH and RH for the normal response and enters message generation to allow the user to supply the RU data and modify the headers. Use the TEXT statement to generate the RU portion of the response. The following example shows how you can reject a message that WSim would not normally reject.

```
DECK6  MSGTXT
        .
        .
        .
TEXT1  TEXT    (INQUIRY)
0      IF      LOC=B+30,TEXT=(DATA),
               THEN=CERROR1,RESP=NO,
               ELSE=CONT,TYPE=3277
        .
        .
        .
       ENDTXT
ERROR1 MSGTXT
TEXT2  TEXT    ('00001234')             User SENSE data.
       RH      EXC=ON,SNI=ON
       ENDTXT
```

In the preceding example, assume that buffer position 30 of the data reply to the message generated by TEXT1 is DATA, which satisfies the logic test condition and causes the THEN action to be taken. Message generation deck ERROR1 is called. Since RESP=NO is coded on the 0 IF statement, WSim builds the SNA response from the TEXT2 statement. An exception response is generated with sense data of X'00001234'.

TH statement

Use the TH statement to change the sequence number field in the transmission header that WSim transmits for requests and responses. The error condition that you can create is an invalid FID2 sequence number. The message MSG1 is transmitted with an invalid sequence number that must be detected by the system under test.

RH statement

Use the RH statement to change fields in the request or response header that WSim transmits. The error conditions that you can create include invalid chaining sequence, bracket protocol error, invalid response type (DR1, DR2, EXC), and incorrect RU type (FM, DFC, NC, SC).

```
TSTRH   MSGTXT
        TEXT  (MSG1)
        RH    CHAIN=FIRST
        TEXT  (MSG2)
        RH    CHAIN=FIRST
        ENDTXT
```

The preceding example causes WSim to transmit two consecutive first-in-chain RUs.

CMND statement

Use the CMND statement to generate SNA commands and send them to the system under test. You can send SNA sense data along with the LUSTAT and SIGNAL commands.

```
TSTCMND  MSGTXT
         CMND  COMMAND=LUSTAT,SENSE='08020000'
         ENDTXT
```

The above CMND statement causes an LUSTAT command to be transmitted with sense data indicating that intervention is required at the logical unit.

Initiating sessions for SNA terminals

When INIT=SEC is coded in the network definition for a simulated SNA terminal, the terminal sends an INIT-SELF command to the system under test requesting a session with an application named in the RESOURCE operand of the definition. You can also initiate the session by coding a CMND statement that generates a formatted INIT-SELF command. The following message generation deck causes a terminal to generate an INIT-SELF command that contains a resource name of RES and a logon mode table entry name of BATCH. The terminal does not continue in message generation until the terminal receives an SDT command.

```
LOGINIT MSGTXT
        CMND   COMMAND=INIT,RESOURCE=RES,MODE=BATCH
        ENDTXT
```

If your test system processes unformatted system services (USS) requests, you can code a SYSREQ statement followed by a TEXT statement containing a logon message for the SLU. In the following example, the DELAY statement causes different start delays when multiple terminals reference the message generation deck to generate their logon messages. The SYSREQ statement places the terminal in the SSCP-LU session state. The TEXT statement generates the logon message. The WAIT statement stops message generation and sets the WAIT indicator for the terminal. The logon message is transmitted and the terminal WAIT indicator remains set until the terminal receives a BIND command.

```
LOGON     MSGTXT
          DELAY  TIME=A20
0         IF     LOC=RU+0,TEXT=(ANYTHING),
                 ELSE=B-NOSYSREQ
          SYSREQ
NOSYSREQ  TEXT   (LOGON APPLID((TSO)) USER(($ID,8$/PASSWORD)))
          WAIT
          ENDTXT
```

IBM 3270 Information Display System

WSim simulates a 3270 terminal by maintaining an updated buffer or screen image for the terminal. The buffer can be modified by generated messages or by messages received from the system under test. For a message generated by a simulated 3270 terminal, WSim builds the message from the current state of the terminal buffer including any necessary orders in the data stream to be transmitted. For a received message, all commands and orders other than 3270 printer orders are processed and the terminal buffer is updated accordingly. WSim automatically maintains attribute bytes in the buffer that define different display or data entry fields.

For a complete list of the IBM 3270 components WSim can simulate, see Part 1, “Defining WSim networks,” on page 1.

Generating messages

WSim can generate messages for simulated 3270 terminals that represent display devices. No messages are generated for 3270 printers by normal message generation. A simulated 3270 display terminal generates a message if all of the following conditions are met:

- A poll sequence has been received.
- The intermessage delay has expired.
- The logical WAIT indicator is not set.
- The terminal WAIT EVENT indicator is not set.
- A status message is not pending.
- The INPUT INHIBITED indicator is not set.
- The terminal is not in the quiesce state.
- The terminal is not in the console recovery state.

The INPUT INHIBITED indicator for a terminal is set whenever a message is generated. No additional normal messages can be generated for the terminal until the indicator is reset. For a list of conditions that cause WSim to reset the INPUT INHIBITED indicator, see “INPUT INHIBITED indicator” on page 220.

A message generated by the statements in a message generation deck is treated as data entered at a 3270 display keyboard. This data is used to modify the screen image buffer for the terminal according to the current cursor location and the attribute bytes in the buffer. If you attempt to enter data into a display field that has the protected attribute, the data is ignored and an informational message is written to the log data set.

After a message is generated, WSim scans the terminal's screen image buffer and constructs the data stream to be transmitted to the system under test, including headers, addresses, the attention identifier (AID), and orders. For 3270 SNA, the data is transmitted in RU chain elements according to the value that you code for BUFSIZE.

For Telnet 3270, the data is transmitted in blocks according to the value that you code for BUFSIZE.

Using the RESET statement

The RESET statement is valid for 3270 terminals only and is ignored for other terminal types. The RESET statement sets the AID for the terminal to a null value and resets the INPUT INHIBITED indicator. The RESET statement also resets insert mode set by the INSERT statement, as discussed in “Simulating the insert and delete keys” on page 231.

You use the RESET statement primarily as the response to an IF statement E (Execute) action. For example, if a 3270 terminal normally enters a logoff message and the application sends a response without unlocking the terminal keyboard, you can use an IF statement to identify the response and execute a RESET statement to reset the keyboard so that the terminal can continue in message generation. This sequence is demonstrated in the following example.

```
NTWRK
  .
  .
```

```
         .
       IF   LOC=D+0,TEXT=(LOGGED OFF),SCAN=YES,THEN=ERESET
         .
         .
         .
RESET  MSGTXT
       RESET
       ENDTXT
```

When encountered during normal message generation, the RESET statement functions like other 3270 key simulation statements. If the AID byte has already been set when a RESET statement is encountered, message generation stops and the generated message is transmitted. If the AID byte has not been set, the RESET statement is processed but is effectively ignored since the AID byte is already null and the INPUT INHIBITED indicator is already reset. The RESET statement does not concatenate TEXT data as do statements such as CURSOR and TAB.

Once message generation is entered for a 3270 terminal, the processing of message generation statements normally continues until data is entered into the terminal buffer and the AID byte is set. You can use the CLEAR, ENTER, PA*n*, PF*n*, and SELECT statements to set the AID byte. The SELECT statement may or may not set the AID byte depending on the character selected. If the character selected is detectable and the designator character for the field selected is valid, one of the following actions is taken:

- If the designator character is “?”, the modified data tag (MDT) bit for the field is set and the designator character is changed to “>”.
- If the designator character is “>”, the MDT bit for the field is reset and the designator character is changed to “?”.
- If the designator character is a blank or null, the MDT bit for the field is set and the selector pen AID is set.
- If the designator character is “&”, the MDT bit for the field is set and the Enter AID is set.

Simulating the insert and delete keys

Use the INSERT and DELETE statements to simulate the actions of the 3270 Insert and Delete keys. The INSERT statement puts the simulated 3270 display into insert mode. In insert mode, data on the simulated screen is shifted to the right as the TEXT data is inserted. By ending message generation or using the RESET statement, you can reset insert mode. If there are not enough nulls at the end of a screen row to satisfy the insertion requirements, log message ITP470I is generated.

Use DELETE CHARS=*n* to delete *n* characters from the simulated screen. The value of *n* can range 1 - 255 and can be a counter value.

Simulating cursor movement

The CURSOR statement simulates the action of the cursor positioning keys on a display device. Use the following operands on the CURSOR statement during 3270 and 5250 simulations:

UP=*value*
: Moves the cursor up the specified number of lines. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

DOWN=*value*
: Moves the cursor down the specified number of lines. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

LEFT=*value*
: Moves the cursor left the specified number of positions. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

RIGHT=*value*
: Moves the cursor right the specified number of positions. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

COLUMN=*value*
: Moves the cursor to the specified column. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

ROW=*value*
: Moves the cursor to the specified row. *value* is an integer 1 - 255 or the name of a counter set to a valid value.

OFFSET=*value*
: Specifies the position of the cursor as an offset from the beginning of the display buffer. For a 3270 simulation with multiple partitions defined, this operand specifies the cursor offset from the beginning of the presentation space of the currently active partition. *value* is an integer 0 - 32766 or the name of a counter set to a valid value.

3270 key options

The following list describes the data field options that enable you to simulate certain keys on 3270 devices. These options are ignored for device types other than 3270. In addition, the following data field options are valid only in the TEXT statement data field.

FM Simulates the action of the 3270 Field Mark key. Each time this option is detected, a Field Mark character is entered into the buffer.

NL Simulates the action of the 3270 New Line key. The cursor is set to the first unprotected character location of the next line. If no unprotected fields exist, the cursor is set to character location zero. If the display contains no fields, the cursor is set to the first position of the next line.

TAB Simulates the action of the 3270 Tab key. The cursor is moved to the first byte of the next unprotected field.

Simulating the 3274 local clear key

The LCLEAR statement simulates the Local Clear key on a display device. It clears the display of a simulated 3270 terminal, but will not transmit a Clear AID to the host. For a detailed description of the LCLEAR statement, see the *WSim Script Guide and Reference*.

Logic testing

You can use the IF statement to perform logic tests for 3270 terminals on two different forms of the data. You can specify a logic test in a terminal's screen image buffer by coding the B+, B-, C+, C-, or (*row*,*col*) location options on an IF statement. This type of logic test operates on the data as it would be displayed at a real terminal. Since bits 0 and 1 of attribute bytes are always zero in the screen image buffer, you can test bits 2 through 7 of an attribute byte using one of the above location options. All logic tests on a screen image buffer are performed after the buffer has been modified according to the message generated or data received. If a received message contains invalid commands or orders, the screen image buffer is not modified, but any active logic tests are still performed.

You can specify a logic test on an incoming or outgoing data stream, including headers, commands, and orders, by coding the D+, TH+, RH+, or RU+ location option on an IF statement. The TH+, RH+, and RU+ options are valid only for 3270 SNA terminals. You can use one of these options to perform a logic test on a start field (SF) order and its following attribute byte.

See “Logic test examples” on page 186 for scripts coded with logic tests.

Simulating errors in an LU2 terminal

Use the RESP and ERROR statements to cause WSim to send an exception response to the next request received instead of the response that would normally be sent. For LU2 terminals, the RESP statement specifies the SNA sense and user sense bytes that will be sent. For an LU2 terminal, the ERROR statement sets the user sense bytes to X'0000' and the SNA sense to X'1003' for command reject or X'1005' for any other error.

Simulating printers

WSim enters message generation for a start delay and allows you to activate IF statements to send printer status information. WSim also generates appropriate line control information and status messages to simulate the start, duration, and end of a printer operation initiated by a message from the system under test.

If a printer operation is requested before the completion of a previous operation, a device busy status message is generated and transmitted in response to the next poll sequence. At the end of a simulated printer operation, a device end status message is automatically generated.

WSim does not support the new line and end of media orders. The duration of a simulated printer operation depends on the PRTSPD value coded on the DEV or LU network definition statement and the number of non-null characters to be printed from the terminal buffer. The amount of time required for this operation is the number of non-null characters divided by the PRTSPD value.

Telnet 3270E supports printer simulation.

Simulating the 3278 magnetic stripe reader

The 3278 Magnetic Stripe Reader (MSR) attaches to a 3270 display device and allows data to be input by reading a magnetic stripe. Data from an MSR can be nonsecure or secure data.

WSim simulates the MSR in two ways, based on whether the data is nonsecure or secure. Nonsecure data is displayed on the screen and can be altered by the operator before it is sent to the host. When nonsecure data is read in from the MSR, no field attribute or AID is generated. The data is displayed and then sent in the same way as operator data, for example, with the Enter key or a PF key. For this reason, nonsecure data is specified for WSim in a TEXT statement. As far as the application is concerned, an operator typed in the data.

When secure data is successfully read, a field attribute is generated at the position of the cursor if it is an unprotected character location. The attribute defines the stripe data field as protected and the MDT bit is set on. If extended and character attributes are supported, they are set to X'00'. The operator cannot alter the data. It is automatically sent to the host in a READ MODIFIED operation with an AID of X'E7'.

Use the STRIPE statement to specify secure magnetic stripe data in message generation decks. The STRIPE statement has the same format and options as a TEXT statement. However, it causes an AID of X'E7' to be set when it is processed. The data is sent to the host in a READ MODIFIED operation when the next delimiter statement in the deck is encountered. Data on a real magnetic stripe is limited to 125 characters. If more than 125 characters are specified on the STRIPE statement, they are ignored. The STRIPE statement is ignored for nondisplay devices.

Simulating the Data Analysis/APL Character Set

The 3270 Data Analysis/APL Character Set (feature code 1066) expands the character set of 3270 terminals by allowing the display of 80 APL-specific characters and 35 TN print train characters not included in the normal character set. The characters that can be entered at the keyboard are dependent upon whether the APL or Text keyboard was installed. Some of the APL-specific characters and all of the additional TN characters are transmitted to and from a 3270 terminal as 2-character sequences consisting of the start field (SF) order (X'1D') plus a second byte that varies depending on the character. The set of allowable values for the second byte is mutually exclusive with the set of values that represent valid attribute bytes when encountered following the SF order.

The ALTCSET=NONE and EXTFUN=NO operands on the DEV or LU statements specify the Data Analysis/APL Character Set feature on any 3270 terminal, regardless of model.

You can enter any 8-bit character as data with the TEXT statement. You must ensure that this data does not include line control or other invalid characters. To include characters that must be transmitted as 2-character sequences, you must specify the hexadecimal value that corresponds to the WSim internal representation of that character. Table 13 on page 271 shows the hexadecimal values you should use for WSim to transmit specific 2-character sequences. It is your responsibility to restrict the characters entered at the simulated keyboard to those that can be entered at a particular type of real keyboard (APL or Text, for example).

Table 14 on page 273 shows the WSim internal hexadecimal value translated into the terminal buffer for each received 2-character sequence. If the internal WSim representation of one of the 2-character sequences is received in the data stream from the application, it is interpreted as the 2-character sequence. If an invalid 2-character sequence is received, for example, an SF order followed by an invalid character, WSim translates the sequence to an X'60' character.

See *IBM 3270 Information Display System* for the specific code assignments for the 3270 Data Analysis/APL Character Set, for both the 1-character and 2-character sequences.

Simulating the APL/Text Character Set

WSim supports the 3270 APL/Text Character Set (feature code 1120) when ALTCSET=APL is specified for device types LU2 and LU3. The 3270 APL/Text Character Set differs from the 3270 Data Analysis/APL Character Set in that the 2-character sequences begin with a graphic escape character (X'08') instead of a start field (SF) order (X'1D'). Also, the character code points for APL symbols may not be the same in both character sets.

Use the CHARSET statement to specify which 3270 character set is used for subsequent data input during message generation. CHARSET APL specifies that the APL/Text Character Set is used. CHARSET FIELD or CHARSET with no operand specifies that the standard EBCDIC character set is used. The following example shows you how to use the CHARSET statement in a message deck.

```
APLTEXT  MSGTXT
         CHARSET   APL            Enter data from APL character set.
         TEXT      ('DF')         APL comment symbol.
*                                 Return to standard EBCDIC character set.
         CHARSET
         TEXT      (  X = 4 1 6)
         NL
         TEXT      (X)            Variable X.
         CHARSET   APL
         TEXT      ('9F')         APL (left) pointer symbol.
         CHARSET   FIELD
         TEXT      (4 1 6)        Vector 4 1 6 assigned to X.
         ENTER
         ENDTXT
```

Simulating 3270 extended functions

WSim supports 3270 extended functions for device types LU2 and LU3. Extended highlighting, extended color, Programmed Symbols (PS), 12-bit and 14-bit buffer addressing, multiple partitions, and structured fields are supported by the EXTFUN, HIGHLITE, COLOR, PS, MAXNOPTN, and MAXPTNSZ operands on the device statements in your network definition. You can simulate the 3270 extended function operator interface by using the message generation statements COLOR, HIGHLITE, CHARSET, JUMP, CLEARPTN, and SCROLL. Descriptions of each of the 3270 extended functions follow.

Extended color

WSim provides seven-color display and four-color printer simulation when you specify COLOR=MULTI in the definition of device types LU2 and LU3. Use the COLOR message generation statement to enter data into the display buffer with the associated character attributes set to reflect the color of the displayed characters. When the current inbound reply mode state does not allow color selection or when COLOR=MULTI is not specified for the simulated device, log message ITP441I is generated. WSim generates a color query response structured field in response to a read partition query that reflects the seven-color display or four-color printer support.

The following example shows a sample message deck for a color terminal or printer.

```
COLOR  MSGTXT COUNT=1
*
*           Color Scripting Example
*
       COLOR  RED
       TEXT   (RED CHARACTERS)
       COLOR  BLUE
       TEXT   (BLUE CHARACTERS)
       COLOR  FIELD
       TEXT   (CHARACTERS OF FIELD-DEFINED COLOR)
       ENTER
       ENDTXT
```

Extended highlighting

WSim provides display and printer extended highlighting simulation when you specify HIGHLITE=YES in the definition of device types LU2 and LU3. Use the HIGHLITE message generation statement to enter data into the display buffer with the associated character attributes set to reflect the highlight of the displayed characters. When the current inbound reply mode state does not allow color selection or when COLOR=MULTI is not specified for the simulated device, log message ITP441I is generated. WSim generates a highlight query response structured field in response to a read partition query that reflects the device highlighting support.

The following example shows a sample message deck using extended highlighting.

```
HIGHLITE MSGTXT   COUNT=1
*
*       Highlighting Scripting Example
*
         HIGHLITE BLINK
         TEXT     (BLINKING CHARACTERS)
         HIGHLITE REVERSE
         TEXT     (REVERSE IMAGE CHARACTERS)
         HIGHLITE FIELD
         TEXT     (CHARACTERS WITH FIELD-DEFINED HIGHLIGHTING)
         ENTER
         ENDTXT
```

Programmed Symbols (PS)

WSim provides display and printer Programmed Symbol (PS) simulation on the PS operand for device types LU2 and LU3. You can define one to six programmed symbols as either single or triple plane. Upon receiving a load PS structured field, WSim saves the PS name value, and it becomes the character set attribute value sent and received with SA, SFE, and MF orders. WSim ignores the load PS structured field character cell data because no characters are displayed. The output-only bit of the load PS structured field is saved to inhibit operator (WSim script) selection of the PS. WSim generates a character set query response structured field in response to a read partition query that reflects the device character set and PS support. You can code the CCSIZE operand to specify the character cell size values in the query response.

Use the CHARSET message generation statement to enter data into the display buffer with the associated character attributes set to reflect the PS character set of the displayed characters. Log message ITP441I is generated when the current inbound reply mode state disallows operator (WSim script) PS character set selection or when the requested PS is not specified for the simulated device.

The following example shows a sample message deck using programmed symbols.

```
CHARSET  MSGTXT  COUNT=1
*
*           Character Set Scripting Example
*
         CHARSET PSA
         TEXT    (CHARACTERS FROM PS 1 CHARACTER SET)
         CHARSET PSF
         TEXT    (CHARACTERS FROM PS 6 CHARACTER SET)
         CHARSET FIELD
         TEXT    (CHARACTERS FROM FIELD-DEFINED CHARACTER SET)
         ENTER
         ENDTXT
```

Multiple partitions and scrolling

The 3290 display terminal with the Enhanced Function or Multiple Partitions and Scrolling feature support screen partitioning. The display area can be divided into as many as 16 rectangular viewports for the 3290. Each viewport is associated with a particular partition and displays all or portions of the data from that partition. The operator controls data entry into the partitions using the Jump, Clear Partition, and Scroll keys.

WSim maintains separate buffers for each partition created and supports all of the structured fields associated with partitioning. WSim generates a partitioning query response structured field in response to a read partition query that reflects the partitioning support.

Use the message generation statements JUMP, CLEARPTN, and SCROLL to control the simulated partitions. Use the CURSOR, DATASAVE, IF, and SELECT statements with the (*row*,*col*) operand to reference the simulated partitions. All references to the display buffer using specifications other than (*row*,*col*) reference data in the active partition. Log message ITP441I is generated whenever a JUMP, CLEARPTN, or SCROLL statement operation cannot be performed.

12-Bit and 14-Bit buffer addressing

WSim supports display and printer 12-bit and 14-bit buffer addressing when you specify EXTFUN=YES for device types LU2 and LU3. Data streams received by WSim can include 12-bit or 14-bit buffer addresses. WSim generates 12-bit buffer addresses unless the current display buffer size is greater than 4096 bytes, which forces 14-bit buffer addresses to be generated. WSim rejects any buffer address with bits 0 and 1 of the first byte set to B'10'.

Structured fields

WSim supports the Write Structured Field command (WSF) when you specify EXTFUN=YES for device types LU2 and LU3. Data received with the WSF command is assumed to be one or more structured fields in the length-parameters structured field data format. When the structured field length (first 2 bytes of the structured field) is zero, the structured field is considered to be the last within the chain of data being received, and all the data received until the end of the data chain is processed as part of the last structured field.

WSim supports the following structured fields that are associated with the 3270 extended functions:

- Load Programmed Symbols
- Read Partition - Query
- Set Reply Mode.

WSim also supports the following features representing recent 327X enhancements:

- Summary Query Reply
- Query List

WSim support for these recent 327X enhancements is explained below:

Summary query reply: The Summary Query Reply is generated by newer 3270 cluster controllers in response to a **Read Partition Query structured** field to indicate support of Query List and to list the types of query replies that are supported. WSim automatically generates the Summary Query Reply in response to a **Read Partition Query structured** field.

Query list: The Query List type parameter (byte 5 = X'03') in a **Read Partition structured** field enables a host application to request a subset of the display function query replies by specifying the type of query list replies wanted. WSim accepts the Query List type parameter of the **Read Partition structured** field and returns the requested query replies as specified.

Testing field and character attributes

You can test for standard field, extended field, and character attributes by using the $ATTR$ data field option to save the attributes at a specific screen location. You can use this option to identify attributes such as color, highlighting, character set, field validation, field outlining, and SO/SI enabled.

The $ATTR$ data field option generates up to eleven EBCDIC characters that provide attribute information. You can save these characters with the DATASAVE statement and test them with IF statements to determine the attribute values for a specified screen location.

$ATTR$ has the following format:

```
$ATTR,location,length$
```

location can be B+*value* or B-*value*, indicating a location relative to the beginning or end of the screen image buffer; C+*value* or C-*value*, indicating a location relative to the position of the cursor; or *row,col*, indicating a specific row and column on the screen. *length* can be 1 - 11, indicating the number of EBCDIC characters to be generated that will provide attribute information.

Table 12 indicates the values of the eleven EBCDIC characters generated by the $ATTR$ data field option:

Table 12. Values generated by the $ATTR$ data field option

Byte	Defines	Explanation	
1	Attribute Information	**E**	Location specification error.
		0	Unformatted screen, default field and actual character attribute values will be generated.
		1	Formatted screen, no field attribute defined at specified location, actual field and character attribute values will be generated.
		2	Formatted screen, field attribute defined at specified location, actual field and default character attribute values will be generated.
2	Standard Field Attribute	**X**	EBCDIC translated standard field attribute character when byte 1 is set to 1 or 2. A blank will be generated when byte 1 is set to E or 0.
3	Highlighting Field Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, normal highlighting
		1	Blinking
		2	Reverse image
		3	Underlined

Table 12. Values generated by the $ATTR$ data field option (continued)

Byte	Defines	Explanation	
4	Highlighting Character Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, normal highlighting
		1	Blinking
		2	Reverse image
		3	Underlined
5	Color Field Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, normal color
		1	Blue
		2	Red
		3	Pink
		4	Green
		5	Turquoise
		6	Yellow
		7	White
6	Color Character Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, field defined
		1	Blue
		2	Red
		3	Pink
		4	Green
		5	Turquoise
		6	Yellow
		7	White
7	Character Set Field Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, base character set
		1	APL
		2	PSA
		3	PSB
		4	PSC
		5	PSD
		6	PSE
		7	PSF
		8	DBCS

Table 12. Values generated by the $ATTR$ data field option (continued)

Byte	Defines	Explanation	
8	Character Set Character Attribute	**N**	No extended attribute buffer or byte 1 set to E
		0	Default, field defined
		1	APL
		2	PSA
		3	PSB
		4	PSC
		5	PSD
		6	PSE
		7	PSF
		8	DBCS
9	Field Validation Field Attribute	**N**	Field validation not supported or byte 1 set to E
		0	Default, no field validation specified
		1	Trigger
		2	Mandatory enter
		3	Trigger and mandatory enter
		4	Mandatory fill
		5	Mandatory fill and trigger
		6	Mandatory fill and mandatory enter
		7	Mandatory fill, mandatory enter, and trigger
10	Field Outlining Field Attribute	**N**	Field outlining not supported or byte 1 set to E
		0–9, A–F	Field outlining settings
11	SO/SI Operator Creation Attribute	**N**	DBCS not supported or byte 1 set to E
		0	SO/SI creation by operator not enabled
		1	SO/SI creation by operator enabled

The following example shows how you can use the $ATTR$ data field option to test the attributes at the current position of the cursor:

```
SAVE1  DATASAVE   AREA=U+0,TEXT=($ATTR,C+0,9$)
IF1    IF         LOC=U+2,TEXT=(1),WHEN=IMMED,ELSE=B-ERROR
IF2    IF         LOC=U+4,TEXT=(2),WHEN=IMMED,ELSE=B-ERROR
WTO1   WTO        (THE CURSOR IS CURRENTLY IN A BLINKING RED FIELD)
```

Logging the display image for formatting by the Loglist Utility

The 3270 display and printer buffers maintained by WSim can be written to the log data set and later formatted by the Loglist Utility into screen images as the 3270 display operator would see them. Use the LOGDSPLY operand on the NTWRK statement in your network definition and the LOG statement with the DISPLAY operand in your message generation decks to control the writing of the display buffers. See the *WSim Script Guide and Reference* for information about using these statements. *WSim Utilities Guide* provides information about using the Loglist Utility.

Display Monitor Facility

The Display Monitor Facility enables you to view simulated 3270 display images on a monitoring 3270 display during a simulation run. See *WSim User's Guide* for a detailed description of the Display Monitor Facility.

Simulating DBCS data entry for simulated 3270 DBCS terminals

WSim supports simulation of 3270 DBCS terminals that send and receive messages with DBCS data. You can enter DBCS data in scripts by either entering the DBCS data directly from a 3270 DBCS display into the script or by using the DATASAVE DBCS functions such as SB2MDBCS to create DBCS data at message generation time. See “Converting data in a save or user area with the DATASAVE statement” on page 133 for more information about converting DBCS data.

When you enter DBCS data into the script from a 3270 DBCS display, the DBCS data is identified by wrapping Shift-Out (SO) (X'0E') and Shift-In (SI) (X'0F') characters around the DBCS data. DBCS data entered in this manner is called ***literal text DBCS data***. You can enter literal text DBCS data as text data and within comments. DBCS data identified using SO and SI characters is also referred to as a “DBCS subfield” or DBCS data in a mixed string.

In the following examples and discussion, the SO character is represented using a "<", the SI character is represented using a ">", and the first byte of each DBCS character, which is referred to as the ward byte, is represented using a "." character.

The following examples enter DBCS data into the simulated screen. The DATASAVE SB2MDBCS function creates a mixed string containing a DBCS subfield with ward 42 (EBCDIC) DBCS data from the input string.

```
TEXT (<.A.B.C>)           DBCS data entered using 3270 DBCS display

DATASAVE AREA=1,          save area 1 = <.A.B.C>
         FUNCTION=SB2MDBCS, convert SBCS to mixed string ward 42 DBCS
         TEXT=(ABC)       SBCS data to be converted to DBCS
TEXT ($RECALL,1$)         Ward 42 (EBCDIC) DBCS data via DATASAVE
```

You can code literal text DBCS data and SBCS data within the same mixed string, as shown in the following examples.

```
TEXT (<.D.B.C.S> and SBCS)  Mixed string from 3270 DBCS display

TEXT ($RECALL,1$ and SBCS)  Mixed string using recall from save area
```

The message generation process depends on the SO and SI characters to identify DBCS data. Once the message generation process detects an SO character, all the data following the SO character is assumed to be DBCS data until an SI character is processed or the TEXT statement is ended. Extra SO and SI characters encountered in the TEXT data are ignored.

The only exception to this rule is for a 3270 DBCS field. When you first start entering text data into a DBCS field, the data is accepted and assumed to be DBCS data.

The simulated screen is updated based on the type of data (SBCS or DBCS) entered and the format of the simulated screen or field the data is being entered into.

IBM 5250 Display System

WSim simulates an IBM 5250 display terminal as a logical unit Type 7 (device type LU7). WSim simulates an IBM 5250 printer as a logical unit Type 4 (device type LU4). WSim maintains a screen image buffer and a format table for each terminal. The buffer can be modified by messages generated by WSim and by commands and orders received from the system under test. The format table can be modified by commands and orders received from system under test.

The format table consists of header information that is supplied by the start of header (SOH) order, and one entry for each field defined in the screen image buffer. Each entry in the format table consists of at least a starting and ending address that defines the field limits on the screen image, and a field format word (FFW) that contains information pertinent to that field, such as type of field, MDT bit, and field specifications that govern how that field is to be processed.

Data generated by WSim is placed into a terminal buffer under control of a Read command that must be received from the system under test before message generation can be entered. WSim automatically breaks the transmitted data into RU chain elements.

When Read commands for a terminal are received in the data stream, they are placed in a queue. When a poll is received and all conditions for entering message generation have been met, a message is generated into the terminal buffer under control of the FFW for the field being written into.

A terminal is placed in Normal Lock State when a message is generated or when certain commands are received. It is placed in Normal Unlock State when its message delay expires or when a Read command is received with the unlock keyboard bit set to B'1'.

Message generation

The following four types of messages can be generated for a 5250 display terminal:

1. Sense responses

   SNA sense responses are generated in response to the following:

   - Invalid commands
   - Orders
   - Parameter data passed with commands or orders
   - Failure of data termination in conjunction with chaining.
2. SIGNAL requests

   SNA SIGNAL requests are generated in response to a poll when an error condition is detected during message generation or is specified on the HELP statement with a CODE operand value. The error code is set into the error row for the terminal and then sent to the system under test in the sense code area of the SIGNAL command. You can code a HELP statement to specify an error code that is to be generated by the simulated terminal. See "Simulating errors in a 5250 terminal" on page 244 for more information about generating errors for 5250 display terminals.

3. Logical unit status requests

   Logical unit status (LUSTAT) requests are generated in response to a poll after a message has been generated into the device buffer and an error has been detected during a Read Input Fields or Read Modified Fields command. LUSTAT requests are also generated on an LU-LU or LU-SSCP session to notify the system under test that a previous request, which was rejected due to the state of the terminal, can now be serviced. An LUSTAT request can be generated, for example, if resequencing is specified and the first field to be transmitted to the host does not exist.

4. Normal messages

   Normal messages are generated under the following conditions:
   - When a device is polled and a Read command has been received
   - When no delay is in effect
   - When the logical WAIT indicator is reset
   - When the INPUT INHIBITED indicator is reset
   - When no status is pending.

   Text messages to be generated are treated as data keyed in at a 5250 display terminal. They are used to modify the screen image buffer according to the cursor position and the field specifications supplied for each field entry within the format table. Data that cannot be entered at a 5250 keyboard should not be coded for these messages. After a message has been generated according to the TEXT statement specifications, WSim interrogates the terminal buffer and constructs the data to be transmitted, including headers, addresses, and the AID byte. You can use the CLEAR, CMD*n*, ENTER, HELP, PRINT, ROLLDOWN, ROLLUP, or SELECT statements to set the AID byte.

When WSim processes a SELECT statement for a 5250 display terminal, one of the following actions is taken if the specified row or column address is within a selectable field:

- If the field has not been modified, the first byte of the field is set to ">", and the master and field MDT bits are set ON.
- If the field has been modified, the first byte of the field is set to "?", and the field MDT is set OFF.
- If the field is specified as an auto enter field, an enter function is attempted. The light pen enter AID and data are returned to the test system only when the field MDT is turned ON.

Logic testing

You can use the IF statement to perform logic tests for 5250 terminals on the screen image buffer data or the incoming or outgoing data stream.

You can specify a logic test on the screen image buffer for the terminal by coding the B+, B-, C+, C-, or (*row*,*col*) location options on an IF statement. This type of logic test operates on the data as it would be displayed at a real terminal. All logic tests on a screen image buffer are performed after the buffer has been modified according to the message generated or received. If a received message contains invalid commands or orders, the data received before the invalid command or order is processed and can modify the screen image buffer, but no data following the invalid command or order is processed.

You can specify a logic test on an incoming or outgoing data stream, including headers, commands, and orders, by coding the D+, TH+, RH+, or RU+ location value on an IF statement. For more information about coding these values on the IF statement, see “Coding the LOC operand” on page 171.

Simulating errors in a 5250 terminal

Use the HELP statement to cause a simulated 5250 terminal to transmit an error code to the system under test. If the 5250 display is in error state when the HELP statement is processed, the data in columns 2 through 5 of the display error line are sent to the host as the sense bytes of a SIGNAL request. If the 5250 display is not in error state, the value specified by the CODE operand is moved into the error line and sent to the host in a SIGNAL request. If the code operand is omitted or if it specifies a value of 0000 and the display is not in error state, the cursor address and HELP AID are sent to the system under test.

Part 4. Using message generation decks

Chapter 19. Integrating decks with network definitions

The preceding chapters described the message generation statements you can code to create message generation decks. Before you can begin a simulation, however, you must provideWSim with a complete script by integrating your decks with a network definition. As discussed in Part 1, "Defining WSim networks," on page 1, you code network definition statements to define the devices on the simulated network, and describe the relationship between the resources and the system under test.

To create a script, you code specific network definition statements and operands that integrate your decks with the network definition. These statements and operands associate decks with simulated resources, define the processing order for each deck, establish how often WSim processes each deck, and name decks that WSim uses for error recovery.

This chapter discusses the following network definition statements and operands that you can code to integrate your decks with the network definition:
- PATH statement
- PATH operand on statements such as the TP, DEV, and LU statements
- CYCLIC operand on the PATH statement
- DIST statement
- FRSTTXT operand on statements such as the TP, DEV, and LU statements
- INCLUDE statement
- ATRDECK and ATRABORT operands on statements such as the DEV and LU statements
- SCAN operand on the NTWRK statement.

Note: For a complete list of the statements on which the PATH, FRSTTXT, ATRDECK, and ATRABORT operands can be coded, see the *WSim Script Guide and Reference*.

After you integrate your decks and the network definition, you must decide how to store your script. Both the Preprocessor and the ITPSYSIN utility program store scripts in the predefined data sets WSim uses when running the simulation. If you use the Preprocessor, however, you can check the syntax of your scripts before you store them.

This chapter describes how to integrate your decks with a network definition and how to store your scripts. A sample script is provided to help you understand the integration process. This script is also used in Chapter 20, "Analyzing simulation results," on page 259, which provides examples of reports you can use to analyze your simulations.

Note: If you use STL to create your message generation decks, you can include the network definition in your STL input. This is an easy way to integrate your decks with your network definition.

Selecting message decks in the network definition

To integrate your decks with the network definition,WSim provides several network definition statements and operands that select message generation decks for processing. The following sections provide information about each statement and operand and describe how they affect the message generation process.

Selecting decks with the PATH statement

With the PATH network definition statement, you can specify a path, that is, a series of message generation decks listed in a specific order. You can define any number of paths for each network definition:

```
name  PATH  deck,deck,...     Specifies the sequence in which message
*                             generation decks are processed during a
*                             simulation.
```

You can identify each path uniquely by coding 1 - 8 alphanumeric characters in the PATH statement's name field. When you code *deck*,*deck*..., you specify the names of message generation decks in the order in which you wantWSim to reference the decks during a simulation:

```
PATH1  PATH  DECK1,DECK4,DECK2   References DECK1, DECK4, then DECK2.
```

During message generation, WSim selects decks for processing in the order indicated by the PATH statement.

Note: The PATH statement names only the main decks to be processed by a simulated resource. Do not code the names of decks that are called or branched to from within a deck.

Assigning paths to simulated resources

After you define paths on the PATH statement, code the PATH operand to assign a sequence of paths to each simulated resource on the network. For example, to specify a resource reference PATH1 and then PATH3, code the PATH statements and the PATH operand as shown in the following example:

```
PATH1  PATH   DECK1,DECK2          References DECK1, then DECK2.
PATH2  PATH   DECK2,DECK1          References DECK2, then DECK1.
PATH3  PATH   DECK1,DECK4,DECK2    References DECK1, DECK4, then DECK2.
*
DEV1   DEV    PATH=(PATH1,PATH3)   Specifies path selection for DEV1.
```

During message generation, DEV1 first processes the decks named by PATH1 in the specified order; then it processes the decks named by PATH3 in the specified order. WSim processes the decks in each path as complete entities, beginning with the MSGTXT statement and continuing until the ENDTXT statement is processed. After completing the last deck in a path, WSim selects the first deck from the next path assigned to the terminal. If the last deck has been selected from the last assigned path,WSim selects the first deck from the first path, repeating the sequence of paths and decks.

If you do not code the PATH operand for a simulated resource, WSim selects path entries sequentially from all of the PATH statements defined in the network. In this way, WSim completes the first path in sequence and then the second path, continuing until all paths have been processed.

Note: For a complete list of the statements on which you can code the PATH operand, see the *WSim Script Guide and Reference*.

Selecting paths in a cycle

You can control how WSim selects assigned decks by coding the CYCLIC operand on the PATH statement. The CYCLIC operand determines how each simulated resource selects decks within the assigned path. For example, any number of simulated resources can reference the same deck within a PATH statement. Normally, each resource begins with the first deck named in the path and selects each deck in order as it proceeds through the path. When you code CYCLIC=YES, however, the next deck is maintained on a PATH rather than a terminal basis. Each terminal begins processing the next deck for the PATH.

WSim remembers the next deck for each PATH statement that specifies CYCLIC=YES.WSim begins with the first deck on the PATH statement and processes the next deck whenever a terminal references that statement. If a terminal references a cyclic path for a deck prior to or equal to the one currently being completed by the terminal, the terminal selects the first available entry from the next PATH statement. If there is only one PATH statement, the terminal selects the first available entry from the same PATH statement.

The following example shows the coding for cyclic path selection:

```
PATH1  PATH  DECK1,DECK2,DECK3       WSim processes DECK1 through DECK3.
             CYCLIC=YES
*
DEV1   DEV   PATH=(PATH1)            Specifies the path for DEV1.
DEV2   DEV   PATH=(PATH1)            Specifies the path for DEV2.
DEV3   DEV   PATH=(PATH1)            Specifies the path for DEV3.
DEV4   DEV   PATH=(PATH1)            Specifies the path for DEV4.
```

In this example, the first terminal referencing PATH1 selects the first deck (DECK1) in the path, the second terminal selects the second deck (DECK2), and so on. The fourth terminal would select the first deck (DECK1) again. When the last deck in PATH1 is selected, the next terminal begins again with the first deck in the path.

Selecting paths with a probability distribution

During a simulation, you might not want a resource simply to repeat a sequence of message generation decks specified on a path. With the DIST network definition statement, you can assign a weighted value to each deck in a path. As shown in the following example,WSim uses these weighted values to select a deck from a probability distribution each time the path is referenced:

```
name  DIST  weight,weight,...     Defines a probability distribution for
*                                 selecting decks from a PATH statement.
```

You associate a DIST statement with a PATH statement by assigning it the same *name* as the PATH statement. The numbers you code for *weight*,*weight*... assign relative weights to corresponding entries on the PATH statement.WSim divides each weight by the total of the weights to obtain fractional values for each corresponding PATH entry. Therefore, each weight is a proportion of the total of the weights on the statement.

Note: If the weights specified on the DIST statement total 100, the weights represent percentages.

The fractional values represent the probability that a particular message generation deck will be chosen from the corresponding PATH statement. For example, when WSim selects a PATH statement with an associated DIST statement, one entry from that path is selected according to the specified distribution before proceeding to the next path for the terminal.

The following example shows PATH and DIST statements with corresponding names. If WSim processes PATH1 100 times, on the average it selects DECK1 20 times and DECK2 80 times:

```
PATH1  PATH  (DECK1,DECK2)  Specifies selection of message decks.
        .
        .                   Network definition statements.
        .
PATH1  DIST  20,80          Defines probability distribution for selecting
*                           decks from PATH statement named PATH1.
```

Note: You can also specify how many times WSim repeats a deck named on a PATH statement by coding the COUNT operand on the MSGTXT message generation statement. When you code COUNT=*integer*, WSim reduces *integer* by one each time it processes the deck. When the count reaches zero,WSim moves on to the next deck named in the path. For more information about the COUNT operand, see the *WSim Script Guide and Reference*.

Specifying the first message generation deck

When you specify a path in the network definition, each resource associated with that path can repeat the sequence of decks many times during the simulation. For some simulations, however, you might not want each deck processed repeatedly. To specify a deck that is used only once at the beginning of a simulation, name that deck on the FRSTTXT operand. During the simulation, WSim selects the named deck as the first deck for the terminal and does not process it again unless the deck is also specified on a PATH statement. Typically, the FRSTTXT operand specifies a message generation deck containing a logon sequence for the terminal.

The following example shows a DEV statement coded with the PATH and FRSTTXT operands.WSim first uses the deck specified by the FRSTTXT operand, then follows the sequence of paths specified by the PATH operand:

```
PATH1  PATH  DECK2,DECK3          Reference DECK2, then DECK3.
PATH2  PATH  DECK3,DECK2          References DECK3, then DECK2.
*
DEV1   DEV   PATH=(PATH2,PATH1),  DEV1 references PATH2, then PATH1.
             FRSTTXT=DECK1        Specifies the first deck used by DEV1.
```

Note: For a complete list of the statements on which you can code the FRSTTXT operand, see the *WSim Script Guide and Reference*.

Including decks in a script

When WSim starts a simulation, it automatically loads all the message generation decks that are referenced within a script. Therefore, you must reference all the message generation decks you want to use during the simulation somewhere within your script. For example, you might want to include a deck that can be referenced with the A (Alter) operator command at the operator's choice. To make decks available that are not referenced in the network definition or in a message generation deck, you can code the INCLUDE network definition statement.

As shown in the following example, the INCLUDE statement enables you to include as many decks in your script as you want:

```
INC1  INCLUDE  DECKA,DECKB,DECKC  Specifies decks to be included that may
*                                 not be referenced anywhere else in the
*                                 script.
```

Specifying decks for error recovery

During a simulation, a terminal might become inactive if one of the decks it uses contains an error or if an unexpected response is received from the system under test. WSim provides an Automatic Terminal Recovery (ATR) feature that attempts to recover automatically any simulated terminal that is inactive for a certain period of time. Whether to attempt recovery and how long to wait after a terminal is detected as inactive before attempting recovery is controlled by the third value specified on the SCAN operand on the NTWRK statement.

The ATRDECK operand names an error recovery deck for the terminal that performs a sequence of actions to return the terminal to a known state. For example, you could name a recovery deck that would log the terminal off the application and log it back on.

With the ATRABORT operand, you can specify how WSim continues message generation after processing the recovery deck. You can code one of the following values for ATRABORT:

NONE	Specifies that message generation continues in the same deck that was being used when the inactivity occurred.
DECK	Specifies that the current deck is aborted and message generation continues with the next deck in the current path.
PATH	Specifies that the current path is aborted and message generation continues with the next path specified for the terminal.

The following example shows how to code the ATRDECK and ATRABORT operands:

```
DEV1  DEV  ATRDECK=rname,  Specifies an error recovery deck.
           ATRABORT=DECK   Abort the current deck; message generation
*                          continues with the next deck in the path.
```

Note: For a complete list of the statements on which you can code the ATRDECK and ATRABORT operands, see the *WSim Script Guide and Reference*.

You can activate automatic terminal recovery with the SCAN operand on the NTWRK statement or the A (Alter) operator command. The following example shows the required syntax for the SCAN operand:

```
NET1  NTWRK  SCAN=(x,y,z,sname)   Activates automatic terminal recovery.
```

With the SCAN operand, you can specify a deck to be used as a replacement for the defective deck. Each time the terminal uses the path containing the defective deck, WSim substitutes the replacement deck for the defective deck.

You can define automatic terminal recovery by coding the following values on the SCAN operand:

x	Specifies the amount of time between inactive terminal reports.
y	Specifies the amount of time that may elapse before a terminal is listed inactive.
z	Specifies the number of minutes WSim delays between detecting an inactive terminal and invoking automatic terminal recovery.
sname	Specifies the name of a message generation deck WSim uses as a substitute for decks that cause terminals to enter automatic terminal recovery.

For information about automatic terminal recovery with the A (Alter) operator command, see *WSim User's Guide*.

Creating a script

The following sections present a sample script to help you understand how message generation decks and the network definition are integrated:

- "Understanding the network definition" presents the network definition and an explanation of the statements coded to define the network.
- "Understanding the sample message generation decks" on page 254 presents four message generation decks that enable you to simulate logging on TSO, accessing ISPF, clearing the screen, and then logging off TSO.
- "Understanding the sample script" on page 255 integrates the network definition and the message generation decks to create a script and provides a step-by-step description of how WSim processes the script.

Understanding the network definition

The network definition presented in the following example defines a simulated network that includes a logical unit representing a VTAM application. The following example provides an illustration of the simulated network's logical configuration.

```
SAMP1    NTWRK     HEAD='Sample Network 1',  Heading for interval reports.
                   UTI=100,                  Network user time interval.
                   MSGTRACE=YES,             Logs MTRC records.
                   BUFSIZE=5000              5000-byte buffer for LU.
*
*   Coding for a network named SAMP1:
*
*
NETIF    IF        LOC=RU+0,TEXT=(***),      Defines logic test for messages
                   THEN=CCLEAR,WHEN=IN,      received from the system under
                   SCAN=YES                  test.
SIMPLE   PATH      INITSESS,LOGOFF           Specifies the order in which
*                                            WSim processes message decks
*                                            named INITSESS and LOGOFF.
WSIMAPPL VTAMAPPL
*                                            Defines VTAM application named
*                                            WSIMAPPL.
SLU      LU        PATH=(SIMPLE),            Specifies a path named SIMPLE.
                   MAXSESS=(0,001),          Defines one secondary half-
                   INIT=SEC,                 session for SLU.  Specifies
                   LUTYPE=LU2,               LU type and the name of a VTAM
                   DLOGMOD=D4A32782,         logon mode table entry, delays
                   THKTIME=UNLOCK,           start of intermessage delays,
                   LOGDSPLY=BOTH,            writes display buffers to the
                   RSTATS=YES                log data set, and accumulates
*                                            response-time statistics.
```

Each of the following statements coded in the sample network definition combines to create the simulated network:

- NTWRK
- IF
- PATH
- VTAMAPPL
- LU.

The operands coded on the sample NTWRK statement determine the following network characteristics:

HEAD='Sample Network 1'
Specifies the heading WSim prints on each interval report for the network.

UTI=100
Specifies a user time interval for the network.WSim multiplies this interval by a delay value to determine the intermessage delay.

MSGTRACE=YES
Specifies that trace records be written to the log data set.

BUFSIZE=5000
Specifies a 5000-byte terminal buffer for network logical units.

The IF statement defines a logic test to be performed when messages are received from the system under text:

LOC=RU+0
Specifies that WSim starts the comparison at the first byte in the request/response unit.

TEXT=(***)
Specifies that WSim tests for the characters "***".

THEN=CCLEAR
Specifies that if the test condition is met, WSim calls the deck named CLEAR and continues message generation.

WHEN=IN
Specifies that WSim performs the test each time a message is received from the system under test.

SCAN=YES
Specifies that WSim scans the data sequentially, character by character, beginning at the cursor location specified by the LOC operand.

Each time a message is received from the system under test, WSim begins testing for the characters *** at a zero offset from the start of the request unit (RU+0). If the characters are not found, message generation continues. If the characters are found, WSim calls the CLEAR deck and continues message generation with that deck.

The PATH statement specifies the order in which WSim processes the message generation decks. In the sample network, the PATH statement named SIMPLE specifies the following order: INITSESS and then LOGOFF.

The VTAMAPPL statement defines a VTAM application named WSIMAPPL. The operands on the LU statement define the simulated logical unit named SLU:

PATH=(SIMPLE)
Specifies that WSim processes the path named SIMPLE for the logical unit.

MAXSESS=(0,001)
Specifies a maximum of one concurrent secondary half-session for the logical unit.

INIT=SEC
Specifies that a secondary logical unit initiates the session.

LUTYPE=LU2
Specifies the type of logical unit being simulated for the SNA half-session.

DLOGMOD=D4A32782
Specifies the name of a VTAM logon mode table (MODETAB) entry for this logical unit.

THKTIME=UNLOCK
Delays the start of intermessage delays for this logical unit.

LOGDSPLY=BOTH
Specifies that display buffers are written to the log data set.

Figure 23 illustrates the logical configuration simulated by the sample network definition.

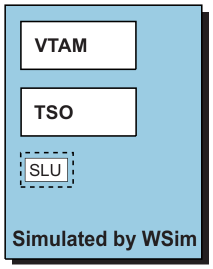


Figure 23. Logical configuration for sample network definition

Understanding the sample message generation decks

This section presents the other part of the sample script: the message generation decks. The four message generation decks that follow enable you to simulate logging on TSO, accessing ISPF, clearing the screen, and then logging off TSO.

The following example illustrates the first message generation deck, which is named INITSESS. INITSESS initiates the message generation session and simulates the logical unit SLU logging on to TSO. In this deck, the TSO resource name is TSO01. The user ID is ID02, and the password is PW02.

```
INITSESS MSGTXT
*                                         Beginning of INITSESS.
WTO1     WTO       (STARTING $MSGTXTID$)  Sends message to console.
CMND1    CMND      COMMAND=INIT,          Initiates session.
                   RESOURCE=TSO01
1        IF        LOC=RU+0,              Tests for characters ENTER
                   TEXT=(ENTER USERID),   USERID.
                   SCAN=YES,THEN=CONT,
                   ELSE=WAIT
WAIT1    WAIT
*                                         Interrupts message generation.
MSG1     TEXT      (ID02)                 Enters user ID ID02.
WTO2     WTO       (Logging on TSO as ),  Sends message to console.
                   (ID02)
ENTER1   ENTER                            Sets ENTER AID byte.
2        IF        LOC=RU+0,              Tests for characters ENTER
                   TEXT=(ENTER LOGON),    LOGON.
                   SCAN=YES,THEN=CONT
WAIT2    WAIT
*                                         Interrupts message generation.
MSG2     TEXT      (PW02)                 Enters user password PW02.
ENTER2   ENTER                            Sets ENTER AID byte.
3        IF        LOC=RU+0,              Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),      PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
4        IF        LOC=RU+0,TEXT=(READY), Tests for characters READY.
```

```
                   SCAN=YES,THEN=CISPFDECK
WAIT3    WAIT
*                                         Interrupts message generation.
WTO3     WTO       (Logged on and ),      Sends message to console.
                   (received ISPF Primary ),
                   (Menu)
         ENDTXT                           End of INITSESS.
```

As shown in the example below, the second deck, named ISPFDECK, simulates SLU accessing ISPF and then waiting for the ISPF primary menu to be returned before continuing message generation.

```
ISPFDECK MSGTXT
*                                         Beginning of ISPFDECK.
WTO1     WTO       (STARTING $MSGTXTID$)  Sends message to console.
MSG1     TEXT      (ISPF)                 Enters ISPF.
ENTER1   ENTER                            Sets ENTER AID byte.
1        IF        LOC=RU+0,              Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),      PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
WAIT1    WAIT
*                                         Interrupts message generation.
WTO2     WTO       (Logged on and ),      Sends message to console.
                   (received ),
                   (ISPF Primary Menu)
         ENDTXT                           End of ISPFDECK.
```

The following example illustrates LOGOFF, which simulates SLU exiting ISPF and logging off TSO.

```
LOGOFF   MSGTXT
*                                         Beginning of LOGOFF.
WTO1     WTO       (Starting $MSGTXTID$)  Sends message to console.
MSG1     TEXT      (X)                    Enter text to exit ISPF.
1        IF        LOC=RU+0,TEXT=(READY), Tests for characters READY.
                   SCAN=YES,THEN=CONT
WAIT1    WAIT
*                                         Interrupts message generation.
MSG2     TEXT      (LOGOFF)               Enter text LOGOFF.
WTO2     WTO       (Logged off TSO)       Sends message to console.
WAIT2    WAIT
*                                         Interrupts message generation.
OPCMND1  OPCMND    (ZEND)                 Specifies operator command ZEND.
         ENDTXT                           End of LOGOFF.
```

As shown in the following example, deck CLEAR clears the screen when the characters "***" appear in the request unit (RU+0).

```
CLEAR    MSGTXT
*                                         Beginning of CLEAR.
CLEAR1   CLEAR                            Clears the screen.
         ENDTXT                           End of CLEAR.
```

Understanding the sample script

Now that you understand the network definition and the message generation decks separately, the following example presents the complete sample script.

```
SAMP1    NTWRK     HEAD='Sample Network 1',  Heading for interval reports.
                   UTI=100,               Network user time interval.
                   MSGTRACE=YES,          Logs MTRC records.
                   BUFSIZE=5000           5000-byte buffer for LU.
*
*                       Sample WSim Script
*   This script defines a simple network in which WSim simulates a
*   single LU representing a VTAM application.  WSim uses three message
*   generation decks to simulate logging a logical unit on to TSO and
*   then off again.
*
*   Coding for a network named SAMP1:
*
NETIF    IF        LOC=RU+0,TEXT=(***),      Defines logic test for messages
                   THEN=CCLEAR,WHEN=IN,      received from the system under
                   SCAN=YES                  test.
*
SIMPLE   PATH      INITSESS,LOGOFF           Specifies the order in which
*                                            WSim processes message decks
*                                            named INITSESS and LOGOFF.
*
WSIMAPPL VTAMAPPL
*                                            Defines VTAM application named
*                                            WSIMAPPL.
SLU      LU        PATH=(SIMPLE),            Specifies path named SIMPLE.
                   MAXSESS=(0,001),          Defines one secondary half-
                   INIT=SEC,                 session for SLU.  Specifies
                   LUTYPE=LU2,               LU type and the name of a VTAM
                   DLOGMOD=D4A32782,         logon mode table entry, delays
                   THKTIME=UNLOCK,           start of intermessage delays,
                   LOGDSPLY=BOTH,            writes display buffers to the
                   RSTATS=YES                log data set, accumulates
*                                            response-time statistics.
INITSESS MSGTXT
*
*   INITSESS deck simulates SLU logging on to TSO.  In this deck, the TSO
*   resource name is TSO01, the user ID is ID02, and the password is PW02.
*
*                                            Beginning of INITSESS.
WTO1     WTO       (STARTING $MSGTXTID$)     Sends message to console.
CMND1    CMND      COMMAND=INIT,             Initiates session.
                   RESOURCE=TSO01
1        IF        LOC=RU+0,                 Tests for characters ENTER
                   TEXT=(ENTER USERID),      USERID.
                   SCAN=YES,THEN=CONT,
                   ELSE=WAIT
WAIT1    WAIT
*                                            Interrupts message generation.
MSG1     TEXT      (ID02)                    Enters user ID ID02.
WTO2     WTO       (Logging on TSO as ),     Sends message to console.
                   (ID02)
ENTER1   ENTER                               Sets ENTER AID byte.
2        IF        LOC=RU+0,                 Tests for characters ENTER
                   TEXT=(ENTER LOGON),       LOGON.
                   SCAN=YES,THEN=CONT
WAIT2    WAIT
*                                            Interrupts message generation.
MSG2     TEXT      (PW02)                    Enters user password PW02.
ENTER2   ENTER                               Sets ENTER AID byte.
3        IF        LOC=RU+0,                 Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),         PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
4        IF        LOC=RU+0,TEXT=(READY),    Tests for characters READY.
                   SCAN=YES,THEN=CISPFDECK
WAIT3    WAIT
*                                            Interrupts message generation.
WTO3     WTO       (Logged on and ),         Sends message to console.
                   (received ISPF Primary ),
                   (Menu)
         ENDTXT                              End of INITSESS.
*
ISPFDECK MSGTXT
*                                            Beginning of ISPFDECK.
*   ISPF deck simulates SLU accessing ISPF and then waiting for the ISPF
*   primary menu to be returned before continuing message generation.
*
WTO1     WTO       (STARTING $MSGTXTID$)     Sends message to console.
MSG1     TEXT      (ISPF)                    Enters ISPF.
ENTER1   ENTER                               Sets ENTER AID byte.
1        IF        LOC=RU+0,                 Tests for characters ISPF/PDF
                   TEXT=(ISPF/PDF ),         PRIMARY.
                   (PRIMARY),SCAN=YES,
                   THEN=CONT
WAIT1    WAIT
*                                            Interrupts message generation.
WTO2     WTO       (Logged on and ),         Sends message to console.
                   (received ),
                   (ISPF Primary Menu)
         ENDTXT                              End of ISPFDECK.
*
LOGOFF   MSGTXT
*                                            Beginning of LOGOFF.
*   The following message generation deck exits ISPF and logs the LU
*   off TSO.
*
WTO1     WTO       (Starting $MSGTXTID$)     Sends message to console.
MSG1     TEXT      (X)                       Enter text to exit ISPF.
1        IF        LOC=RU+0,TEXT=(READY),    Tests for characters READY.
                   SCAN=YES,THEN=CONT
WAIT1    WAIT
*                                            Interrupts message generation.
MSG2     TEXT      (LOGOFF)                  Enter text LOGOFF.
WTO2     WTO       (Logged off TSO)          Sends message to console.
WAIT2    WAIT
*                                            Interrupts message generation.
OPCMND1  OPCMND    (ZEND)                    Specifies operator command ZEND.
         ENDTXT                              End of LOGOFF.
*
CLEAR    MSGTXT
*                                            Beginning of CLEAR.
*   WSim calls the following message generation deck to clear the screen
*   when '***' appears in the request unit (RU+0).
*
CLEAR1   CLEAR                               Clears the screen.
         ENDTXT                              End of CLEAR.
```

Storing your scripts

To store your scripts in the data sets that WSim uses to perform simulations, you can use the STL Translator, thePreprocessor or the ITPSYSIN utility program, or you can use your editor to enter the scripts directly into the appropriate data sets. It is recommended that you use the Preprocessor to store your network definition and decks because it checks the syntax of your statements. In addition, the Preprocessor provides several reports, including an indication of where errors occurred and an estimate of the amount of storage required to store the network control blocks. ITPSYSIN also stores your scripts in the appropriate data sets but does not check the syntax of your statements. If you use STL to create your scripts and you include the network definition in your STL input, the STL Translator invokes thePreprocessor for you.

Using the preprocessor

The Preprocessor stores the network definition and message generation decks in the data sets named in the JCL or TSO CLIST used to run the Preprocessor. The data set for the network definition is named by the INITDD DD statement in the JCL. The data set for the message generation decks is named by the MSGDD DD statement in the JCL. If you want, you can use the same data set for both your network definition and message generation decks.

To use the Preprocessor, place all the message generation decks after the last statement in the network definition in your input sequential data set. The data set you place them in must be referenced by the SYSIN DD statement in the JCL or the ALLOC DDNAME(SYSIN) statement in TSO.

Start the Preprocessor by running the WSim load module ITPENTER with the parameter PREP specified. If the Preprocessor detects no syntax errors, the network definition and message generation decks are stored in the appropriate data sets. If there are syntax errors, the Preprocessor provides a report indicating where the errors occurred. When Preprocessor is finished processing, you receive a return code indicating whether the program has succeeded or failed. For a list of return codes, see *WSim Messages and Codes*.

You can include multiple networks in the input to the Preprocessor or specify that a network definition that has already been preprocessed is not to be rewritten to the INITDD data set. You can also replace any members of the data sets that have already been preprocessed.

See *WSim Utilities Guide* for complete instructions for using the Preprocessor.

Using the ITPSYSIN utility program

ITPSYSIN stores the network definition and message generation decks in the appropriate data sets without checking syntax. ITPSYSIN runs much faster than the Preprocessor, providing a quick method for storing your scripts. Use ITPSYSIN any time you do not need to check syntax, for example, when storing previously processed networks or automatically generated networks. If you use ITPSYSIN to store a script for a simulation you are ready to run, syntax is checked when you initialize the network. In this way, you can avoid checking the syntax twice.

To use ITPSYSIN, place your message generation decks in a sequential data set after the last statement in your network definition. As with the Preprocessor, the data set that contains your network definition and decks is referenced by the SYSIN DD statement in the JCL or the ALLOC DDNAME(SYSIN) statement in TSO.

When you run ITPSYSIN, your network definition and your message generation decks are placed in the appropriate data set. After ITPSYSIN is complete, you receive a return code indicating whether the program has succeeded or failed.

Refer to *WSim Utilities Guide* for complete information about running the ITPSYSIN utility program.

Chapter 20. Analyzing simulation results

When you run a simulation, you can use WSim output and online facilities to analyze the results. WSim produces certain types of output automatically; other types can be requested with operator commands or by running one of the WSim utilities. With the online facilities provided by WSim, you can use online displays to monitor the progress of the test.

When you use WSim reports and displays, remember that WSim is an external driver; it acts like a terminal operator typing at a terminal. For this reason, WSim reports and displays do not provide specific information about the internal workings of your system. To obtain this information, use the same type of monitoring programs during the test that you would use if you were testing your system without WSim.

This chapter provides information about running a simulation and describes the following WSim output and online facilities:

- WSim output
  - Using operator reports
  - Using the log data set
    - The Loglist Utility
    - The Log Compare Utility
    - The Response Time Utility.
- Online facilities
  - Display Monitor Facility
  - Response-Time Statistics Facility.

The chapter also provides detailed examples of selected WSim reports and online displays. Each example resulted from a simulation based on the script presented in Chapter 19, “Integrating decks with network definitions,” on page 247.

Running a simulation

After you check the syntax of your script and store the network definition and the message generation decks in the appropriate data sets, you are ready to run a simulation. As mentioned in “Developing scripts” on page 103, your first simulation should use a small network definition and a simple message generation deck. When this simulation runs correctly, you can gradually make the network definition more complex and add more message generation decks.

The steps required to run a simulation are described in *WSim User's Guide*; it provides detailed information about running WSim. These books also provide information about using the various features available when operating WSim, such as the Display Monitor Facility.

Using WSim output

WSim output can help you understand how WSim interacts with the system under test. It is especially useful when you debug scripts or when you want to ensure that the simulation is proceeding as expected.

WSim provides the following output to help analyze your simulations:

- Operator reports that indicate what is happening during operation
- The log data set that contains complete records of the test run. You can analyze the log data set with the following utilities:
  - The Loglist Utility, which lists the WSim log data set in a formatted report.
  - The Log Compare Utility, which compares the 3270 display records from two WSim log data sets and reports when a difference is detected.
  - The Response Time Utility, which provides detailed statistical analysis of the response times.

Using operator reports

You can use operator reports to determine what is happening during a simulation. Some operator reports, such as end of run reports, are printed automatically; you can obtain others by issuing operator commands during the simulation or by coding an option on the NTWRK statement.

This section briefly describes the following types of operator reports:

- Interval reports
- End of run reports
- Trace reports
- Inactivity Report.

The examples of each report included in the following sections resulted from running the sample script introduced in Chapter 19, “Integrating decks with network definitions,” on page 247. To view the sample script, see “Understanding the sample script” on page 255.

For additional information about operator reports, see *WSim User's Guide*.

Interval reports

Interval reports provide information about the current activity and status of each simulated resource in the network. You can use these reports to monitor what is happening within the simulated network. For example, you can use interval reports to ensure that a specific terminal is sending and receiving messages.

Figure 24 on page 261 illustrates the interval report that resulted from running the sample script.

```
                                                     INTERVAL REPORT
         NETWORK  SAMP1                              Sample Network 1                                          NTWRKUTI     100
                           STATUS    MESSAGES       ADDRESSES    POLLS    NEGATIVE                RESPONSES          ISOLATED PACING
                                 RECEIVED   SENT    RECEIVED   RECEIVED   RESPONSES       RECEIVED          SENT     RECEIVED   SENT
                                                                         (RR'S SENT)     DEF     EXC     DEF     EXC
VTAMAPPL WSIMAPPL            S
  LU       SLU-1             A        17       6          0          0          0          1       0      17       0       0       0
  VTAMAPPL TOTALS                     17                  0                     0                  0               0               0
                                               6                     0                     1              17               0
  CUMULATIVE TOTALS                   17                  0                     0                  0               0               0
                                               6                     0                     1              17               0
  INTERVAL TOTALS                     17                  0                     0                  0               0               0
                                               6                     0                     1              17               0
  RATE (PER MINUTE)                   17                  0                     0                  0               0               0
                                               6                     0                     1              17               0
```

Figure 24. WSim Interval Report

End of run reports

Ends of run reports are a type of interval report that provides summary data about the activity and status of each resource during the simulation. You can use them to obtain general information about what happened during the test, and they print automatically at the end of each run.

Figure 25 illustrates the end of run report that resulted from running the sample script.

```
                                                     END OF RUN REPORT
         NETWORK  SAMP1                              Sample Network 1                                          NTWRKUTI     100
                           STATUS    MESSAGES       ADDRESSES    POLLS    NEGATIVE                RESPONSES          ISOLATED PACING
                                 RECEIVED   SENT    RECEIVED   RECEIVED   RESPONSES       RECEIVED          SENT     RECEIVED   SENT
                                                                         (RR'S SENT)     DEF     EXC     DEF     EXC
VTAMAPPL WSIMAPPL
  LU       SLU-1                      52      18          0          0          0          3       0      52       0       0       0
  VTAMAPPL TOTALS                     52                  0                     0                  0               0               0
                                              18                     0                     3              52               0
  CUMULATIVE TOTALS                   52                  0                     0                  0               0               0
                                              18                     0                     3              52               0
```

Figure 25. WSim End of Run Report

Trace reports

You can use trace reports to obtain traces of the information transmitted or received between the WSim control program and the system under test. This information can help you debug your network definition statements and message generation decks.

For more information about trace reports, see *WSim User's Guide*.

The inactivity report

The Inactivity Report contains information on the status of terminals and devices in the network. You can use this report to find problems in your network. For example, you can know whether a particular terminal is active or inactive by seeing the Inactivity Report. It includes the following information:

- Last message transmitted
- Last message received
- Time of the last message transmitted
- Time of the last message received
- Name of the message generation deck, if any
- Response field coded on the TEXT statement, if any
- Indication of whether the device was last transmitting or receiving.

Note: The Inactivity Report is time stamped so that you can determine how long a terminal has been inactive.

For more information about this report, refer to *WSim User's Guide*, which also provides a sample Inactivity Report. For a list of the criteria used to determine if a terminal is active or inactive, refer to Part 1, "Defining WSim networks," on page 1.

Using the log data set

The log data set stores all data transmitted or received by a network's simulated resources during a simulation. The message logging facility that writes data to the log data set is active by default for every resource in the network. If you do not want messages logged for certain resources, you can deactivate message logging for an entire network by specifying MLOG=NO on the NTWRK network definition statement. You can also code the MLOG operand on the APPCLU, VTAMAPPL, or TCPIP network definition statements.

Although you do not use the log data set directly, you can write user exit routines to read the log data set and print information about the simulation. For more information about writing user exit routines, refer to *WSim User Exits*.

As discussed in the following sections, you can also format and analyze the log data set with three utilities provided by WSim:

- The Loglist Utility
- The Log Compare Utility
- The Response Time Utility.

With the output from these utilities, you can debug scripts and analyze the results of a simulation.

In general, you do not need to know how WSim logs and time stamps messages to use the log data set. However, you might find this information helpful as you debug your scripts or analyze response times. For more information about how messages are logged and time stamped, see *WSim Utilities Guide*.

Formatting the log data set with the Loglist Utility

The Loglist Utility formats and prints the records in the log data set. You can specify the types of records you want to see, and you can limit the resources for which records are printed. With these capabilities, you can gain specific information about the behavior of your network.

You can use the formatted records produced by the Loglist Utility to debug your scripts and to learn about your network. For example, you can determine whether new application program functions executed correctly. You can also obtain an estimate of your system's performance by using the time stamps in each record to compute the response times between the messages transmitted and received by the simulated terminals.

One feature of the Loglist Utility that is especially helpful is the printing of screen image records. The images are updated each time a message is sent or received by a device, and you can tell WSim to write the image to the log each time it is updated.

When the simulation is complete, you can use the Loglist Utility to format and print these screen images. The output of the screen images looks the same as the images you would see on the real device. You'll find it helpful to use these images when you are trying to get a new message generation deck to work because screen images are easier to understand than the raw 3270 data stream.

The following example shows the input file used to run the Loglist Utility after running the sample script:

```
RUN
END
```

Figure 26 illustrates the beginning of the Loglist Utility output; Figure 27 illustrates another part of the output, showing the first panel for ISPF.

Figure 26. Beginning of WSim Loglist Utility Report

```
WSIM LOGLIST OUTPUT

NETWORK APPCLU/TCPIP/VTAMAPPL    DEV/LU/TP         START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENC
 NAME             NAME             NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
-------------------------------------------------------------------------------------------------------------------------------------
                                                 12484431 0096031  11000000  CNSL  0800 000000      7
-------------------------------------------------------------------------------------------------------------------------------------
                                                 12484458 0096031  11000000  CNSL  0800 000000     49
  ITP029I INITIALIZATION COMPLETE FOR NETWORK SAMP1
-------------------------------------------------------------------------------------------------------------------------------------
                                                 12490631 0096031  11000000  CNSL  0800 000000      1
-------------------------------------------------------------------------------------------------------------------------------------
                                                 12490650 0096031  11000000  CNSL  0800 000000     29
  ITP006I NETWORK SAMP1 STARTED
-------------------------------------------------------------------------------------------------------------------------------------
SAMP1           WSIMAPPL          SLU-1          12490651 0096031  11000000  MTRC  0100 083060     53   E2   INITSESS  00          0
  ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK INITSESS
-------------------------------------------------------------------------------------------------------------------------------------
                                                 12490651 0096031  11000000  CNSL  0800 000000     51
  ITP137I SAMP1    SLU     -00001 - Starting INITSESS
-------------------------------------------------------------------------------------------------------------------------------------
```

Figure 27. WSim Loglist Utility Report including ISPF panel

```
WSIM LOGLIST OUTPUT
-------------------------------------------------------------------------------------------------------------------------------------
SAMP1           WSIMAPPL          SLU-1          12492819 12492820 12492820  RECV  8000 080000   1084   E2   ISPFDECK  00         82
   RECV (DATA) REQUEST
       TH   2C0001010009          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=9
       RH   038020   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=DEF1
                     INDICATORS=        CHANGE DIRECTION
       RU   F1C31140 401DF83C 40D86040 40C9E2D7    C661D7C4 C640D7D9 C9D4C1D9 E840D6D7   *1C.  .8. Q-  ISPF/PDF PRIMARY OP*
 00000020   E3C9D6D5 40D4C5D5 E440403C C150601D    E8D6D7E3 C9D6D540 407E7E7E 6E1DC83C   *TION MENU  .A&-.YOPTION  ===>.H.*
 00000040   C25F001D 601DE83C C35C401D 60E4E2C5    D9C9C440 40406040 E3D7D5E2 F0F24040   *B—..-.Y.C* .-USERID   - WSIM02  *
 00000060   1DE84040 40F0401D 60C9E2D7 C640D7C1    D9D4E240 406040E2 97858389 86A840A3   *.Y   0 .-ISPF PARMS  - Specify t*
 00000080   85999489 95819340 81958440 A4A28599    40978199 819485A3 8599A240 40401D60   *erminal and user parameters   .-*
 000000A0   E3C9D4C5 3CC4F640 6040F1F1 7AF4F940    40401DE8 404040F1 401D60C2 D9D6E6E2   *TIME.D6 - 11:49   .Y   1 .-BROWS*
 000000C0   C53CC5D3 406040C4 89A29793 81A840A2    96A49983 85408481 A3814096 994096A4   *E.EL - Display source data or ou*
 000000E0   A397A4A3 409389A2 A3899587 A2401D60    E3C5D9D4 C9D5C1D3 406040F3 F2F7F840   *tput listings .-TERMINAL - 3278 *
 00000100   4040401D E8404040 F2401D60 C5C4C9E3    3CC6E340 6040C399 8581A385 40969940   *   .Y   2 .-EDIT.FT - Create or *
 00000120   83888195 878540A2 96A49983 85408481    A3813CC7 4C401D60 D7C640D2 C5E8E240   *change source data.G< .-PF KEYS *
 00000140   406040F2 F43CC760 401DE840 4040F340    1D60E4E3 C9D3C9E3 C9C5E240 40406040   * - 24.G- .Y   3 .-UTILITIES   - *
 00000160   D7859986 96999440 A4A38993 89A3A840    86A49583 A3899695 A23CC8F0 401DE840   *Perform utility functions.H0 .Y *
 00000180   4040F440 1D60C6D6 D9C5C7D9 D6E4D5C4    40406040 C995A596 92854093 819587A4   *  4 .-FOREGROUND  - Invoke langu*
 000001A0   81878540 97999683 85A2A296 99A24089    95408696 99858799 96A49584 3C4A4040   *age processors in foreground.`  *
 000001C0   1DE84040 40F5401D 60C2C1E3 C3C83C4A    D3406040 E2A48294 89A34091 96824086   *.Y   5 .-BATCH.`L - Submit job f*
 000001E0   96994093 819587A4 81878540 97999683    85A2A289 95873C4B 50401DE8 404040F6   *or language processing..& .Y   6*
 00000200   401D60C3 D6D4D4C1 D5C43C4B E3406040    C595A385 9940E3E2 D6408396 94948195   * .-COMMAND..T - Enter TSO comman*
 00000220   84409699 40C3D3C9 E2E33C4C 60401DE8    404040F7 401D60C4 C9C1D3D6 C740E3C5   *d or CLIST.<- .Y   7 .-DIALOG TE*
 00000240   E2E34060 40D78599 86969994 40848981    93968740 A385A2A3 8995873C 4DF0401D   *ST - Perform dialog testing.(0 .*
 00000260   E8404040 F8401D60 D3D440E4 E3C9D3C9    E3C9C5E2 6040D785 99869699 94409389   *Y   8 .-LM UTILITIES- Perform li*
 00000280   82998199 A8408184 94899589 A2A39981    A3969940 A4A38993 89A3A840 86A49583   *brary administrator utility func*
 000002A0   A3899695 A23C4F40 401DE840 4040F940    1D60C9C2 D440D7D9 D6C4E4C3 E3E26040   *tions.|  .Y   9 .-IBM PRODUCTS- *
 000002C0   C1848489 A3899695 819340C9 C2D44097    99968799 81944084 85A58593 96979485   *Additional IBM program developme*
 000002E0   95A34097 999684A4 83A3A23C 5050401D    E8404040 C1401D60 E3D7D5E2 61C1E3D7   *nt products.&& .Y   A .-WSIM/ATP*
 00000300   40404040 6040C995 A5969285 40A38885    40E3D7D5 E261C1E3 D740C584 89A39699   *    - Invoke the WSIM/ATP Editor*
 00000320   40869699 40D1A493 89853CD1 60401DE8    404040C3 401D60C3 C8C1D5C7 C5E23CD1   * for Julie.J- .Y   C .-CHANGES.J*
 00000340   F3406040 C489A297 9381A840 A2A49494    8199A840 96864083 88819587 85A24086   *3 - Display summary of changes f*
 00000360   969940A3 8889A240 99859385 81A2853C    D2F0401D E8404040 E3401D60 E3E4E3D6   *or this release.K0 .Y   T .-TUTO*
 00000380   D9C9C1D3 40404040 6040C489 A2979381    A8408995 86969994 81A38996 95408182   *RIAL    - Display information ab*
 000003A0   96A4A340 C9E2D7C6 61D7C4C6 3CD44040    1DE84040 40E7401D 60C5E7C9 E33CD4D3   *out ISPF/PDF.M  .Y   X .-EXIT.ML*
 000003C0   406040E3 85999489 9581A385 40C9E2D7    C640A4A2 89958740 93968740 81958440   * - Terminate ISPF using log and *
 000003E0   9389A2A3 40848586 81A493A3 A23CD550    401DE83C D660401D 60C595A3 85991DE8   *list defaults.N& .Y.O- .-Enter.Y*
 00000400   C5D5C41D 60839694 94819584 40A39640    A3859994 899581A3 8540C9E2 D7C64B3C   *END.-command to terminate ISPF..*
 00000420   D7F0401D E83CD940 401DE83C 40400011    C15E13                                *P0 .Y.R  .Y.  ..A;.             *
-------------------------------------------------------------------------------------------------------------------------------------
SAMP1           WSIMAPPL          SLU-1          12492820 0096031  11000000  MTRC  0100 083000     67   E2   ISPFDECK  00         83
  ITP429I IN  IF CLE ( NETWORK IF   ) NOT MET - ELSE ACTION NOT CODED
  ITP427I IN  IF 0   (ISPFDECK 00004) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)
-------------------------------------------------------------------------------------------------------------------------------------
SAMP1           WSIMAPPL          SLU-1          12492821 12492824 12492219  XMIT  8000 880000      9   E2   ISPFDECK  00         85
```

```
   XMIT RESPONSE
       TH   2C0001010009          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=9
       RH   838000   RESPONSE        FM DATA          ONLY IN CHAIN   RESPONSE TYPE=DEF1

            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1| ----------------------  ISPF/PDF PRIMARY OPTION MENU  ------------------------| 1
 2| OPTION  ===>                                                                  | 2
 3|                                                            USERID   - WSIM02  | 3
 4|    0  ISPF PARMS  - Specify terminal and user parameters    TIME     - 11:49   | 4
 5|    1  BROWSE      - Display source data or output listings  TERMINAL - 3278    | 5
 6|    2  EDIT        - Create or change source data            PF KEYS  - 24      | 6
 7|    3  UTILITIES   - Perform utility functions                                 | 7
 8|    4  FOREGROUND  - Invoke language processors in foreground                  | 8
 9|    5  BATCH       - Submit job for language processing                        | 9
10|    6  COMMAND     - Enter TSO command or CLIST                                |10
11|    7  DIALOG TEST - Perform dialog testing                                    |11
12|    8  LM UTILITIES- Perform library administrator utility functions           |12
13|    9  IBM PRODUCTS- Additional IBM program development products               |13
14|    A  WSIM        - Invoke WSIM                                               |14
15|    C  CHANGES     - Display summary of changes for this release               |15
16|    T  TUTORIAL    - Display information about ISPF/PDF                        |16
17|    X  EXIT        - Terminate ISPF using log and list defaults                |17
18|                                                                              |18
19| Enter END command to terminate ISPF.                                         |19
20|                                                                              |20
21|                                                                              |21
22|                                                                              |22
23|                                                                              |23
24|                                                                              |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW(  2) COLUMN( 15)   AID: NULL DISPLAY AID          WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083060     53   E2   ISPFDECK  00        87
  ITP447I MSG GEN ENTERED: STMT# 00006 OF DECK ISPFDECK
------------------------------------------------------------------------------------------------------------------------------------
                                              12492921 0096031  11000000  CNSL  0800 000000     74
  ITP137I SAMP1    SLU     -00001 - Logged on and received ISPF Primary Menu
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083020     80   E2   INITSESS  00        88
  ITP452I RETURN FROM STMT# 00007 OF DECK ISPFDECK TO STMT# 00015 OF DECK INITSESS
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083020     80   E2   INITSESS  00        88
  ITP137I SAMP1    SLU     -00001 - Logged on and received ISPF Primary Menu
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083020     48   E2   LOGOFF    00        89
  ITP449I MSG GEN CONTINUES: DECK LOGOFF   STARTED
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083020     48   E2   LOGOFF    00        89
  ITP137I SAMP1    SLU     -00001 - Starting LOGOFF
------------------------------------------------------------------------------------------------------------------------------------
SAMP1          WSIMAPPL         SLU-1         12492921 0096031  11000000  MTRC  0100 083020     53   E2   LOGOFF    00        90
  ITP448I MSG GEN ENDED:   STMT# 00005 OF DECK LOGOFF
------------------------------------------------------------------------------------------------------------------------------------
```

Comparing 3270 display records with the Log Compare Utility

The Log Compare Utility compares 3270 display records from 2 log data sets and reports when a difference is detected. You select the records to be compared and the fields to be compared for each record. You can also request a set of reports that identify the differences found between the display records. These reports can help you verify whether an application has changed over a period or determine the results of a regression test.

The Log Compare Utility also prints an Active Command List automatically after each run. This report lists the commands you issued and the operand values that were in effect during the run. The Log Compare Utility can also print more reports that show you which display records were compared and the differences that were detected.

The following example shows the input file used to run the Log Compare Utility:

```
REPORT R,S,C,D
RUN
END
```

When you use this input file, the Log Compare Utility provides the following reports:

- Active Command List
- Complete Records List
- Compare List

- Differences Report
- Summary Report.

Figure 28 illustrates the Log Compare Utility output that resulted after running the sample script.

Figure 28. WSim Log Compare Utility reports

```
WSIM COMPARE UTILITY OUTPUT                                                 TIME  9.44.58, JANUARY 31, 2002     PAGE      1
------------------------------------------------------------------------------------------------------------------------------------
                                                  Active Command List

------------------------------------------------------------------------------------------------------------------------------------

   Selection Commands Issued
   -------------------------

              There were no Selection commands issued for this run

   Process Commands Issued
   -----------------------

        Command    Command
         Number     Type       Operands
        -------  -----------   ------------------------------------------------------------
                   REPORT      RECORDS,COMPARES,DIFFERENCES,SUMMARY

------------------------------------------------------------------------------------------------------------------------------------
                                                  Complete Records List
Master:  NETWORK     SAMP1                                                             Test:  NETWORK     SAMP1
         VTAMAPPL    WSIMAPPL                                                                 VTAMAPPL    WSIMAPPL
         DEV/LU      SLU-00001                                                                DEV/LU      SLU-00001
------------------------------------------------------------------------------------------------------------------------------------

  MASTER Records
  --------------

    Sequence
     Number      MSGTXT        Usage        Reason
    --------    ----------    ----------    -------------------------------------------------------------------------------
         0       INITSESS      Used
         1       INITSESS      Used
         2       INITSESS      Used
         3       INITSESS      Used
         4       INITSESS      Used
         5       ISPFDECK      Used
         6       ISPFDECK      Used
         7       ISPFDECK      Used
         8       LOGOFF        Used
         9       LOGOFF        Used
        10       LOGOFF        Used
        11       LOGOFF        Used
        12       INITSESS      Used
        13       INITSESS      Used
        14       INITSESS      Used
        15       INITSESS      Used
        16       ISPFDECK      Used
        17       ISPFDECK      Used
        18       ISPFDECK      Used
        19       LOGOFF        Used
        20       LOGOFF        Used
        21       LOGOFF        Used
        22       LOGOFF        Used
        23       INITSESS      Used
        24       INITSESS      Used
        25       INITSESS      Used
        26       INITSESS      Used
        27       ISPFDECK      Used
        28       ISPFDECK      Used
        29       ISPFDECK      Used
        30       LOGOFF        Used
        31       LOGOFF        Used
        32       LOGOFF        Used
  TEST Records
  ------------

    Sequence
     Number      MSGTXT        Usage        Reason
    --------    ----------    ----------    -------------------------------------------------------------------------------
         0       INITSESS      Used
         1       INITSESS      Used
         2       INITSESS      Used
         3       INITSESS      Used
         4       INITSESS      Used
         5       ISPFDECK      Used
         6       ISPFDECK      Used
         7       ISPFDECK      Used
         8       LOGOFF        Used
         9       LOGOFF        Used
        10       LOGOFF        Used
        11       LOGOFF        Used
        12       INITSESS      Used
        13       INITSESS      Used
        14       INITSESS      Used
        15       INITSESS      Used
        16       ISPFDECK      Used
        17       ISPFDECK      Used
        18       ISPFDECK       Used
        19       LOGOFF         Used
        20       LOGOFF         Used
        21       LOGOFF         Used
        22       LOGOFF         Used
        23       INITSESS       Used
        24       INITSESS       Used
        25       INITSESS       Used
        26       INITSESS       Used
        27       ISPFDECK       Used
        28       ISPFDECK       Used
        29       ISPFDECK       Used
        30       LOGOFF         Used
        31       LOGOFF         Used
        32       LOGOFF         Used

------------------------------------------------------------------------------------------------------------------------------------
                                                  Compare List
Master:  NETWORK     SAMP1                                                                Test:  NETWORK     SAMP1
         VTAMAPPL    WSIMAPPL                                                                    VTAMAPPL    WSIMAPPL
         DEV/LU      SLU-00001                                                                   DEV/LU      SLU-00001
------------------------------------------------------------------------------------------------------------------------------------

    MASTER            TEST                            ALL
Sequence Number   Sequence Number   Checkonly  Mask   Mask    Result     REASON FOR DIFFERENCE
---------------   ---------------   ---------  -----  ----   ---------   -----------------------------------------------------------
         0                 0                                   Equal
         1                 1                                   Equal
         2                 2                                   Equal
         4                 4                                   Equal
         5                 5                                   Equal
         6                 6                                   Equal
         7                 7                                   Equal
         8                 8                                   Equal
         9                 9                                   Equal
        10                10                                   Equal
        11                11                                   Equal
        12                12                                   Equal
        13                13                                   Equal
        14                14                                   Equal
        15                15                                   Equal
        16                16                                   Equal
        17                17                                   Equal
        18                18                                   Equal
        19                19                                   Equal
        20                20                                   Equal
        21                21                                   Equal
        22                22                                   Equal
        23                23                                   Equal
        24                24                                   Equal
        25                25                                   Equal
        26                26                                   Equal
        27                27                                   Equal
        28                28                                   Equal
        29                29                                   Equal
        30                30                                   Equal
        31                31                                   Equal
        32                32                                   Equal
------------------------------------------------------------------------------------------------------------------------------------
                                                  Differences Report
Master:  NETWORK     SAMP1                                                                Test:  NETWORK     SAMP1
         VTAMAPPL    WSIMAPPL                                                                    VTAMAPPL    WSIMAPPL
         DEV/LU      SLU-00001                                                                   DEV/LU      SLU-00001
------------------------------------------------------------------------------------------------------------------------------------
No differences were found for this resource

------------------------------------------------------------------------------------------------------------------------------------

                                                  Summary Report

------------------------------------------------------------------------------------------------------------------------------------

                                                                                                   Synchronization        Sequence
                                         Result #Records  Processed #Records #Differences   Run              SYNCPOINT     Number
Resource                                  (RC)   MASTER      TEST   Compared   Detected   Aborted Attempted Command Used MASTER TEST
---------------------------------------- ------ --------   -------- -------- ------------ ------- --------- ------------ -----------
NETWORK     SAMP1
VTAMAPPL    WSIMAPPL
DEV/LU      SLU-00001                        0       33       33       33         0       NO       NO

------------------------------------------------------------------------------------------------------------------------------------
```

Determining response times with the Response Time Utility

The Response Time Utility is a postprocessor that analyzes the message log data set and measures the time it takes to enter a command at a simulated terminal and receive a response from the system under test. You use this utility when you need detailed statistics about response times following a simulation.

The Response Time Utility uses the time stamps from a pair of transmit and receive records and calculates response times for each terminal. Although the Response Time Utility uses a set of default rules for determining the transmit and receive record pairings, you can change these rules by specifying parameters for the utility to use when selecting the transmit and receive records. You can also define logical transactions to the Response Time Utility by specifying the messages that mark the beginning and end of the transaction. These transactions can include any number of messages.

Because the Response Time Utility determines the transmit-receive pairs for a particular terminal, be sure that the terminals in your network definition have unique names.

Note: To monitor response time during a simulation, WSim provides the Response-Time Statistics (RSTATS) facility. For more information about RSTATS, see "Using the Response-Time Statistics Facility" on page 269.

For more information about using the Response Time Utility, refer to *WSim Utilities Guide*.

The following example shows the input file used to run the Response Time Utility:

```
REPORT LEVEL=TERM,TERM=(TGRAPH,CGRAPH,GRAPH,NOTRANS,LIST)
RUN
END
```

The commands issued by this input file specify that a response time report is to be printed for each terminal, group of terminals, and for the summary of all terminals. In addition, the file provides a Time Graph, Cumulative Distribution Graph, Frequency Distribution Graph, List of Computed Response Times, and a single report for all transaction types.

Figure 29 illustrates the Response Time Utility output that resulted after running the sample script.

Figure 29. WSim Response Time Utility Report

```
WSIM RESPONSE TIME ANALYSIS
--------------------------------------------------------------------------------------------------------------------------
TERMINAL REPORT    NETWORK  ALL NETWORKS      PROCESS  SYSTEM      TIME LIMITS ALL
                   VTAMAPPL WSIMAPPL          EXIT                 START TIME  124907
                   TERMINAL SLU-1             TERMTYPE LU2         END TIME    125035
--------------------------------------------------------------------------------------------------------------------------
RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT
         0.06        1            0.07        2            0.43        1            0.78        1            0.81        1
         1.32        1            1.37        1            1.41        1            1.47        1            1.57        1
         1.67        1            1.70        1            1.75        1            1.89        1            3.42        1
         3.94        1            5.12        1
MEAN   RESPONSE            1.60   MESSAGES SENT           18   NUMBER OF RESPONSES        18
MEDIAN RESPONSE            1.44      AVERAGE LENGTH       13      PER MINUTE              12
MODE   RESPONSE            0.07      PER MINUTE           12   RESPONSES DISCARDED         0
LOW    RESPONSE            0.06   MESSAGES RECEIVED       35   VARIANCE               1.8355
HIGH   RESPONSE            5.12      AVERAGE LENGTH      223   95 PERCENT CI              --
AVERAGE QUEUE TIME         0.03      PER MINUTE           23
PERCENTILE    RESPONSE TIME        AVERAGE
    90               3.42          1.23

WSIM RESPONSE TIME ANALYSIS
                                                  RESPONSE TIME FREQUENCY DISTRIBUTION
NETWORK               100 |---------------------------------------------------------------------------------------------------------|
VTAMAPPL WSIMAPPL         |                                                                                                         |
TERMINAL SLU-1            |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                       90 |---------------------------------------------------------------------------------------------------------|
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                       80 |---------------------------------------------------------------------------------------------------------|
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                       70 |---------------------------------------------------------------------------------------------------------|
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
                       60 |---------------------------------------------------------------------------------------------------------|
                          |                                                                                                         |
                          |                                                                                                         |
                          |                                                                                                         |
           PERCENTAGE  50 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
               OF         |                                                                                                        |
                          |                                                                                                        |
           RESPONSES      |                                                                                                        |
                       40 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                       30 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                       20 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |     *                                                                                                  |
                          |     *                                                                                                  |
                          |     *                         * *   *                                                                  |
                       10 |-----*-------------------------*-*---*------------------------------------------------------------------|
                          |     *                         * *   *                                                                  |
                          |     *       *     * *         * * * * * *                               *         *                    |
                          |     *       *     * *         * * * * * *                               *         *                    |
                          |     *       *     * *         * * * * * *                               *         *                    |

                              | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + +
                             0.0       0.5       1.0       1.5       2.0       2.5       3.0       3.5       4.0       4.5
                                                             RESPONSE TIME (SECONDS)          INCREMENT =  0.1 SECONDS
                                                                                              PERCENTAGE ABOVE LAST INCREMENT =  5.6
 WSIM RESPONSE TIME ANALYSIS

                                                   CUMULATIVE RESPONSE TIME DISTRIBUTION
 NETWORK              100 |--------------------------------------------------------------------------------------------------------|
 VTAMAPPL WSIMAPPL        |                                                                                                        |
 TERMINAL SLU-1           |                                                                                                        |
                          |                                                                                   * * * * * * * * * *  |
                          |                                                                                                        |
                       90 |--------------------------------------------------------------------------------------------------------|
                          |                                                                         * * * * *                      |
                          |                                                                                                        |
                          |                                         * * * * * * * * * * * * * * * *                                |
                          |                                                                                                        |
                       80 |--------------------------------------------------------------------------------------------------------|
                          |                                       *                                                                |
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                     *                                                                  |
                       70 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                   *                                                                    |
                       60 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                 *                                                                      |
                          |                                                                                                        |
                          |                                                                                                        |
           PERCENTAGE  50 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
               OF         |                                                                                                        |
                          |                               *                                                                        |
           RESPONSES      |                                                                                                        |
                       40 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                                                                                        |
                          |                     * * * * *                                                                          |
                          |                                                                                                        |
                       30 |--------------------------------------------------------------------------------------------------------|
                          |                   *                                                                                    |
                          |                                                                                                        |
                          |                                                                                                        |
                          |             * * *                                                                                      |
                       20 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |     * * * *                                                                                            |
                          |                                                                                                        |
                          |                                                                                                        |
                       10 |--------------------------------------------------------------------------------------------------------|
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |
                          |                                                                                                        |

                              | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + +
                             0.0       0.5       1.0       1.5       2.0       2.5       3.0       3.5       4.0       4.5
                                                             RESPONSE TIME (SECONDS)          INCREMENT =  0.1 SECONDS
 WSIM RESPONSE TIME ANALYSIS
 NETWORK                                              TIME GRAPH OF RESPONSES                                        < MINIMUM
 VTAMAPPL WSIMAPPL                                                                            INTERVAL =    10 SEC   * AVERAGE
 TERMINAL SLU-1                                                                               INCREMENT=   0.1 SEC   > MAXIMUM
                           RESPONSE TIME (SECONDS)
                         |                                                                                                     |
    TIME     NUMBER OF   |                                                                                                    1|
             RESPONSES   |0----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0|
  12.49.10        1      |                   *                                                                                 |
  12.49.20        2      | <  *   >                                                                                            |
  12.49.30        1      |                                       *                                                             |
  12.49.40        2      |             < *  >                                                                                  |
  12.49.50        3      | <      *      >                                                                                     |
  12.50.00        1      |                                                   *                                                 |
  12.50.10        2      |                *>                                                                                   |
  12.50.20        3      | <    *       >                                                                                      |
  12.50.30        1      |                                  *                                                                  |
  12.50.40        2      |              < *>                                                                                   |
 WSIM RESPONSE TIME ANALYSIS
 --------------------------------------------------------------------------------------------------------------------------
 TERMGRP  REPORT    NETWORK  ALL NETWORKS      PROCESS  SYSTEM      TIME LIMITS ALL
                    VTAMAPPL WSIMAPPL          EXIT                 START TIME  124907
                                               TERMTYPE             END TIME    125035
 --------------------------------------------------------------------------------------------------------------------------
 RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT
          0.06        1            0.07        2            0.43        1            0.78        1            0.81        1
          1.32        1            1.37        1            1.41        1            1.47        1            1.57        1
          1.67        1            1.70        1            1.75        1            1.89        1            3.42        1
          3.94        1            5.12        1
 MEAN   RESPONSE             1.60    MESSAGES SENT            18    NUMBER OF RESPONSES         18
 MEDIAN RESPONSE             1.44       AVERAGE LENGTH        13       PER MINUTE               12
 MODE   RESPONSE             0.07       PER MINUTE            12    RESPONSES DISCARDED          0
 LOW    RESPONSE             0.06    MESSAGES RECEIVED        37    VARIANCE                1.8355
 HIGH   RESPONSE             5.12       AVERAGE LENGTH       212    95 PERCENT CI               --
 AVERAGE QUEUE TIME          0.03       PER MINUTE            25
 PERCENTILE    RESPONSE TIME        AVERAGE
     90                 3.42           1.23

 WSIM RESPONSE TIME ANALYSIS                                                TIME 13.53.54,   JANUARY 31, 2002   PAGE      8
 --------------------------------------------------------------------------------------------------------------------------
 SUMMARY  REPORT    NETWORK  ALL NETWORKS      PROCESS  SYSTEM      TIME LIMITS ALL
                                               EXIT                 START TIME  124907
                                               TERMTYPE             END TIME    125035
 --------------------------------------------------------------------------------------------------------------------------
 RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT   RESPONSE TIME    COUNT
          0.06        1            0.07        2            0.43        1            0.78        1            0.81        1
          1.32        1            1.37        1            1.41        1            1.47        1            1.57        1
          1.67        1            1.70        1            1.75        1            1.89        1            3.42        1
          3.94        1            5.12        1
 MEAN   RESPONSE             1.60    MESSAGES SENT            18    NUMBER OF RESPONSES         18
 MEDIAN RESPONSE             1.44       AVERAGE LENGTH        13       PER MINUTE               12
 MODE   RESPONSE             0.07       PER MINUTE            12    RESPONSES DISCARDED          0
 LOW    RESPONSE             0.06    MESSAGES RECEIVED        37    VARIANCE                1.8355
 HIGH   RESPONSE             5.12       AVERAGE LENGTH       212    95 PERCENT CI               --
 AVERAGE QUEUE TIME          0.03       PER MINUTE            25
 PERCENTILE    RESPONSE TIME        AVERAGE
     90                 3.42           1.23
```

Using online facilities

WSim provides two online facilities that enable you to monitor a simulation:

- With the Display Monitor Facility, you can watch the messages sent and received by a simulated 3270 terminal.
- The RSTATS facility provides response time calculations for the simulated resources. The operator can access these values online at the operator console.

The following sections describe how you can use these facilities to detect errors as they happen and to determine response times during a simulation.

Using the Display Monitor Facility

The Display Monitor Facility is a VTAM application program within WSim that displays simulated 3270 display images on a monitor that is accessible by VTAM. It helps you develop and debug scripts by enabling you to watch the messages being sent and received by a simulated 3270 terminal. This is useful when you are trying to get a new message generation deck to work since you can detect errors as they happen. The Display Monitor Facility helps you detect the following types of errors:

- Invalid cursor position
- Incorrect screen format
- Hung devices.

For more information about the WSim display monitor facility, refer to *WSim User's Guide*.

Using the Response-Time Statistics Facility

The Response-Time Statistics (RSTATS) Facility is an online response statistics reporting facility unrelated to the Response Time Utility. It enables you to monitor the response times of simulated devices while WSim is running by measuring the time it takes to transmit data from a simulated terminal and receive a response from the system under test. You can use it to provide online response time calculations for terminals simulated by WSim. This is especially helpful when you want to ensure that messages are being transmitted and received by the simulated resources at reasonable rates.

RSTATS collects online response-time statistics only for those simulated resources that generate messages. These include devices and logical units. WSim supports the RSTATS feature for all terminal types.

For more information about using RSTATS, refer to *WSim User's Guide*.

Figure 30 illustrates an example of the online statistics that resulted from running the sample script.

```
ITP190I  RESPONSE TIME STATISTICS FOR SLU-1
ITP190I  AT 12.49.13.13
ITP190I                 PROCESS SYSTEM      PROCESS ACTUAL
ITP190I  AVERAGE                0.09                0.32
ITP190I  MOST RECENT            0.11                0.42
ITP190I  LOW                    0.02                0.03
ITP190I  HIGH                   0.15                0.51
ITP190I  TOTAL RESPONSES          15                  14
```

Figure 30. Example of RSTATS online Response-Time Utility output

Chapter 21. 3270 extended character set

The 3270 Data Analysis/APL Character Set expands the character set of 3270 terminals by allowing the display of 80 APL-specific characters and 35 TN print train characters not included in the normal character set. This appendix is a list of these extended characters, presented in two different ways. Table 13 shows which hexadecimal values you should use forWSim to transmit specific 2-character sequences. Table 14 on page 273 shows the WSim internal hexadecimal value translated into the terminal buffer for each received 2-character sequence.

Table 13. 3270 Data Analysis/APL sequences transmitted for WSim codes

WSim Code	Extended Character Set Code	Name
08	1D8C	superscript minus
0A	1D8A	subscript 1
0D	1D8D	superscript (
0E	1D8E	superscript plus
0F	1D8F	DA cross
10	1D90	open brace
18	1D1E	plus or minus
1A	1D9A	subscript 2
1B	1D9B	lozenge
1D	1D9B	superscript)
1F	1D9F	histogram
20	1DA0	close brace
21	1DA1	degree
2A	1DAA	subscript 3
2B	1DAB	lower left corner
2C	1DAC	upper left corner
2D	1DAD	left junction
2E	1DAE	right junction
2F	1DAF	bullet
30	1DB0	superscript 0
31	1DB1	superscript 1
32	1DB2	superscript 2
33	1DB3	superscript 3
34	1DB4	superscript 4
35	1D15	superscript 5
36	1DB6	superscript 6
37	1DB7	superscript 7
38	1DB8	superscript 8
39	1D1B9	superscript 9
3A	1DBA	subscript n

Table 13. 3270 Data Analysis/APL sequences transmitted for WSim codes (continued)

WSim Code	Extended Character Set Code	Name
3B	1DBB	lower right corner
3C	1DBC	upper right corner
3D	1DBD	top junction
3E	1DBE	bottom junction
3F	1DBF	extended dash
41	1D81	A underscore
42	1D82	B underscore
43	1D83	C underscore
44	1D84	D underscore
45	1D85	E underscore
46	1D86	F underscore
47	1D87	G underscore
48	1D88	H underscore
49	1D89	I underscore
51	1D91	J underscore
52	1D92	K underscore
53	1D93	L underscore
54	1D94	M underscore
55	1D95	N underscore
56	1D96	O underscore
57	1D97	P underscore
58	1D98	Q underscore
59	1D99	R underscore
62	1DA2	S underscore
63	1DA3	T underscore
64	1DA4	U underscore
65	1DA5	V underscore
66	1DA6	W underscore
67	1DA7	X underscore
68	1DA8	Y underscore
69	1DA9	Z underscore
71	1D6A	and
72	1DC3	dieresis
78	1D6B	or
80	1D7B	tilde
8B	1DC2	down
CA	1D4A	nand
CB	1D4B	nor
CD	1D4E	circle stile
CF	1D4F	circle slope

Table 13. 3270 Data Analysis/APL sequences transmitted for WSim codes (continued)

WSim Code	Extended Character Set Code	Name
DA	1DD2	I beam
DB	1DD3	quote dot
DC	1DD6	del stile
DD	1DD7	delta stile
DE	1D5A	quote quad
DF	1D5B	cap null
EA	1D5E	slash bar
EB	1D5F	slope bar
ED	1DE2	circle bar
EE	1DE3	domino
EF	1DE6	top null
FB	1DC6	del tilde
FC	1DF3	delta underscore
FD	1DC7	log
FE	1DE7	base null

Table 14. WSim code for received 3270 Data analysis/APL sequences

Extended Character Set Code	WSim Code	Name
1D8C	08	superscript minus
1D8A	0A	subscript 1
1D8D	0D	superscript (
1D8E	0E	superscript plus
1D8F	0F	DA cross
1D90	10	open brace
1D1E	18	plus or minus
1D9A	1A	subscript 2
1D9B	1B	lozenge
1D9B	1D	superscript)
1D9F	1F	histogram
1DA0	20	close brace
1DA1	21	degree
1DAA	2A	subscript 3
1DAB	2B	lower left corner
1DAC	2C	upper left corner
1DAD	2D	left junction
1DAE	2E	right junction
1DAF	2F	bullet
1DB0	30	superscript 0
1DB1	31	superscript 1
1DB2	32	superscript 2

Table 14. WSim code for received 3270 Data analysis/APL sequences (continued)

Extended Character Set Code	WSim Code	Name
1DB3	33	superscript 3
1DB4	34	superscript 4
1D15	35	superscript 5
1DB6	36	superscript 6
1DB7	37	superscript 7
1DB8	38	superscript 8
1D1B9	39	superscript 9
1DBA	3A	subscript n
1DBB	3B	lower right corner
1DBC	3C	upper right corner
1DBD	3D	top junction
1DBE	3E	bottom junction
1DBF	3F	extended dash
1D81	41	A underscore
1D82	42	B underscore
1D83	43	C underscore
1D84	44	D underscore
1D85	45	E underscore
1D86	46	F underscore
1D87	47	G underscore
1D88	48	H underscore
1D89	49	I underscore
1D91	51	J underscore
1D92	52	K underscore
1D93	53	L underscore
1D94	54	M underscore
1D95	55	N underscore
1D96	56	O underscore
1D97	57	P underscore
1D98	58	Q underscore
1D99	59	R underscore
1DA2	62	S underscore
1DA3	63	T underscore
1DA4	64	U underscore
1DA5	65	V underscore
1DA6	66	W underscore
1DA7	67	X underscore
1DA8	68	Y underscore
1DA9	69	Z underscore
1D6A	71	and

Table 14. WSim code for received 3270 Data analysis/APL sequences (continued)

Extended Character Set Code	WSim Code	Name
1DC3	72	dieresis
1D6B	78	or
1D7B	80	tilde
1DC2	8B	down
1D4A	CA	nand
1D4B	CB	nor
1D4E	CD	circle stile
1D4F	CF	circle slope
1DD2	DA	I beam
1DD3	DB	quote dot
1DD6	DC	del stile
1DD7	DD	delta stile
1D5A	DE	quote quad
1D5B	DF	cap null
1D5E	EA	slash bar
1D5F	EB	slope bar
1DE2	ED	circle bar
1DE3	EE	domino
1DE6	EF	top null
1DC6	FB	del tilde
1DF3	FC	delta underscore
1DC7	FD	log
1DE7	FE	base null

Part 5. Samples

Chapter 22. Introduction

This chapter introduces Workload Simulator (WSim) and describes the types of examples presented in this manual.

Sample installation networks

The sample installation network definitions supplied in the WSim sample data set (WSIM.SITPSAMP on MVS) are simple, pre-defined networks that you can run after performing the WSim installation procedure. Executing these definitions as soon as possible after the installation is highly recommended for the following reasons:

1. You can use the sample definitions without knowing much about WSim.
2. You do not have to write decks or network definitions before you perform any WSim simulation.
3. If you are a new user, it is a quick way to learn the basics of WSim.
4. You can verify that many of your individual WSim network components are working properly.
5. You can verify that the WSim installation process was successfully completed.
6. It introduces you to the following WSim utility programs:
   - The Preprocessor
   - The sample VTAM application (ITPECHO)
   - The Loglist Utility
   - The Response Time Utility.

Functionally equivalent STL procedures for the example message generation decks are also provided. Therefore, you will be able to use either the message generation decks, or the STL procedures (which you will translate into message generation statements) when running these networks. See *WSim Script Guide and Reference* for more information about this translation process.

Message scripting examples

The samples provided help you to write your own network definitions and message generation decks. The examples show you how to write WSim network configuration statements for the equipment you are simulating and WSim message generation statements for the terminal operator actions you are simulating. The notes after each example help you understand what each example does. Also included, where applicable, are STL procedures which can be translated intoWSim message generation statements using the STL translator.

AVMON example

This example describes the AVMON (Availability Monitor) sample network, discusses the components of the network, and shows you how to modify the network for your configuration.

Chapter 23. Sample installation networks

This chapter describes a sample WSim network definition. The sample WSim test network simulates a 3270 logical unit (LU) Type 2 which logs on to ITPECHO, the WSim sample VTAM application program. Note that the network can be easily modified to simulate other terminal types. You can run the sample test as an ordinary job on a production system without dedicating the system to a test environment. This is not a stress test. You can use this network to verify your WSim installation.

The test network runs a WSim simulation via the VTAM application program interface (API). Only active VTAMAPPL definitions are necessary. **This is not an actual installation procedure.** Before attempting the test, install WSim as directed in *WSim User's Guide* and the WSim Program Directory.

The ITPECHO sample VTAM application program

ITPECHO is a VTAM application program supplied with WSim as a sample routine. It is not the same as WSim simulation through the VTAM API. This is a separate utility program that can be run independently of WSim. Use ITPECHO to help you learn about and plan your WSim configuration.

ITPECHO is essentially an "echo" program; it receives data from a terminal and transmits the same data back to the terminal that issued the request. Terminal types supported include 3270s (LU2) and any non-3270 devices that do not depend on any specific data stream (such as LU0).

ITPECHO is easy and flexible to use, and is ideally suited for this installation test. However, it might also become a part of regular WSim tests when an application program is not running in the network subsystem to be tested. It provides a string repetition feature that allows the terminal to request that a response of a certain length be returned. In addition, this response can be repeated consecutively, and the length can optionally be incremented with each repetition.

For complete information on the ITPECHO program itself, refer to *WSim Utilities Guide*.

WSim as a VTAM application (INSTALL1)

These directions explain how to run the WSim sample network INSTALL1, which simulates a logical unit Type 2 (LU2) through the VTAM application program interface. You can run WSim and the ITPECHO application in the same CPU. The simulated LU2 drives ITPECHO as if a real 3270 were in session. One possible real CPU configuration is shown in Figure 31 on page 282. The network definition for INSTALL1 is shown in "Sample installation network (INSTALL1)" on page 283.

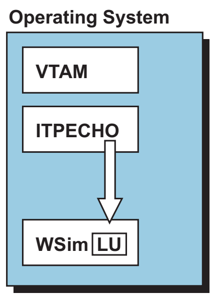


Figure 31. Installation test 1 system configuration. ITPECHO “sees” a real 3270 terminal.

Directions for an MVS system

When running WSim on MVS, you can choose to use the WSim/ISPF Interface or to invoke WSim utilities by way of JCL or EXECs. The WSim/ISPF Interface provides a panel-driven interface to the WSim utilities. Refer to *WSim User's Guide* for information on installing the WSim/ISPF Interface and *WSim Utilities Guide* for information on using the WSim/ISPF Interface.

1. Run the WSim Preprocessor using the INSTALL1 member of WSIM.SITPSAMP as the SYSIN data. If you know that a particular BIND will be required, change the DLOGMOD= operand value in the WSIMLU statement to name the wanted logon mode name before preprocessing. Refer to *WSim Utilities Guide* for sample JCL or EXECs to run the Preprocessor.

   Note: If you will be using the STL procedure, shown in “STL procedure” on page 285 instead of the message generation deck, you will need to translate the STL procedure into message generation statements using the STL Translator. Place the output from the translation into MSGDD so the message generation deck will be ready for use. See *WSim Script Guide and Reference* for more information on using the STL translator.
2. Define an “ITPECHO” application in your VTAM VTAMLST file. For example:
   ```
   ITPECHO   APPL
   ```
   You do not have to use “ITPECHO” as the APPLID (application ID) for ITPECHO. If you chose to use a different APPLID to run ITPECHO, you must also do the following actions:
   - Start ITPECHO in step 5 with the APPLID name as an execution parameter.
   - Change the sample network WSIMLU LU RESOURCE= operand value to match your APPLID.

   You might want to create a new VTAMLST member just for WSim definitions. This will become useful in the future.
3. Define the WSim simulated LU application “WSIMAPPL” in VTAMLST. For example:
   ```
   WSIMAPPL    APPL
   ```
   This definition may go into the same VTAMLST member used in step 2.
4. Activate the application IDs from steps 2 and 3 with the VTAM operator command.
5. Start ITPECHO with the application ID used in its APPL definition.
   ```
   S ITPECHO,APPLID=ITPECHO
   ```
   Refer to *WSim Utilities Guide* for a sample execution PROC.

6. Start WSim. Refer to *WSim User's Guide* for sample JCL and EXECs.
7. Enter the following WSim operator command:
   ```
   I INSTALL1,S,L
   ```
   This initializes, starts, and lists (in the SYSPRINT print file) the sample network INSTALL1.
8. Let the network run if necessary, observing message rates and WSim write-to-operator (WTO) messages. The terminal WSIMLU will log on to ITPECHO automatically, and send messages that will be echoed back to the terminal. This process will loop indefinitely.
9. End WSim with the WSim operator command:
   ```
   ZEND
   ```
   This will cancel the network and end the WSim job.

   Run the following steps if you would like to analyze the log data set.
10. Run the Loglist Utility. Refer to *WSim Utilities Guide* for sample JCL and EXECs.
11. Run the Response Time Utility. Refer to *WSim Utilities Guide* for sample JCL and EXECs.

This concludes the sample WSim simulation under MVS using the VTAM application program interface to simulate an LU2. You might want to repeat this test to gain familiarity with other WSim operator commands and to use other network definitions you created yourself.

Suggested exercise

Create a copy of network INSTALL1 named TEST1. Add another LU2 named WSIMLU2 identical to WSIMLU and change the name on the NTWRK statement to TEST1. You will also need to code PARSESS=YES on the ITPECHO APPL and WSIMAPPL APPL definitions in your VTAMLST file. This allows multiple, concurrent sessions for each of the APPL definitions. Run this new network. Both of the simulated LUs will log on to ITPECHO. You can compare the results of this run with those from the previous one.

Sample installation network (INSTALL1)

The following example shows the sample installation network, INSTALL1, and message generation deck, INSTMTXT. This script is located in the WSim sample data set as member INSTALL1 on MVS or file name INSTALL1 PREPIN under VM.

```
INSTALL1 NTWRK  HEAD='SAMPLE NETWORK 1',

***********************************************************************
* SAMPLE INSTALLATION TEST NETWORK 1:                                 *
*       VTAM API SIMULATION WITH ITPECHO                              *
***********************************************************************

                UTI=500,
                MSGTRACE=YES,
                BUFSIZE=2048,
                STLTRACE=YES,
                LOGDSPLY=BOTH,
                THKTIME=UNLOCK,
                INIT=SEC,
                OPTIONS=(DEBUG,CONRATE)

ITPECHO  PATH   INSTMTXT
```

```
**********************************************************************
*   ONE VTAM APPLICATION, ONE 3270 LU2 SESSION                       *
*   DEFAULT SCREEN SIZE IS 24 X 80                                   *
*   --> ADJUST VTAMAPPL NAME TO MATCH VTAM   <--                     *
*   --> ADJUST LU DLOGMOD TO SELECT LOGMODE  <--                     *
**********************************************************************

WSIMAPPL VTAMAPPL
WSIMLU   LU     LUTYPE=LU2,
                RESOURCE=ITPECHO,
                DLOGMOD=D4A32782     (D6327802 POSSIBLE, TOO)


         INSTMTXT MSGTXT  PAD='40'
         **********************************************************************
         *                                                                    *
         * SAMPLE MESSAGE TEXT DECK TO EXERCISE ITPECHO.                      *
         * USED BY SAMPLE NETWORKS 'INSTALL1' AND 'INSTALL2'.                 *
         *                                                                    *
         **********************************************************************
         STAY     LABEL
00001    0        IF     LOC=B+0,                    START OF TERMINAL BUFFER
                         TEXT=(WELCOME TO ITPECHO),  LOOK FOR THE 'WELCOME'
                         SCAN=YES,                   SCAN ENTIRE BUFFER
                         THEN=B-GO                   BRANCH TO 'GO' IF FOUND
00002             WAIT
00003             BRANCH LABEL=STAY       TRY AGAIN ON NEXT MESSAGE
00004    GO       WTO    (NOW LOGGED ON TO ITPECHO)


         ********** MAIN MESSAGE LOOP **********

00005    LOOP     TEXT   (THIS IS A SIMPLE MESSAGE),RESP=(SIMPLE MESSAGE)
00006    1        IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                         TEXT=RESP,   EXPECTED RESPONSE ABOVE
                         SCAN=YES,    SCAN ENTIRE BUFFER
                         THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00007             ENTER
00008             WAIT


         **** MESSAGE OF LENGTH 50

00009             TEXT   (THIS IS A 50 CHARACTER MESSAGE ),
                         LENG=50,
                         RESP=(50 CHARACTER)
00010    2        IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                         TEXT=RESP,   EXPECTED RESPONSE ABOVE
                         SCAN=YES,    SCAN ENTIRE BUFFER
                         THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00011             ENTER
00012             WAIT



         **** MESSAGE OF LENGTH 200

00013             TEXT   (THIS IS A 200 CHARACTER MESSAGE ),
                         LENG=200,
                         RESP=(200 CHARACTER)
00014    3        IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                         TEXT=RESP,   EXPECTED RESPONSE ABOVE
                         SCAN=YES,    SCAN ENTIRE BUFFER
                         THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00015             ENTER
00016             WAIT
```

```
         **** TRY PF 5 FEATURE OF ITPECHO: REQUEST 10 BYTE STRING

00017              TEXT   (10),
                          RESP=(ABCDEFGHIJ)
00018    4         IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                          TEXT=RESP,   EXPECTED RESPONSE ABOVE
                          SCAN=YES,    SCAN ENTIRE BUFFER
                          THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00019              PF5                     PRESS PF 5
00020              WAIT


         **** REQUEST 20 BYTE MESSAGE REPEATED 2 TIMES

00021              TEXT   (20,2),
                          RESP=(ABCDEFGHIJKLMNOPQRST)
00022    5         IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                          TEXT=RESP,   EXPECTED RESPONSE ABOVE
                          SCAN=YES,    SCAN ENTIRE BUFFER
                          THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00023              PF5                     PRESS PF 5
00024              WAIT


         **** REQUEST 30 BYTE MESSAGE, REPEAT 10 TIMES, INCREMENT BY 5

00025              TEXT   (30,10,5),
                          RESP=(ABCDEFGHIJKLMNOPQRST)
00026    6         IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                          TEXT=RESP,   EXPECTED RESPONSE ABOVE
                          SCAN=YES,    SCAN ENTIRE BUFFER
                          THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00027              PF5                     PRESS PF 5
00028              WAIT


         **** SEND CLEAR TO RESTORE ORIGINAL SCREEN FORMAT

00029              TEXT   (),              NO TEXT, BUT...
                          RESP=(WELCOME)   RESET THE RESP OPERAND EXPECTED
00030    7         IF     LOC=B+0,     START LOOKING AT TERMINAL BUFFER
                          TEXT=RESP,   EXPECTED RESPONSE ABOVE
                          SCAN=YES,    SCAN ENTIRE BUFFER
                          THEN=CONT    THEN CONTINUE MESSAGE GENERATION
00031              CLEAR                   PRESS CLEAR KEY
00032              WAIT


         ********* LOOP HAS SUCCESSFULLY EXECUTED, REPEAT UNTIL ZEND *****

00033              WTO    (LOOP SUCCESSFULLY EXECUTED $DSEQ,5$ TIMES)
00034              BRANCH LABEL=LOOP       LOOP AGAIN
00035              ENDTXT
```

STL procedure

The example below is an STL procedure which, when translated, performs the same functions as the preceding message generation deck, INSTMTXT. For more information about STL and the STL Translator see *WSim Script Guide and Reference*. This STL procedure is located on the installation tape in the sample data set as member STLINST on MVS or file name STLINST STLIN on VM.

```
@network
install1 ntwrk  head='Sample Network 1',

**********************************************************************
```

```
* Sample Installation Test Network 1:                             *
*       VTAM API Simulation with ITPECHO                          *
*******************************************************************

                uti=500,
                msgtrace=yes,
                stltrace=yes,
                bufsize=2048,
                logdsply=both,
                thktime=unlock,
                init=sec,
                options=(debug,conrate)

itpecho  path   instmtxt
*******************************************************************
*   One VTAM application, one 3270 LU2 session                    *
*   Default screen size is 24 x 80                                *
*   --> Adjust VTAMAPPL name to match VTAM   <--                  *
*   --> adjust LU DLOGMOD to select logmode  <--                  *
*******************************************************************

tpnsappl vtamappl
tpnslu   lu     lutype=lu2,
                resource=itpecho,
                dlogmod=d4a32782     (d6327802 possible, too)
@endnetwork

@program=installx
/*****************************************************************/
/*                                                               */
/* Sample STL program to exercise ITPECHO.  May be used by sample  */
/* networks 'INSTALL1' and 'INSTALL2'.                           */
/*                                                               */
/*****************************************************************/
INSTMTXT: msgtxt

/*---------------------------------------------------------------*/
/* Wait until the WELCOME TO ITPECHO message appears on the screen */
/* The loop is required here since an incoming SNA BIND RU will  */
/* satisfy the wait condition before the message appears.        */
/*---------------------------------------------------------------*/
do while index(screen,'WELCOME TO ITPECHO') = 0
   wait until onin index(screen,'WELCOME TO ITPECHO') > 0
end


say 'Now logged on to ITPECHO.'

loop_count = 0                     /* Initialize loop counter to 0*/
do forever                         /* Main message loop          */
   loop_count = loop_count + 1     /* Increment the loop count   */


/*------------------------------------------------------------*/
   /* Send a simple message and wait until the message SIMPLE    */
   /* MESSAGE appears before continuing.                         */
   /*------------------------------------------------------------*/
   type 'THIS IS A SIMPLE MESSAGE'
   transmit and wait until onin index(screen,'SIMPLE MESSAGE') > 0


/*------------------------------------------------------------*/
   /* Send a message of length 50 and wait until the message 50  */
   /* CHARACTER appears before continuing.                       */
   /*------------------------------------------------------------*/
   type 'THIS IS A 50 CHARACTER MESSAGE '||repeat(' ',19)
   transmit and wait until onin index(screen,'50 CHARACTER') > 0
```

```
/*-------------------------------------------------------------*/
   /* Send a message of length 200 and wait until the message 200  */
   /* CHARACTER appears before continuing.                         */
   /*-------------------------------------------------------------*/
   type 'THIS IS A 200 CHARACTER MESSAGE '||repeat(' ',169)
   transmit and wait until onin index(screen,'200 CHARACTER') > 0


/*-------------------------------------------------------------*/
   /* Try the PF5 feature of ITPECHO:  Request a 10 character      */
   /* message and wait until the message ABCDEFGHIJ appears before */
   /* continuing.                                                  */
   /*-------------------------------------------------------------*/
   type '10'
   transmit using PF5 and wait until onin index(screen,'ABCDEFGHIJ') > 0


/*-------------------------------------------------------------*/
   /* Request a 20 character message repeated 2 times and wait     */
   /* until the message ABCDEFGHIJKLMNOPQRST appears before        */
   /* continuing.                                                  */
   /*-------------------------------------------------------------*/
   type '20,2'
   transmit using PF5 and ,
            wait until onin index(screen,'ABCDEFGHIJKLMNOPQRST') > 0

/*-------------------------------------------------------------*/
   /* Request a 30 character message repeated 10 times and         */
   /* incremented by 5 each time and wait until the message        */
   /* ABCDEFGHIJKLMNOPQRST appears before continuing.              */
   /*-------------------------------------------------------------*/
   type '30,10,5'
   transmit using PF5 and ,
            wait until onin index(screen,'ABCDEFGHIJKLMNOPQRST') > 0


/*-------------------------------------------------------------*/
   /* Send a CLEAR to restore the original screen and wait until   */
   /* WELCOME reappears on the screen before continuing.           */
   /*-------------------------------------------------------------*/
   transmit using clear and wait until onin index(screen,'WELCOME') > 0

   say 'Loop successfully executed 'char(loop_count)' times.'
end /* Main message loop */
endtxt /* End of program */
```

See “INSTALL1 loglist” on page 375 for sample loglist output.

Chapter 24. Message scripting examples

The following sections provide examples which you might find useful in understanding network definition and the message generation and logic testing processes. Discussions of the examples point out device and application dependencies. You can use the examples as prototypes for coding the final network and message generation decks, consider the specific configuration and application requirements of your system.

Note: STL procedures are provided for the following message generation deck examples. The message generation deck examples are intended to represent what you would code had you written them using WSim message generation deck statements. The STL procedures are what you would code if you were to write the functionally equivalent deck as an STL procedure. Note that although the two decks would be functionally equivalent, they might not necessarily contain the same statements. For more information about STL, see *WSim Script Guide and Reference*.

WSim as an application

In the following network, WSim will simulate a primary LU acting as an application program. Secondary LUs (simulated by WSim or real terminal users) can log on to this application.

This is a simple application. It is essentially an “echo” program. It receives data and transmits it to the terminal that issued the request. This application should not be confused with the sample VTAM application program supplied with WSim, ITPECHO. That is a separate application program that can be run independently of WSim. The following script is a WSim script which is simulating an application. You can note, however, various similarities between the two, as this script was patterned after ITPECHO and has many of the same functions. This script is an example of how WSim might be used to prototype applications before their development.

Additional features of this simulated application can be achieved by a PF key, data stream content, or a combination of a PF key and data stream content. Specifically, the sequences with special features are Enter, PF5, PF9, Clear, and Logoff. Since the functions of this script are patterned after ITPECHO, see the chapter about ITPECHO in *WSim Utilities Guide* for a detailed description.

The network is a VTAMAPPL configuration, which requires no additional hardware. The only change that you need to make to your system is the addition of a VTAM APPL definition in your system VTAMLST similar to the following example:

```
WSIMECHO APPL PARSESS=YES
```

Coding PARSESS=YES is not required, but allows parallel sessions if wanted.

Network definition

The following example shows the network definition for the resources that drive the application. Note that the LU has ten primary half-sessions defined. This determines that ten secondary logical units can log on to this application. This script (network definition and message generation decks) is located in the sample data set as member ECHO.

```
ECHO     NTWRK UTI=0,BUFSIZE=32000,CHAINING=AUTO,
               STLTRACE=YES,MSGTRACE=YES,MLOG=YES

PLU      PATH  PLUECHO
VA1      VTAMAPPL APPLID=WSIMECHO
PLU      LU    LUTYPE=LU0,PATH=(PLU),MAXSESS=(10,0),INIT=SEC
```

Message generation deck

Following are the message generation decks needed for this network.

```
         PLUECHO  MSGTXT
         *--------------------------------------------------------------------*
         * BUILD THE REPEAT STRING FOR ALL SESSIONS INTO NETWORK SAVE AREA 1.  *
         *--------------------------------------------------------------------*
00001             IF   LOC=NSW1,WHEN=IMMED,THEN=B-START   BRANCH ALREADY SETUP
00002             SETSW NSW1=ON                           SET SW FOR NEXT TIME
00003             DATASAVE AREA=N1,TEXT=(ABCDEFGHIJKLMNOPQRSTUVWXYZ)
00004    BUILDN1  DATASAVE AREA=N1,TEXT=($RECALL,N1$$RECALL,N1$)
00005             SET  DC1=LENG(N1)
00006             IF   LOC=DC1,TEXT=32000,COND=LT,WHEN=IMMED,THEN=B-BUILDN1

         *--------------------------------------------------------------------*
         * FORMAT THE SLU 3270 SCREEN.                                        *
         *--------------------------------------------------------------------*
00007    START    DATASAVE AREA=1,TEXT=()                 CLEAR MSG RECV AREA
00008             CALL NAME=FIRSTMSG                      FORMAT 3270 SCREEN

         *--------------------------------------------------------------------*
         * ACTIVATE THE IFS TO SAVE THE RU WHEN DATA IS RECEIVED AND RESET THE *
         * WAIT INDICATOR.                                                    *
         *--------------------------------------------------------------------*
00009    0        IF   LOC=RU+0,TEXT=('00'),COND=GE,THEN=ESAVERU,STATUS=HOLD
00010    1        IF   LOC=RU+0,TEXT=('00'),COND=GE,THEN=CONT,STATUS=HOLD

         *--------------------------------------------------------------------*
         * TOP OF PLU ECHO LOOP                                               *
         *--------------------------------------------------------------------*
00011    ECHOLOOP DATASAVE AREA=3,TEXT=($RECALL,1$)       SAVE LAST MSG FOR PF9
00012             DATASAVE AREA=1,TEXT=()                 CLEAR MSG AREA
00013             WAIT

         *--------------------------------------------------------------------*
         * PROCESS BASED ON AID BYTE RECEIVED AND SAVED IN DEVICE SAVE AREA 1. *
         *--------------------------------------------------------------------*

00014             IF   LOC=1+0,TEXT=('F9'),WHEN=IMMED,THEN=B-PF9    PF9
00015             IF   LOC=1+0,TEXT=('C9'),WHEN=IMMED,THEN=B-PF9    PF21
00016             BRANCH LABEL=NOTPF9
00017    PF9      DATASAVE AREA=1,TEXT=($RECALL,3$)       RECALL LAST MESSAGE
         NOTPF9   LABEL
00018             IF   LOC=1+0,TEXT=('6D'),WHEN=IMMED,THEN=B-CLEAR  CLEAR
00019             IF   LOC=1+0,TEXT=('F5'),WHEN=IMMED,THEN=B-PF5    PF5
00020             IF   LOC=1+0,TEXT=('C5'),WHEN=IMMED,THEN=B-PF5    PF17

         ENTER    LABEL                                   DEFAULT ACTION
00021             IF   LOC=1+6,TEXT=(LOGOFF),WHEN=IMMED,THEN=B-LOGOFF
00022             IF   LOC=1+6,TEXT=(LOGOFF),WHEN=IMMED,THEN=B-LOGOFF
00023             TEXT ('F1'),                            WRITE COMMAND
                       ('C3'),                            WCC
                       ('11D160'),                        SBA (15,01)
                       ('13'),                            IC
                       ('114040'),                        SBA (01,01)
                       ('3C4E7F00'),                      RA  (12,80) '00'X
```

```
                     ('114040'),                         SBA (01,01)
                     ('124040'),                         EUA (01,01)
                     ($RECALL,1+6$)                      DATA RECEIVED
00024           BRANCH LABEL=ECHOLOOP
00025  LOGOFF   CMND COMMAND=UNBIND                      UNBIND SESSION
00026           DEACT IFS=ALL                            DEACTIVATE IFS
00027           BRANCH LABEL=START

       *--------------------------------------------------------------------*
       * REPEAT DATA BASED ON NUMBERS RECEIVED                              *
       *                                                                    *
       * MSG_LENGTH<,REPEAT_COUNT<,INCREMENT>>                              *
       *  0-32000      1-32000      0-32000                                 *
       *    DC2          DC3          DC4                                   *
       *--------------------------------------------------------------------*
       PF5      LABEL
00028           SET  DC1=LENG(1)                         SAVE LENGTH OF RU
00029           IF   LOC=DC1,TEXT=23,COND=GT,            IF LENGTH OF RU MORE
                     WHEN=IMMED,THEN=B-PF5ERROR          THAN 23, THEN ERROR
00030           DATASAVE AREA=4,TEXT=($RECALL,1+6$)      SAVE DATA FROM RU
00031           SET  DC1=LENG(4)                         LENGTH OF RU DATA
00032           IF   LOC=DC1,TEXT=17,COND=EQ,            IF LENGTH OF RU DATA
                     WHEN=IMMED,THEN=B-NOPAD             17, PROCESS AS IS
00033           SET  DC2=17,DC2=-DC1                     DC2 IS LENGTH OF DATA
00034           DATASAVE AREA=4,TEXT=($RECALL,4$$DUP, ,DC2$) PAD WITH BLANKS
       NOPAD    LABEL
00035           SET  DC1=0,DC2=0,DC3=1,DC4=0             DEFAULT VALUES
00036           CALL LABEL=GETNUM                        GET MSG_LENGTH
00037           SET  DC2=DC7                             MSG_LENGTH
00038           IF   LOC=SW7,WHEN=IMMED,THEN=B-PF5ERROR VALUE IN ERROR
00039           IF   LOC=DC2,TEXT=32000,COND=GT,         IF MSG_LENGTH IS TOO
                     WHEN=IMMED,THEN=B-PF5ERROR          LARGE, THEN ERROR
00040           IF   LOC=4+DC1,TEXT=(,),WHEN=IMMED,      CHECK FOR SECOND VALUE
                     ELSE=B-DOPF5                        (REPEAT_COUNT)
00041           SET  DC1=+1                              LOOK PAST COMMA
00042           CALL LABEL=GETNUM                        GET REPEAT_COUNT
00043           SET  DC3=DC7                             REPEAT_COUNT
00044           IF   LOC=SW7,WHEN=IMMED,THEN=B-PF5ERROR VALUE IN ERROR
00045           IF   LOC=DC3,TEXT=32000,COND=GT,         IF REPEAT_COUNT IS
                     WHEN=IMMED,THEN=B-PF5ERROR          TOO LARGE, THEN ERROR
00046           IF   LOC=DC3,TEXT=0,WHEN=IMMED,          IF REPEAT_COUNT IS
                     THEN=B-PF5ERROR                     ZERO, ERROR
00047           IF   LOC=4+DC1,TEXT=(,),WHEN=IMMED,      CHECK FOR THIRD VALUE
                     ELSE=B-DOPF5                        (INCREMENT)
00048           SET  DC1=+1                              LOOK PAST COMMA
00049           CALL LABEL=GETNUM                        GET INCREMENT VALUE
00050           SET  DC4=DC7                             INCREMENT
00051           IF   LOC=SW7,WHEN=IMMED,THEN=B-PF5ERROR VALUE IN ERROR
00052           IF   LOC=DC4,TEXT=32000,COND=GT,         IF INCREMENT IS TOO
                     WHEN=IMMED,THEN=B-PF5ERROR          LARGE, THEN ERROR
00053  DOPF5    TEXT ('F1C311D160131140403C4E7F00114040124040'),
                     ($RECALL,N1,DC2$)                   GENERATE MESSAGE
00054           SET  DC3=-1,DC2=+DC4                     UPDATE COUNTERS
00055           IF   LOC=DC3,TEXT=0,WHEN=IMMED,          CHECK IF ALL REPETI-
                     THEN=B-ECHOLOOP                     TIONS COMPLETE
00056           RH   CDI=OFF
00057           BRANCH LABEL=DOPF5
00058  PF5ERROR TEXT ('F1C311D160131140403C4E7F00114040124040'),
                     (INVALID LENGTH REQUEST)            GENERATE ERROR MSG
00059           BRANCH LABEL=ECHOLOOP
       *--------------------------------------------------------------------*
       * GETNUM AND NUMLOOP PARSES THROUGH THE DATA RECEIVED WITH THE PF5   *
       * AID BYTE CHECKING FOR A VALID STRING.                              *
       *--------------------------------------------------------------------*
00060  GETNUM   SET  DC5=0,DC6=0,DC7=0
00061           SETSW SW7=OFF
```

```
        NUMLOOP  LABEL
00062            IF   LOC=4+DC1,TEXT=(,),WHEN=IMMED,THEN=RETURN
00063            IF   LOC=4+DC1,TEXT=( ),WHEN=IMMED,THEN=RETURN
00064            IF   LOC=4+DC1,TEXT=(0),COND=LT,WHEN=IMMED,THEN=B-NUMERR
00065            IF   LOC=4+DC1,TEXT=(9),COND=GT,WHEN=IMMED,THEN=B-NUMERR
00066            SET  DC7=*10,DC6=(E,4+DC1,1),DC7=+DC6,DC1=+1,DC5=+1
00067            IF   LOC=DC5,TEXT=5,WHEN=IMMED,THEN=RETURN
00068            BRANCH LABEL=NUMLOOP
00069   NUMERR   SETSW SW7=ON                            ERROR IN VALUE
00070            RETURN

00071   CLEAR    CALL NAME=FIRSTMSG
00072            BRANCH LABEL=ECHOLOOP
00073            ENDTXT

        FIRSTMSG MSGTXT
00001            TEXT ('F5'),                            ERASE/WRITE COMMAND
                      ('C7'),                            WCC
                      ('114E7F'),                        SBA (12,80)
                      ('1DF8'),                          SF  PROTECTED H.I.
                      (WELCOME TO ITPECHO.),
                      ('1D60'),                          SF  PROTECTED
                      ( ENTER=ECHO   CLEAR=RESTORE   ),
                      (5=STRING REPT   9=REPEAT),
                      ('115050'),                        SBA (14,01)
                      (ENTER DATA TO ECHO BELOW:),
                      ('11D15F'),                        SBA (14,80)
                      ('1D40'),                          SF  UNPROTECTED
                      ('13'),                            IC
                      ('115D7F'),                        SBA (24,80)
                      ('1DF0')                           SF  PROTECTED
00002            RETURN
00003            ENDTXT


        SAVERU   MSGTXT
00001            DATASAVE AREA=2,LOC=*,LENG=32000        SAVE DATA RECEIVED
00002            DATASAVE AREA=1,TEXT=($RECALL,1$$RECALL,2$)  APPEND IT
00003            DATASAVE AREA=2,TEXT=()                 FREE WORK SAVE AREA
00004            RETURN
00005            ENDTXT
```

STL procedure

The following example is an STL procedure which, when translated, can be used instead of the preceding message generation deck for the application. This STL procedure is in the sample data set as member STLECHO.

```
@program=echoit
string  shared repeat_string           /* This is the repeat string */
bit     shared not_first_pass          /* Set if logged on "again"  */
constant alphabet 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' /* For repeat string  */
string  unshared work_area             /* Work area used by SAVERU  */
bit     unshared error?                /* Set if bad number given   */
integer unshared repeat_count          /* The numbers provided, by  */
integer unshared msg_length            /* the user, if the repeat   */
integer unshared increment             /* function is requested     */


pluecho: msgtxt

/*******************************************************************/
/* Build the repeat string for all sessions, if 1st logon          */
/*******************************************************************/

if not_first_pass = off then do        /* If 1st logon, then        */
   not_first_pass = on                 /* Indicate already setup    */
   repeat_string = alphabet            /* Initialize repeat string  */
   do while length(repeat_string) < 32000 /* Build 32000 char string */
      repeat_string = repeat_string||repeat_string /* Keep building  */
   end /* Build 32000 char string */
end /* If 1st logon, then */

do forever                           /* Repeat forever            */
   message_area = ''                 /* Clear this to get started */
   call firstmsg                     /* Format the 3270 SLU screen*/

                                     /* When data is received,    */
   onin substr(ru,1,1) >= '00'x then do /* save the RU            */
      work_area = RU                 /* Save the RU for later use */
      message_area = message_area||work_area /* Append to message  */
      work_area = ''                 /* Clear work area for later */
   end
   logoff = off                      /* Indicate LOGOFF not hit   */
   do while logoff = off             /* Do till LOGOFF found      */
      hold_last_msg = message_area   /* Hold this for PF9 check   */
      message_area = ''              /* Clear for next message    */
      wait until onin substr(ru,1,1) >= '00'x /* Wait till data is */
                                     /* received                  */

      /***********************************************************/
      /* Process based on the AID byte just received             */
      /***********************************************************/

      if substr(message_area,1,1) = 'F9'x | ,
         substr(message_area,1,1) = 'C9'x then /* PF9 or PF21     */
         message_area = hold_last_msg   /* Restore last message   */
      select
         when substr(message_area,1,1) = '6D'x then /* Clear      */
            call firstmsg            /* Refresh the screen        */
      /***********************************************************/
      /* Repeat data if PF5 or 17 pressed based on the following: */
      /*                                                          */
      /* MSG_LENGTH<,REPEAT_COUNT<,INCREMENT>>                    */
      /*  0-32000       1-32000    0-32000                        */
      /*                                                          */
      /***********************************************************/
         when substr(message_area,1,1) = 'F5'x | ,
              substr(message_area,1,1) = 'C5'x then do /* PF5 or PF17*/
            call getnums             /* Get each data field       */
            if error? then do        /* Bad number specified      */
               type 'F1C311D160131140403C4E7F00114040124040'x|| ,
                    'INVALID LENGTH REQUEST' /* Give error message  */
                    iterate          /* Redisplay the screen      */
            end /* Bad number specified */
            else do while repeat_count > 0 /* Input is Okay!        */
                                     /* Display repeat_count times*/
               type 'F1C311D160131140403C4E7F00114040124040'x|| ,
                    substr(repeat_string,1,msg_length) /* Set msg   */
               transmit              /* Send the message          */

               repeat_count = repeat_count - 1 /* One less time to go*/
               msg_length = msg_length + increment /* Increment it   */
               setrh off(cdi)        /* Turn CDI off in the RH    */
            end /* Input is Okay! */
         end /* PF5 or PF17 */
         when substr(message_area,7,6) = 'LOGOFF' | ,
              substr(message_area,7,6) = 'logoff' then do /* LOGOFF  */
            snacmnd(unbind)          /* Send an UNBIND            */
            deact all io ons         /* End all ONIN tests        */
            logoff = on              /* Finished with ECHO screen */
         end /* LOGOFF */
         otherwise                   /* Handle ENTER or PF9/21    */
            type 'F1'x|| ,           /* WRITE command             */
                 'C3'x|| ,           /* WCC                       */
                 '11D160'x|| ,       /* SBA (15,01)               */
                 '13'x|| ,              /* IC                        */
                 '114040'x|| ,          /* SBA (01,01)               */
                 '3C4E7F00'x|| ,        /* RA (12,80) '00'X          */
                 '114040'x|| ,          /* SBA (01,01)               */
                 '124040'x|| ,          /* EUA (01,01)               */
                 substr(message_area,7) /* The ECHOd data            */
      end /* select */
   end /* Do till LOGOFF found */
end /* Repeat forever */
endtxt /* End of PLUDECK */

firstmsg: msgtxt                        /* Format the 3270 SLU screen*/
type 'F5'x|| ,                          /* ERASE/WRITE command       */
     'C7'x|| ,                          /* WCC                       */
     '114E7F'x|| ,                      /* SBA (12,80)               */
     '1DF8'x|| ,                        /* SF Protected H.I.         */
     'WELCOME TO ITPECHO.'|| ,
     '1D60'x|| ,                        /* SF Protected              */
     ' ENTER=ECHO   CLEAR=RESTORE   5=STRING REPT   9=REPEAT'|| ,
     '115050'x|| ,                      /* SBA (14,01)               */
     'ENTER DATA TO ECHO BELOW:'|| ,
     '11D15F'x|| ,                      /* SBA (14,80)               */
     '1D40'x|| ,                        /* SF UnProtected            */
     '13'x|| ,                          /* IC                        */
     '115D7F'x|| ,                      /* SBA (24,80)               */
     '1DF0'x                            /* SF Protected              */
return
endtxt /* End of FIRSTMSG */


getnums: msgtxt                         /* Parse the user input      */

user_input = substr(message_area,7)||',' /* Get the user input,      */
                                        /* Append , for processing of*/
                                        /* last number               */
the_length = 0                          /* Initialize each of these  */
repeat_count = 1                        /* fields to their default   */
increment = 0                           /* values                    */
error? = off                            /* Used to indicate an error */
pass = 1                                /* Indicates field processing*/
num_chars = 0                           /* Length of number          */
number = 0                              /* The number to be converted*/
do i=1 to length(user_input)            /* Do all input              */
   character = substr(user_input,i,1)   /* Strip off 1 character     */
   select                               /* Validate the character    */
      when character >= '0' & ,
           character <= '9' then do     /* If a number is specified  */
         if num_chars = 0 then          /* If 1st char of number     */
            number = e2d(character)     /* Save it                   */
         else                           /* Not 1st char of number    */
            number = number*10+e2d(character) /* Build the number    */
         num_chars = num_chars + 1      /* Bump number of chars found*/
      end /* If a number is specified */

      when character = ',' then         /* If a comma is found       */
           if number > 32000 | ,        /* If number is too big or   */
              num_chars > 5 | ,         /* has too many characters or*/
              (num_chars = 0 & ,        /* is null, .ie no characters*/
              i = length(user_input)) then do /* specified, then     */
              error? = on               /* Bad number specified      */
              return                    /* Return to caller          */
           end /* If number is too big or ... */
           else do                      /* Number is okay so far     */
              select                    /* See which number this is  */
                 when pass = 1 then     /* If 1st number, then       */
                    msg_length = number /* Set 'msg_length'          */
                 when pass = 2 then     /* If 2nd number, then       */
                    if number = 0 then  /* If it is 0, then          */
                       error? = on      /* It is an error            */
                    else                /* If 2nd number is okay     */
                       repeat_count = number /* Set 'repeat_count'   */
                 when pass = 3 then     /* If 3rd number, then       */
                    increment = number  /* Set 'increment'           */
                 otherwise              /* If none of above, then    */
                    error? = on         /* It is an error            */
              end /* See which number this is */
              if error? then            /* If an error above         */
                 return                 /* Return to the caller      */
              number = 0                /* Reset the number          */
              pass = pass + 1           /* Increment the pass count  */
              num_chars = 0             /* Reset the num chars also  */
           end /* Number is okay so far */
      when character = ' ' then nop     /* Blanks are okay           */
      otherwise do                      /* A bad character is found  */
         error? = on                    /* Indicate an error         */
         return                         /* Return to the caller      */
      end /* A bad character is found */
   end /* Validate the character */
end /* Do all input; the +1 is to ... */
return
endtxt /* GETNUMS */
```

A portion of the Loglist Utility output that can be run with this sample can be found in "WSIM application loglist" on page 380.

TCP/IP examples

The following examples present sample networks and WSim scripts for various TCP/IP examples.

The logical configuration for a TCP/IP network is shown in Figure 32.

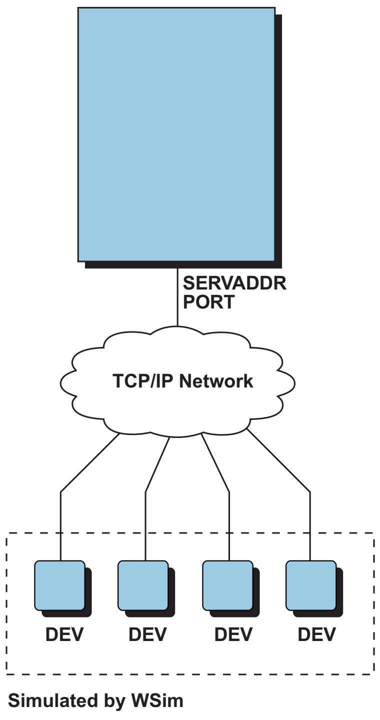


Figure 32. Sample TCP/IP network (logical configuration)

The physical configuration for a TCP/IP network is shown in Figure 33.

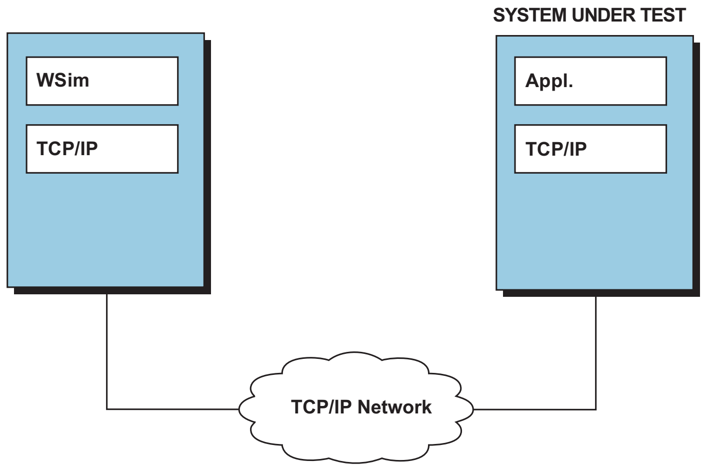


Figure 33. Sample TCP/IP network (physical configuration)

The WSim host processor does not require a specific hardware interface into the TCP/IP network. Any available interface (3745, 3172, and so on) is acceptable.

Telnet 3270 example

In the following network, WSim simulates four 3270 terminals connecting to a host via Telnet 3270 sessions. WSim manages the connections to the host while the user script interacts with the host application. Connection to the host normally causes the simulated terminal to receive a host application logon screen. The message generation deck receives control after connection to the host and waits for the logon screen. Upon receiving the logon screen, the message deck specifies the user ID and logon password of an account on the host. The message deck then waits for a ready prompt and logs off.

Network definition statements

```
*-----------------------------------------------------------------------
TN3270   NTWRK HEAD='TCPIP 3270 TEST NETWORK',
               CONRATE=YES,
               OPTIONS=(DEBUG,MONCMND),
               ITIME=1,
               MSGTRACE=YES,
               LOGDSPLY=BOTH,
               BUFSIZE=2048,
               THKTIME=UNLOCK,
               INIT=PRI,
               RSTATS=YES,
               UTI=100,
               SEQ=0,
               SERVADDR=9.67.6.1
*-----------------------------------------------------------------------
SOMEHOST PATH  SOMEHOST
*-----------------------------------------------------------------------
TCONN1   TCPIP
DEV11    DEV    MAXCALL=5,THKTIME=IMMED
DEV12    DEV    DELAY=F(10)
DEV13    DEV    THKTIME=UNLOCK
DEV14    DEV
```

Message generation deck

```
*--------------------------------------------------------------------
SOMEHOST MSGTXT
         CALL   NAME=WAITSCRN
         WTO    ($DEVID$ ESTABLISHED TCPIP SESSION, LOGGING ON)
         SET    NC1=NSEQ
         SET    NSEQ=+1
         TEXT   ($UTBL,IDS,NC1$),MORE=YES
         TAB
         TEXT   ($UTBL,PWS,NC1$)
         ENTER
         CALL   NAME=WAITREDY
         WTO    (GOT READY PROMPT)
         TEXT   (LOGOFF)
         ENTER
         CALL   NAME=WAITLOGF
         WTO    (GOT LOGOFF MESSAGE)
         OPCMND (ZEND)
         ENDTXT
*--------------------------------------------------------------------
WAITSCRN MSGTXT
0        IF     WHEN=IN,LOC=B+0,TEXT=(VM/ESA ONLINE),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*--------------------------------------------------------------------
WAITREDY MSGTXT
0        IF     LOC=B+0,TEXT=(Ready),SCAN=YES,THEN=CONT
         WAIT
         ENDTXT
*--------------------------------------------------------------------
WAITLOGF MSGTXT
0        IF     LOC=B+0,TEXT=(Logoff),SCAN=YES,THEN=CONT
         WAIT
         CLEAR
         ENDTXT
*--------------------------------------------------------------------
IDS      MSGUTBL  (USER1),(USER2),(USER3),(USER4)
PWS      MSGUTBL  (PASSWORD),(PASSWORD),(PASSWORD),(PASSWORD)
```

STL procedures

```
allocate nextnum 'NSEQ'
integer  shared nextid

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


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


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

Sample WSim script for a Telnet 3270E simulation

Below is an example of a WSim 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 WSim 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.                        *
*                                                                      *
************************************************************************

*----------------------------------------------------------------------*
* 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 WSim Script Guide       *
* and Reference for details on valid operands.                         *
*----------------------------------------------------------------------*
TCONN1   TCPIP   TNPORT=23
DEV11    DEV     TYPE=TN3270E,RESOURCE=WSIM01,PATH=(HOST1)
DEV12    DEV     TYPE=TN3270E,FUNCTS=(0,1,2),PATH=(HOST1)
DEV13    DEV     TYPE=TN3270P,RESOURCE=WSIM02,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, WSim 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, WSim  *
* 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
```

Sample Telnet Line Mode Network Virtual Terminal message generation deck

Below 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

Below 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
```

File Transfer Protocol (FTP) example

In the following network, WSim simulates a sample network with one File Transfer Protocol (FTP) device connecting to a host via a TCP/IP FTP session. WSim manages the connections to the host while the user script transfers files with the host application.

Network definition statement

```
* File Transfer Protocol                                             *
**********************************************************************
* Network Configuration:  File Transfer Protocol simulation          *
*                                                                    *
* Description:  This WSim script will simulate one FTP client user   *
*               connecting to a server, putting a file to that server, *
*               then retrieving the same file.  The SERVADDR operand *
*               specifies the IP dotted address of the host to which *
*               the device 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.                      *
*                                                                    *
**********************************************************************

*--------------------------------------------------------------------*
* Network statement operands.                                        *
*--------------------------------------------------------------------*
FTP      NTWRK HEAD='Model FTP Network', * Set the title line
               CONRATE=YES,         * Print message rates on console
               OPTIONS=(DEBUG,MONCMND), * 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
               TCPNAME=TCPIP,       * <== Default name of the local
*                                   *     TCPIP virtual machine
               SERVADDR=9.67.43.62  * <== Remote Server
*                                   *     to which you will connect
```

```
*--------------------------------------------------------------------*
1        UTBL (Hes the invisible man), * First Record Data
              (Catch him if you can)   * Second Record Data
*--------------------------------------------------------------------*
FTPDECK  PATH  FTPDECK
*--------------------------------------------------------------------*
* Define the file data to be simulated                               *
*--------------------------------------------------------------------*
FILE1    FILE  TYPE=E,              * File Data is EBCDIC
               RECFM=F,             * Fixed Record Format
               RECLEN=80,           * Record Length is 80
               DATA=1               * Get data from UTBL 1
*--------------------------------------------------------------------*
* 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 WSim    *
* Script Guide and Reference for details on valid operands.          *
*--------------------------------------------------------------------*
TCONN1   TCPIP  PATH=(FTPDECK)
DEV010   DEV  TYPE=FTP
```

Message generation deck

```
FTPDECK  MSGTXT
*--------------------------------------------------------------------*
* The Message Generation deck.                                       *
*                                                                    *
* Generates FTP commands for file transfer.                          *
* user     - identifies the user to the server.                      *
* pass     - supplies a password to the server.                      *
* ebcdic   - set file transfer type to EBCDIC.                       *
* mode     - specifies file transfer mode.                           *
* sendsite - disables automatic transmission of SITE subcommand.     *
* put      - transfers the FILE data defined by WSim FILE statement.  *
* get      - transfers the data from a file on the server.           *
*                                                                    *
*--------------------------------------------------------------------*
*  Wait for connect
1        IF    WHEN=IN,LOC=D+0,COND=GE,
               TEXT=('00'x),THEN=CONT     /* Wait for next message    */
         WAIT
LOOP     TEXT  (user testuser)            /* <== Enter User ID        */
         TEXT  (pass testpass)            /* <== Enter Password       */
         TEXT  (ebcdic)                   /* EBCDIC Translation       */
         TEXT  (mode B)                   /* Block Mode               */
         TEXT  (sendsite)                 /* Automatic SITE Off       */
         TEXT  (put FILE1 TEST.FILE1)     /* Put File1                */
         TEXT  (get TEST.FILE1)           /* Get File1 back           */
4        IF    WHEN=IN,
               LOCTEXT=($RECALL,B+0,3$),COND=EQ,TEXT=(250),THEN=E-SUCC
         BRANCH LABEL=NOTSU
SUCC     WTO   (Get successful.)
         RETURN
NOTSU    LABEL
5        IF    WHEN=IN,
               LOCTEXT=($RECALL,B+0,3$),COND=EQ,TEXT=(426),THEN=E-FAIL
         BRANCH LABEL=QUIT
FAIL     WTO   (Get failed.)
         RETURN
QUIT     TEXT  (quit)
* Following quiesce will keep automatic reconnect from
* occurring.  Release the DEV to recycle through the
* script.  Remove the QUIESCE to
* automatically recycle after 30 seconds.
         QUIESCE
         BRANCH  LABEL=LOOP
         ENDTXT
```

STL procedure

```
ftpdeck:  msgtxt
/*--------------------------------------------------------------------*
* The Message Generation deck.                                        *
*                                                                     *
* Generates FTP commands for file transfer.                           *
* user     - identifies the user to the server.                       *
* pass     - supplies a password to the server.                       *
* ebcdic   - set file transfer type to EBCDIC.                        *
* mode     - specifies file transfer mode.                            *
* sendsite - disables automatic transmission of SITE subcommand.      *
* put      - transfers the FILE data defined by WSim FILE statement.  *
* get      - transfers the data from a file on the server.            *
*                                                                     *
*--------------------------------------------------------------------*/
/*  Wait for connect                                                  */
  wait until onin
 Do forever
  type 'user testuser'                  /* <== Enter User ID          */
  transmit
  type 'pass  testpass'                 /* <== Enter Password         */
  transmit
  type 'ebcdic'                         /* EBCDIC Transfer Type       */
  transmit
  type 'mode B'                         /* Block Mode                 */
  transmit
  type 'sendsite'                       /* Automatic SITE Off         */
  transmit
  type 'put FILE1 TEST.FILE1'           /* Put File1                  */
  transmit
  s: onin substr(buffer,1,3)='250' then   /* Check message number     */
    say 'Get successful.'
  f: onin substr(buffer,1,3)='426' then   /* Check message number     */
    say 'Get failed.'
  type 'get TEST.FILE1'                 /* Get File1 back             */
  transmit
  deact s,f
  type 'quit'                           /* Drop Connection            */
/* Following quiesce will keep automatic reconnect from */
/* occurring.  Release the DEV to recycle through the   */
/* script.  Replace the quiesce with a transmit to      */
/* automatically recycle after 30 seconds.              */
  quiesce
 end
 endtxt
```

Simple TCP Client example

In the following network, WSim simulates a sample network with one Simple TCP Client device connecting to a host via a TCP/IP STCP session.

Network definition statement

```
* Simple TCP Client
***********************************************************************
* Network Configuration:  Simple TCP Client simulation                *
*                                                                     *
* Description:  This WSim 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 a sequence of   *
*                   two carriage return/line feed (CR/LF) sequences;  *
*                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.                        *
*                                                                      *
************************************************************************

*----------------------------------------------------------------------*
* Network statement operands.                                          *
*----------------------------------------------------------------------*
STCP     NTWRK HEAD='Model STCP Network', * Set the title line
               CONRATE=YES,         * Print message rates on console
               OPTIONS=(MONCMND),   * 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
               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 WSim      *
* Script Guide and Reference for details on valid operands.            *
*----------------------------------------------------------------------*
TCONN1   TCPIP
DEV010   DEV  TYPE=STCP,           * Simple TCP Client
              PORT=5555,           * Server Port for connection
              SERVADDR=9.67.43.62, * Server IP Address for connection
              PATH=(STCPDECK)      * Path Sequence for this DEV
```

Message generation deck

```
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 ASCII
         DATASAVE AREA=1,
                  TEXT=('000102031A091A7F1A1A1A0B0C0D0E0F')+  * 00-0F
                       ('101112131A1A081A18191A1A1C1D1E1F')+  * 10-1F
                       ('1A1A1C1A1A0A171B1A1A1A1A1A050607')+  * 20-2F
                       ('1A1A161A1A1E1A041A1A1A1A14151A1A')+  * 30-3F
                       ('20A6E180EB909FE2AB8B9B2E3C282B7C')+  * 40-4F
                       ('26A9AA9CDBA599E3A89E21242A293B5E')+  * 50-5F
                       ('2D2FDFDC9ADDDE989DACBA2C255F3E3F')+  * 60-6F
                       ('D78894B0B1B2FCD6FB603A2340273D22')+  * 70-7F
                       ('F861626364656667686996A4F3AFAEC5')+  * 80-8F
                       ('8C6A6B6C6D6E6F7071729787CE93F1FE')+  * 90-9F
                       ('C87E737475767778797AEFC0DA5BF2F9')+  * A0-AF
                       ('B5B6FDB7B8B9E6BBBCBD8DD9BF5DD8C4')+  * B0-BF
                       ('7B414243444546474849CBCABEE8ECED')+  * C0-CF
                       ('7D4A4B4C4D4E4F505152A1ADF5F4A38F')+  * D0-DF
                       ('5CE7535455565758595AA0858EE9E4D1')+  * E0-EF
                       ('30313233343536373839B3F7F0FAA7FF')   * F0-FF
* Set length of input data each time data is received
1        IF       WHEN=IN,STATUS=HOLD,SNASCOPE=ALL,
                  LOC=NC1,COND=GE,TEXT=0,
                  THEN=E-SAVELENG
         BRANCH   LABEL=TRANLOOP
SAVELENG DATASAVE AREA=2,
                  TEXT=($RECALL,B+0$)
         SET      DC2=LENG(2)
         RETURN
* Loop sending transaction
TRANLOOP DATASAVE AREA=2,
                  FUNCTION=TRANSLATE,
                  TEXT=(Sample Transaction from $ID,8$),
                  TABLEO=($RECALL,1$)
         DATASAVE AREA=3,
                  FUNCTION=TRANSLATE,
                  TEXT=(Sample Transaction line 2),
                  TABLEO=($RECALL,1$)
         TEXT     ($RECALL,2$'0D0A'$RECALL,3$'0D0A0D0A')
2        IF       WHEN=IN,STATUS=HOLD,SNASCOPE=ALL,
                  LOC=DC2,TEXT=0,COND=EQ,
                  THEN=CONT
         WAIT
         DEACT    IFS=(2)
         BRANCH   LABEL=TRANLOOP
         ENDTXT
```

STL procedure

```
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 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 */
/* Set length of input data each time data is received */
  onin then ilen=length(buffer)
/* Loop sending transaction                            */
 Do forever
  type translate('Sample Transaction from' id(),ebc2asc)||'0d0a'x||,
       translate('Sample Transaction line 2',ebc2asc)||'0d0a0d0a'x;
  transmit and wait until onin ilen=0      /* Send in the transaction  */
                                          /* and wait for connection  */
                                          /* to close                 */
 end
 endtxt
```

CPI-C example with single-instance transaction programs

In the following network, WSim simulates a sample network with two LUs and three single-instance transaction programs (TPs): one client TP and two server TPs.

The logical configuration for this network is shown in Figure 34.

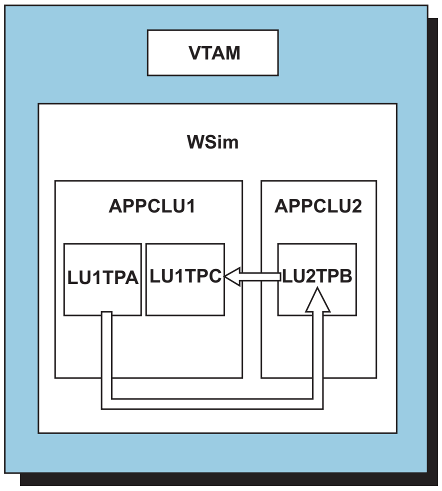


Figure 34. Sample CPI-C network with single-instance TPs (logical configuration)

Network definition statements

```
***********************************************************************
*                                                                     *
*  Network Configuration: CPI-C Transaction Program simulation        *
*                                                                     *
*  Network Description:                                               *
*                                                                     *
*  This WSim Network simulates a CPI-C client Transaction Program     *
*  communicating with a CPI-C server Transaction Program.  The server *
*  TP then communicates with a second CPI-C server Transaction        *
*  Program.  There are two simulated LU's.  The client TP has 1       *
*  initial instance, and the server TP's are started by incoming      *
*  attach requests.  A different level of CPI-C tracing is requested  *
*  by each of the TP's.                                               *
*                                                                     *
*  Network Diagram:                                                   *
*                                                                     *
*    APPCLU: APPCLU1                     APPCLU: APPCLU2              *
*    +-----------------+                 +-----------------+          *
*    |                 |                 |                 |          *
*    |   +---------+   |  conversation 1 |   +---------+   |          *
*    |   | TP: TPA ===========================>        |   |          *
*    |   +---------+   |   mode=#inter   |   |         |   |          *
*    |                 |                 |   | TP: TPB |   |          *
*    |   +---------+   |  conversation 2 |   |         |   |          *
*    |   | TP: TPC <===========================        |   |          *
*    |   +---------+   |   mode=#batch   |   +---------+   |          *
*    |                 |                 |                 |          *
*    +-----------------+                 +-----------------+          *
*                                                                     *
*  Notes:                                                             *
*                                                                     *
*   1. Conversation 1 uses mode name #INTER.  The conversation        *
*      sync-level is "none".                                          *
*                                                                     *
*   2. Conversation 2 uses mode name #BATCH.  The conversation        *
*      sync-level is "confirm".                                       *
*                                                                     *
*   3. The VTAM APPLID names used in this network (APPCLU1 and        *
*      APPCLU2) must be defined to VTAM and must be active prior to   *
*      running this network.                                          *
*                                                                     *
*   4. The CNOS operand on the APPCLU1 definition is only required     *
*      if you want to control the number of sessions.  If the operand  *
*      is not specified, sessions will be managed by WSim as required  *
*      by the simulation.                                              *
*                                                                      *
************************************************************************

CPICSMP1 NTWRK HEAD='CPI-C NETWORK SAMPLE 1',
               TPSTATS=YES        * Keep stats for all TP instances    *

*----------------------------------------------------------------------*
* Transaction Program paths.                                           *
*----------------------------------------------------------------------*
LU1TPA   PATH   TPADECK           * Define the LU1TPA TP path          *
LU1TPC   PATH   TPCDECK           * Define the LU1TPC TP path          *
LU2TPB   PATH   TPBDECK           * Define the LU1TPB TP path          *

*----------------------------------------------------------------------*
* Network resources.                                                   *
*----------------------------------------------------------------------*
APPCLU1  APPCLU APPLID=APPCLU1,   * APPC LU; VTAM APPLID is APPCLU1    *
                SIDEINFO=((DESTNAME=LU2TPB,MODENAME=#INTER,
                           TPNAME=TPB,LUNAME=APPCLU2)),
*                                 * Define destination name LU2TPB     *
                CNOS=((LUNAME=APPCLU2,MODENAME=#INTER,
                       SESSIONS=4,CWL=2,CWP=2))
*                                 * Specify CNOS values                *
LU1TPA   TP     TPNAME=TPA,       * TP name is TPA                     *
                PATH=(LU1TPA),    * TP is defined by LU1TPA path       *
                TPTYPE=CLIENT,    * TP type is CLIENT                  *
                INSTANCE=(1,1),   * 1 initial TP instance              *
                CPITRACE=MSG      * Log CPI-C trace messages           *
LU1TPC   TP     TPNAME=TPC,       * TP name is TPC                     *
                PATH=(LU1TPC),    * TP is defined by LU1TPC path       *
                TPTYPE=SERVER,    * TP type is SERVER                  *
                INSTANCE=(0,1),   * No initial TP instances            *
                CPITRACE=VERBEND  * Log CPI-C verb completions         *

APPCLU2  APPCLU APPLID=APPCLU2,   * APPC LU; VTAM APPLID is APPCLU2    *
                SIDEINFO=((DESTNAME=LU1TPC,MODENAME=#BATCH,TPNAME=TPC,
                           LUNAME=APPCLU1))
*                                 * Define destination name LU1TPC     *
LU2TPB   TP     TPNAME=TPB,       * TP name is TPC                     *
                PATH=(LU2TPB),    * TP is defined by LU2TPB path       *
                TPTYPE=SERVER,    * TP type is SERVER                  *
                INSTANCE=(0,1),   * No initial TP instances            *
                CPITRACE=VERB     * Log CPI-C verbs issued & complete  *
```

Message generation decks

```
TPADECK  MSGTXT
************************************************************************
* Message deck defining the TPA Transaction Program.                   *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=destination name
*  3=send buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=send length
*  dc3=request-to-send received
*
         WTO  (Transaction Program $TPID$ starting.)
*
```

```
************************************************************************
*       Initialize and allocate a conversation with TPB.              *
*
*       Set the symbolic destination name to "LU2TPB".
        DATASAVE AREA=2,TEXT=(LU2TPB)
*
*       Initialize a conversation with TPB.
        CMINIT(1,2,DC1)
*
*       Allocate a conversation with TPB; the sync-level defaults
*       to "none", and the conversation type defaults to "mapped".
        CMALLC(1,DC1)
*
*       Setup the send buffer and length.
        DATASAVE AREA=3,TEXT=(Data sent from A to B.)  * Send buffer
        SET  DC2=LENG(3)                                * Send length
*
*       Send data to TPB.
        CMSEND(1,3,DC2,DC3,DC1)
*
*       Deallocate the conversation with TPB.
        CMDEAL(1,DC1)
*
        WTO  (Transaction Program $TPID$ complete.)
*
        ENDTXT

TPCDECK  MSGTXT
************************************************************************
* Message deck defining the TPC Transaction Program.                  *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*
        WTO  (Transaction Program $TPID$ starting.)
*
*       Accept the conversation with TPB.
        CMACCP(1,DC1)
*
*       Setup requested length for receive.
        SET  DC2=100
*
*       Receive data from TPB.
        CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
*       Confirm the data was received.
        CMCFMD(1,DC1)
*
        WTO  (Transaction program $TPID$ complete.)
        WTO  (Simulation complete.)
*
        ENDTXT

TPBDECK  MSGTXT
************************************************************************
* Message deck defining the TPB Transaction Program.                  *
************************************************************************
```

```
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*  3=destination name
*  4=send buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*  dc7=send length
*  dc8=sync-level
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPA.
         CMACCP(1,DC1)
*
*        Setup requested length for receive.
         SET  DC2=100
*
*        Receive data from TPA.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
************************************************************************
*        Initialize and allocate a conversation with TPC.              *
*
*        Set the symbolic destination name to "LU1TPC".
         DATASAVE AREA=3,TEXT=(LU1TPC)
*
*        Initialize a conversation with TPC.
         CMINIT(1,3,DC1)
*
*        Set the conversation sync-level to "confirm".
         SET  DC8=1                                     * Sync-level
         CMSSL(1,DC8,DC1)
*
*        Allocate a conversation with TPC; the conversation type
*        defaults to "mapped".
         CMALLC(1,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=4,TEXT=(Data sent from B to C.)  * Send buffer
         SET  DC7=LENG(4)                               * Send length
*
*        Send data to TPC.
         CMSEND(1,4,DC7,DC6,DC1)
*
*        Deallocate the conversation with TPC.
         CMDEAL(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT
```

STL procedures

```
@PROGRAM=CPICSMP1

@include cpicvar
@include cpiccon

TPADECK: MSGTXT
```

```
/*********************************************************************
* STL deck defining the TPA Transaction Program.                     *
*********************************************************************/

say 'Transaction Program' tpid() 'starting.'

/*******************************************************************/
/* Initialize and allocate a conversation with TPB.                */

/* Set the symbolic destination name to "LU2TPB". */
sym_dest_name='LU2TPB'

/* Initialize a conversation with TPB. */
CMINIT(conversation_ID,,
       sym_dest_name,,
       return_code)

/* Allocate the conversation; the sync-level defaults to "none", */
/* and the conversation type defaults to "mapped".               */
CMALLC(conversation_ID,,
       return_code)

/* Setup the send buffer and length. */
send_buffer='Data sent from A to B.'   /* send buffer for mapped conv */
/*send_buffer='0018'x||'Data sent from A to B.'   buff for basic conv */
send_length=length(send_buffer)

/* Send data to TPB. */
CMSEND(conversation_ID,,
       send_buffer,,
       send_length,,
       request_to_send_received,,
       return_code)

/* Deallocate the conversation with TPB. */
CMDEAL(conversation_ID,,
       return_code)

say 'Transaction Program' tpid() 'complete.'

ENDTXT

TPCDECK: MSGTXT
/*********************************************************************
* STL deck defining the TPC Transaction Program.                     *
*********************************************************************/

say 'Transaction Program' tpid() 'starting.'

/* Accept the conversation with TPB. */
CMACCP(conversation_ID,,
       return_code)

/* Set requested length for receive. */
requested_length=100

/* Receive data from TPB. */
CMRCV(conversation_ID,,
      receive_buffer,,
      requested_length,,
      data_received,,
      received_length,,
      status_received,,
      request_to_send_received,,
      return_code)

/* Confirm the data was received. */
```

```
CMCFMD(conversation_ID,,
       return_code)

say 'Transaction Program' tpid() 'complete.'
say 'Simulation complete.'

ENDTXT

TPBDECK: MSGTXT
/*********************************************************************
* STL deck defining the TPB Transaction Program.                     *
*********************************************************************/

say 'Transaction Program' tpid() 'starting.'

/* Accept the conversation with TPA. */
CMACCP(conversation_ID,,
       return_code)

/* Set requested length for receive. */
requested_length=100

/* Receive data from TPA. */
CMRCV(conversation_ID,,
      receive_buffer,,
      requested_length,,
      data_received,,
      received_length,,
      status_received,,
      request_to_send_received,,
      return_code)

/* Confirm the data was received. */
CMCFMD(conversation_ID,,
       return_code)

/*******************************************************************/
/* Initialize and allocate a conversation with TPC.                */

/* Set the symbolic destination name to "LU1TPC". */
sym_dest_name='LU1TPC'

/* Initialize a conversation with TPC. */
CMINIT(conversation_ID,,
       sym_dest_name,,
       return_code)

/* Set the conversation sync-level to "confirm". */
CMSSL(conversation_ID,,
      cm_confirm,,
      return_code)

/* Allocate the conversation; the conversation type defaults to */
/* "mapped".                                                    */
CMALLC(conversation_ID,,
       return_code)

/* Setup the send buffer and length. */
send_buffer='Data sent from B to C.'
send_length=22

/* Send data to TPC. */
CMSEND(conversation_ID,,
       send_buffer,,
       send_length,,
       request_to_send_received,,
       return_code)
```

```
/* Deallocate the conversation with TPC. */
CMDEAL(conversation_ID,,
       return_code)

say 'Transaction Program' tpid() 'complete.'

ENDTXT
```

CPI-C example with multiple-instance transaction programs

In the following network, WSim simulates a sample with two LUs and two transaction programs (TPs): a client TP and a server TP. The client TP has three instances, which WSim starts .01 seconds apart. Each client TP instance sends an attach request to the server TP; each attache request arriving at the server TP activates an instance of the server TP.

The logical configuration for this network is shown in Figure 35.

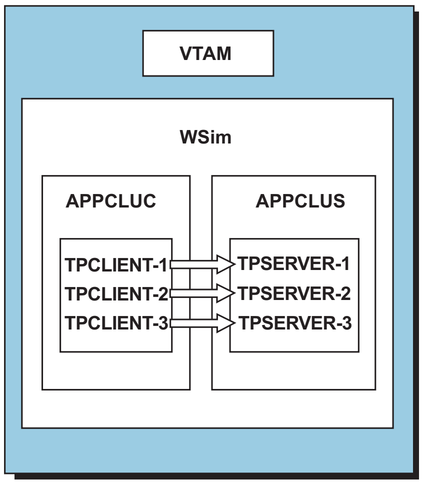


Figure 35. Sample CPI-C network with multiple instance TPs (logical configuration)

Network definition statements

```
**************************************************************************
*                                                                        *
*  Network Configuration: CPI-C Transaction Program simulation          *
*                                                                        *
*  Network Description:                                                  *
*                                                                        *
*  This WSim Network simulates a CPI-C client Transaction Program       *
*  communicating with a CPI-C server Transaction Program.  The client   *
*  TP has 3 instances that are started with a .01 second stagger.       *
*  The server TP is started by incoming attach requests.  Statistics    *
*  are only kept for the first instance of each TP.  CPI-C verb         *
*  tracing is requested.                                                 *
*                                                                        *
*  Network Diagram:                                                      *
*                                                                        *
*    APPCLU: APPLID1                         APPCLU: APPLID2            *
*    +--------------------+                  +--------------------+     *
*    |                    |                  |                    |     *
*    |  +--------------+  |                  |  +--------------+  |     *
*    |  |              |  |  conversation 1  |  |              |  |     *
*    |  | TP: CLIENT-1 ========================> TP: SERVER-1  |  |     *
*    |  |              |  |                  |  |              |  |     *
*    | |              | |  conversation 2  |  |              | |    *
*    | | TP: CLIENT-2 =========================> TP: SERVER-2 | |    *
*    | |              | |                  |  |              | |    *
*    | |              | |  conversation 3     |              | |    *
*    | | TP: CLIENT-3 =========================> TP: SERVER-3 | |    *
*    | |              | |                  |  |              | |    *
*    |  +--------------+ |                  |  +--------------+ |    *
*    |                   |                  |                   |    *
*    +--------------------+                 +--------------------+    *
*                                                                     *
*  Notes:                                                             *
*                                                                     *
*   1. Conversations 1 thru 3 use mode name #INTER.  The conversation *
*      sync-level is "confirm".                                       *
*                                                                     *
*   2. Since there are 3 client TP instances and the server is started *
*      by attach requests, three server TP instances are required     *
*      to accept each of the 3 incoming conversations.                *
*                                                                     *
*   3. The VTAM APPLID names used in this network (APPLID1 and        *
*      APPLID2) must be defined to VTAM and must be active prior to   *
*      running this network.                                          *
*                                                                     *
***********************************************************************

CPICSMP2 NTWRK HEAD='CPI-C NETWORK SAMPLE 2',
               CPITRACE=VERB,     * Trace CPI-C verbs in the log       *
               TPSTATS=NO         * Keep stats for 1st TP instance only*

*---------------------------------------------------------------------*
* Network-wide Side Information Table.                                *
*---------------------------------------------------------------------*
         SIDEINFO
         SIDEENT DESTNAME=SERVER,MODENAME=WSIMLU62,
                 LUNAME=APPLID2,TPNAME=TPSERVER
         SIDEEND

*---------------------------------------------------------------------*
* Transaction Program paths.                                          *
*---------------------------------------------------------------------*
CLIENT PATH   CLIENT              * Define the CLIENT TP path         *
SERVER PATH   SERVER              * Define the SERVER TP path         *

*---------------------------------------------------------------------*
* Network resources.                                                  *
*---------------------------------------------------------------------*
APPCLUC  APPCLU APPLID=APPLID1    * Client APPC LU                    *
TPCLIENT TP     PATH=(CLIENT),    * TP name is TPCLIENT, path is CLIENT*
                TPTYPE=CLIENT,    * TP type is client                 *
                INSTANCE=(3,3),   * 3 initial & 3 concurrent instances *
                TPSTIME=1         * Stagger start-up by .01 seconds   *
APPCLUS  APPCLU APPLID=APPLID2    * Server APPC LU                    *
TPSERVER TP     PATH=(SERVER),    * TP name is TPSERVER, path is SERVER*
                TPTYPE=SERVER,    * TP type is server                 *
                INSTANCE=(0,1)    * No initial TP instances           *
```

Message generation decks

```
CLIENT: MSGTXT
***********************************************************************
* Message deck defining the TPCLIENT Transaction Program.             *
***********************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=destination name
*  3=send buffer
```

```
*
* Device counter usage:
*  dc1=return code
*  dc2=sync-level
*  dc3=send length
*  dc4=request-to-send received
*
         WTO  (Transaction Program $TPID$ starting.)
*
************************************************************************
*  Initialize and allocate a conversation with TPSERVER.               *
*
*        Set the symbolic destination name to "SERVER".
         DATASAVE AREA=2,TEXT=(SERVER)
*
*        Initialize a conversation with TPSERVER.
         CMINIT(1,2,DC1)
*
*        Set the conversation sync-level to "confirm".
         SET  DC2=1                                    * Sync-level
         CMSSL(1,DC2,DC1)
*
*        Allocate a conversation with TPSERVER.
         CMALLC(1,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=3,TEXT=(LU $appcluid$, TP $tpid$$tpinstno$)+
                 (: Data sent from CLIENT to SERVER.)   * Send data
         SET  DC3=LENG(3)                              * Send length
*
*        Send data to TPSERVER.
         CMSEND(1,3,DC3,DC4,DC1)
*
*        Deallocate the conversation with TPSERVER.
         CMDEAL(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT

SERVER: MSGTXT
************************************************************************
* Message deck defining the TPSERVER Transaction Program.              *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPCLIENT.
         CMACCP(1,DC1)
*
*        Set requested length for receive.
         SET  DC2=100
*
*        Receive data from TPCLIENT.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
```

```
*
*        Confirm the data was received.
         CMCFMD(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
         IF $TPINSTNO$='-00003' THEN
          WTO (Simulation complete.)
*
         ENDTXT
```

STL procedures

```
@PROGRAM=CPICSMP2

@include cpicvar
@include cpiccon

CLIENT: MSGTXT
/***********************************************************************
* STL deck defining the TPCLIENT Transaction Program.                  *
***********************************************************************/

say 'Transaction Program' tpid() 'starting.'

/**********************************************************************/
/* Initialize and allocate a conversation with TPSERVER.              */

/* Set the symbolic destination name to "SERVER". */
sym_dest_name = 'SERVER'

/* Initialize a conversation with TPSERVER. */
CMINIT (conversation_ID, sym_dest_name, return_code)

/* Set the conversation sync-level to "confirm". */
CMSSL  (conversation_ID, cm_confirm, return_code)

/* Allocate the conversation; the conversation type defaults to */
/* "mapped".                                                    */
CMALLC (conversation_ID, return_code)

/* Setup the send buffer and length. */
send_buffer = 'LU' appcluid()', TP' tpid()tpinstno()||,
              ': Data sent from A to B.'
send_length = length(send_buffer)

/* Send data to TPSERVER. */
CMSEND (conversation_ID, send_buffer, send_length,,
        request_to_send_received, return_code)

/* Deallocate the conversation with TPSERVER. */
CMDEAL (conversation_ID, return_code)

say 'Transaction Program' tpid() 'complete.'

ENDTXT

SERVER: MSGTXT
/***********************************************************************
* STL deck defining the TPSERVER Transaction Program.                  *
***********************************************************************/

say 'Transaction Program' tpid() 'starting.'

/* Accept the conversation with TPCLIENT. */
CMACCP (conversation_ID, return_code)

/* Receive data from TPCLIENT. */
```

```
CMRCV  (conversation_ID, receive_buffer, 100, data_received,,
        received_length, status_received,,
        request_to_send_received, return_code)

/* Confirm the data was received. */
CMCFMD (conversation_ID, return_code)

say 'Transaction Program' tpid() 'complete.'
if tpinstno()='-00003' then
 say 'Simulation complete.'

ENDTXT
```

Chapter 25. AVMON example

AVMON (AVailability MONitor) is a WSim network designed to monitor the availability and performance of real network subsystems (NetView® and TSO). This sample monitors only NetView and TSO, but you can add more subsystems by modifying the script as described later in this section.

Availability monitoring

AVMON monitors performance by logging the WSim LU on to a subsystem, periodically sending a message, and sensing when the subsystem becomes unavailable. When the LU detects unavailability, it sends a message to the operator console alerting the operator to the problem.

Performance monitoring

WSim tracks the time for returning a response from the subsystem and reports whenever a user-specified threshold is exceeded. Furthermore, by using the WSim Response Time Utility the entire run's performance statistics can be compiled into a single report.

Automated operations

AVMON can be modified to perform operator functions when it senses a resource needs restarting. For a complete description of this feature, refer to "AVMON as an automated operator" on page 322.

AVMON processing description

There are three levels of logical units (LUs) in the AVMON network: the network controller (NETCTRL), the subsystem controllers (NETVCTRL and TSOCTRL), and the subsystem terminal pools (NV01-03 and TS01-03 in this example). Figure 36 on page 320 illustrates the relationship of each layer. Each level performs a different function as described in the figure and in the subsequent paragraphs.

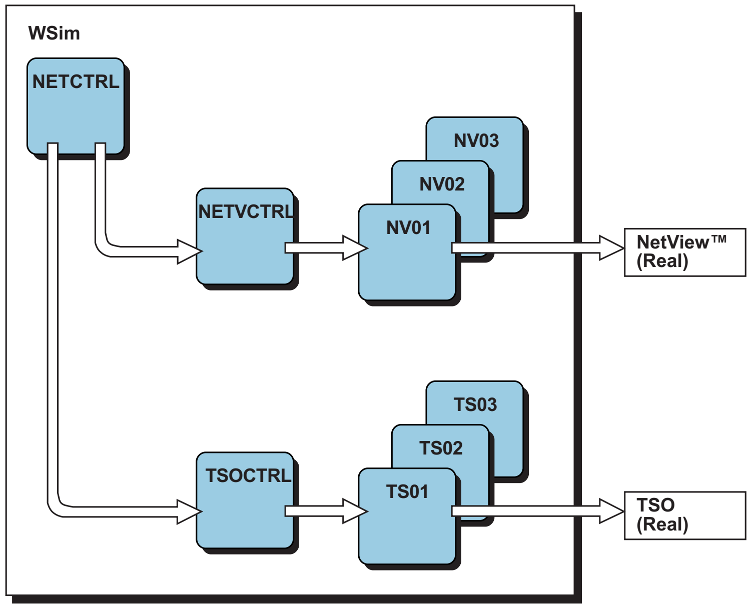


Figure 36. How AVMON is structured

Note: This figure shows the relationship between WSim resources and the subsystems under test. The physical configuration is not depicted here.

Network controller level

The NETCTRL LU uses the ACTRLNET message generation deck to act as a network coordinator. It has the following functions:

- Providing initial coordination so the network will maintain synchronization.
- Starting the monitoring process at a designated start-of-day time and stopping the process at a specified end-of-day time.
- Issuing messages to the operator console when requested to do so by another LU or the AVMON operator or when providing a summary of availability report.

Subsystem controller level

The NETVCTRL and TSOCTRL LUs use the AMONNETV and AMONTSO message generation decks to act as controllers of the subsystem status pools. They monitor the session between the currently active status pool LU and the subsystem being monitored. If the pool LU fails to indicate that all is well (SW1=ON), the subsystem controller LU takes these actions:

- Quiesces the pool LU whose session has recently failed
- Releases the next LU in the status pool from its quiesced state
- Restarts the network and refreshes the pool, if the pool becomes exhausted. It uses the following WSim commands:
  - OPCMND (C AVMON)
  - OPCMND (I AVMON)
  - OPCMND (S AVMON).

Subsystem terminal pool level

The subsystem terminal pool LUs initiate a session with the subsystem to be monitored. In this example, only NetView and TSO are being monitored. You can monitor more subsystems by copying and modifying the AMONTSO and ACHKTSO decks to match your requirements.

For NetView

The NV01, NV02, and NV03 LUs use the ALOGNETV and ACHKNETV message generation decks to log on to a NetView operator console and check the status of network resources using the VTAM [D NET,NONE,ID=*resource*] command. The value supplied for *resource* is user-determined and you may define it by modifying the deck ACHKNETV at the locations indicated by comments.

When a D NET,NONE,ID=*resource* command is issued to a NetView console, one of the following three responses may occur:

IST486I	NAME=*nodename*, STATUS=*status* (Pre-VTAM 3.2)
IST486I	CURRENT STATE=*cstate*, DESIRED STATE=*dstate*
IST088I	DISPLAY FAILED- NODE NAME INVALID OR INACTIVE
IST453I	*parameter* PARAMETER INVALID

A request is made to NETCTRL to issue a message for each one of these responses. A message is issued for response IST486I only if the status is not active.

For TSO

The TSO1, TSO2, and TSO3 LUs use the ALOGTSO and ACHKTSO message generation decks to log on to TSO and monitor TSO availability and performance.

Modifying AVMON for other subsystem monitoring

For each additional subsystem to be monitored for AVMON, you must make eight changes to the original AVMON network:

1. Code a logon deck for the new subsystem status pool LUs to establish a session with the subsystem to be monitored. Note that the DELAY=F180 on the NTWRK statement will affect the logon process, so code DELAY=CANCEL on any IF statements in the logon deck.
2. Copy the deck AMONTSO and change its name to match the new subsystem. Study the script and modify it to meet the requirements of the new subsystem.
3. Copy the deck ACHKTSO and change its name to match the new subsystem. Study the script and modify it to meet the requirements of the new subsystem.
4. Locate this statement in the deck ACTRLNET:
   ```
   OPCMND (A TSOCTRL,RELEASE)
   ```
   Copy that statement and change the name TSOCTRL to the name of your new subsystem controller LU.
5. Provide another PATH statement with the new message generation deck you modelled after ACHKTSO.
6. Provide a network configuration definition for the subsystem-level controller you are adding. You can model it after the TSO subsystem-level controller configuration statement. Code an FRSTTXT statement naming the new deck that you modelled after AMONTSO.
7. Provide a pool of subsystem status LUs under the subsystem controller. You can model them after the TSO pool. Code an FRSTTXT statement with the name of your subsystem logon deck and a PATH= statement referring to the new path statement.

8. Look in the deck ACTRLNET for the section labeled “PUTAV”. Update the WTO and LOG statements to reflect your new subsystems.

AVMON as an automated operator

The example of AVMON provided in this chapter is a passive monitoring tool. It detects problems in the network, alerts the operator, but takes no action to correct the problem. With only minor changes, AVMON can be made to perform just as an operator would and correct problems automatically.

Automated operator requirements

To make AVMON an automated operator, you need to provide the following information:

- WSim logic and message generation statements to take the action required to correct the problem
- A method of delivering an operator command to the host operating system.

The WSim logic to recognize a resource problem is already in place. The message generation deck ACHKNETV, executed by the LU name NV01, is continually monitoring the availability of the network resources that you are interested in. The STATUS=*status* (or CURRENT STATE=*cstate*) field of the VTAM message IST486I (returned in response to the [D NET,NONE,ID=*resource*] sent by NV01) tells how VTAM views the status of the resource. The immediate IF 8 in ACHKNETV is looking for ACTIV, and if anything else is returned then there is a problem and AVMON signals the operator. It is from this point that you will make AVMON an automated operator.

The method of delivering an operator command to the host operating system is not in place. It is up to you to provide this. The example that follows uses the IBM program product Operator Communications Control Facility (MVS/OCCF, program number 5665-288) to allow operator commands to be entered from a NetView operator. You can use any product that provides a similar function.

An example of an AVMON automated operator

Assume you want AVMON to restart TSO whenever NV01 receives a STATUS=CONCT (or CURRENT STATE=CONCT) from its [D NET,NONE,ID=TSO] command. CONCT means that TSO is “connectable”, which means that it is known by VTAM but not started. Using MVS/OCCF, you must make some changes.

The NetView operator that NV01 uses to monitor system resources must log on to MVS/OCCF. Find the section of ACHKNETV that looks like the following sample:

```
TEXT (AUTOWRAP YES)
ENTER
```

Next, add these two statements immediately after the ENTER:

```
TEXT (OCCF /QLOGON)
ENTER
```

Note: This example assumes the term “OCCF” is a recognizable NetView command.

The restart function is provided immediately after IF 8. This is what this section looks like before the changes:

```
   8      IF TEXT=(ACTIV),LOC=U+100,WHEN=IMMED,SCAN=50,
             THEN=B-NOMSG

          DATASAVE AREA=N+0,
             TEXT=(+$RECALL,U+60,16$ $RECALL,U+14$ $RECALL,U+10,8$)

          BRANCH LABEL=SIGNALSP
```

And this is how it would look after it has been tailored to include automated operations:

```
   8      IF TEXT=(ACTIV),LOC=U+100,WHEN=IMMED,SCAN=50,
             THEN=B-NOMSG

   9      IF TEXT=(TSO),LOC=U+60,SCAN=20,WHEN=IMMED,
             ELSE=B-OTHER

   10     IF TEXT=(CONCT),LOC=U+100,WHEN=IMMED,SCAN=50,
             ELSE=B-OTHER

          TEXT (OCCF S TSO)
          ENTER
          STOP
          DATASAVE AREA=N+0,
             TEXT=(ATTEMPT RESTART OF TSO BY AVMON AT ),
             ($RECALL,U+10,8$)

          BRANCH LABEL=SIGNALSP
  OTHER   DATASAVE AREA=N+0,
             TEXT=(+$RECALL,U+60,16$ $RECALL,U+14$ $RECALL,U+10,8$)

          BRANCH LABEL=SIGNALSP
```

The following changes were made to the message generation deck ACHKNETV to perform the automated restart of TSO whenever CONCT is observed:

- Immediate IF 9 was added to see if the resource status was for TSO. If it wasn't, then processing would branch to OTHER and a normal special message would be sent to the operator.
- Immediate IF 10 was added to see if the status was CONCT. If it wasn't, then processing would branch to OTHER and a normal special message would be sent to the operator.
- A message is generated and sent to the NetView operator with the MVS/OCCF interface with the TEXT/ENTER/STOP sequence. The term “OCCF” preceding the operator command “S TSO” is included because in this example, MVS/OCCF is used.
- A special message is generated and sent to the operator console by the DATASAVE and BRANCH LABEL=SIGNALSP.

Generating a summary report with the WSim Loglist Utility

During the monitoring run, AVMON writes log records to the log data set. You can use the Loglist Utility to format these log records into a concise report showing the activity of the day.

The following Loglist Utility parameters configure the loglist into the final report. You may want to modify these parameters to meet your specific requirements. Figure 37 on page 325 is an example of a Loglist Utility report generated from the log data set of a 2-hour AVMON run. The loglist of an AVMON run shows a brief history of the network activity from AVMON's point of view. Many of AVMON's features were used during the run represented in Figure 37 on page 325, which includes:

- Response time monitoring. AVMON detected many instances of TSO response times longer than the maximum time allowed.
- Availability monitoring. AVMON detected the status of ITPECHO as “CONCT”. Monitoring of the resource ITPECHO was accomplished by modifying the message generation deck ACHKNETV. After AVMON notified the operator three times, the operator restarted ITPECHO.
- Availability status reports. AVMON produces a summary report of the total time of availability for the resources being monitored.
- Releasing additional terminals from the pool. The TSO terminal named TSO-01 lost contact with TSO. AVMON sensed this and released another from the TSO pool. Monitoring of TSO then continued.
- Restart of the network when a pool is depleted. The TSO terminal pool was exhausted because TSO went down (notice the records that show TSO STATUS=CONCT). The Operator restarted TSO and AVMON restarted its network to refresh the pool. Monitoring continued.
- Automatic end-of-day stop of AVMON. AVMON stopped WSim when a specified end-of-day time was reached.

Loglist Utility run parameters

You can enter the following parameters from the console or provide them in a data set referenced on the SYSIN statement of the Loglist Utility job stream.

- LOG
- NOCNSL
- NOFMT SHORT
- RUN
- END.

For more information concerning the use of the Loglist utility, refer to *WSim Utilities Guide*.

```
NETWORK  APPCLU/TCPIP/  DEV/LU/TP        START    STOP     READY  RECORD  DATA USER
 NAME    VTAMAPPL NAME  NAME             TIME     TIME     TIME    TYPE   LENG DATA
AVMON         APPLNET  NETCTRL-1      04541899 0096196  11000000  LOG       1 00
AVMON         APPLNET  NETCTRL-1      04541899 0096196  11000000  LOG      42 00   ***  BEGIN MONITORING - 4:54:18, 07/14/02
AVMON         APPLN1   NV01-1         04543705 0096196  11000000  LOG      37 00   +NETVIEW IS NOW RESPONDING - 4:54:37
AVMON         APPLT1   TSO1-1         04545158 0096196  11000000  LOG      33 00   +TSO IS NOW RESPONDING - 4:54:51
AVMON         APPLT1   NETCTRL-1      05095710 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 05:09:57
AVMON         APPLT1   NETCTRL-1      05215948 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 05:21:59
AVMON         APPLT1   NETCTRL-1      05224433 0096196  11000000  LOG      60 00   +ITPECHO CK ECHO STATUS= CONCT    05:22:44
AVMON         APPLT1   NETCTRL-1      05255448 0096196  11000000  LOG      60 00   +ITPECHO CK ECHO STATUS= CONCT    05:25:54
AVMON         APPLT1   NETCTRL-1      05290965 0096196  11000000  LOG      60 00   +ITPECHO CK ECHO STATUS= CONCT    05:29:09
AVMON         APPLT1   NETCTRL-1      05464936 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 05:46:49
AVMON         APPLT1   NETCTRL-1      05542095 0096196  11000000  LOG      60 00   -TSO-01 IS NOT  RESPONDING - 5:54:20
AVMON         APPLT1   NETCTRL-1      05543007 0096196  11000000  LOG      43 00   +AVAILABILITY WITHIN 060 MINUTES -  5:54:34
AVMON         APPLT1   NETCTRL-1      05543007 0096196  11000000  LOG      20 00   +NETVIEW 060 TSO 057
AVMON         APPLT2   TSO2-1         05563860 0096196  11000000  LOG      33 00   +TSO IS NOW RESPONDING - 5:56:38
AVMON         APPLNET  NETCTRL-1      05564384 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 05:56:43
AVMON         APPLNET  NETCTRL-1      06084372 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 06:08:43
AVMON         APPLNET  NETCTRL-1      06114271 0096196  11000000  LOG      60 00   -TSO HAS BAD RESPONSE TIME AT 06:11:42
AVMON         APPLNET  NETCTRL-1      06310374 0096196  11000000  LOG      60 00   +TSO  CHECK TSO   STATUS=CONCT 6:31:03
AVMON         APPLNET  NETCTRL-1      06340903 0096196  11000000  LOG      60 00   +TSO  CHECK TSO   STATUS=CONCT 6:34:09
AVMON         APPLNET  NETCTRL-1      06352124 0096196  11000000  LOG      60 00   -TSO-02 IS NOT RESPONDING -  6:35:21
AVMON         APPLNET  NETCTRL-1      06371918 0096196  11000000  LOG      60 00   +TSO  CHECK TSO   STATUS=CONCT 6:37:19
AVMON         APPLNET  NETCTRL-1      06402133 0096196  11000000  LOG      60 00   -TSO-03 IS NOT RESPONDING -  6:40:21
AVMON         APPLNET  NETCTRL-1      06422164 0096196  11000000  LOG      43 00   +AVAILABILITY WITHIN 102 MINUTES -  6:42:21
AVMON         APPLNET  NETCTRL-1      06422164 0096196  11000000  LOG      20 00   +NETVIEW 012 TSO 096
AVMON         APPLNET  NETCTRL-1      06422164 0096196  11000000  LOG      60 00   *** TSO POOL EXHAUSTED, REINIT AVMON -  6:42:21
AVMON         APPLNET  NETCTRL-1      06422164 0096196  11000000  LOG      60 00   *** RESTART OF AVMON BY TSOCTRL -  6:42:21
AVMON         APPLNET  NETCTRL-1      06423352 0096196  11000000  LOG       1 00
AVMON         APPLNET  NETCTRL-1      06423352 0096196  11000000  LOG      42 00   ***  BEGIN MONITORING -  6:42:33, 07/14/02
AVMON         APPLN1   NV01-1         06424929 0096196  11000000  LOG      37 00   +NETVIEW IS NOW RESPONDING - 6:42:49
AVMON         APPLT1   TSO1-1         06443307 0096196  11000000  LOG      33 00   +TSO IS NOW RESPONDING - 6:44:33
AVMON         APPLNET  NETCTRL-1      06445690 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:44:56
AVMON         APPLNET  NETCTRL-1      06473708 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:47:37
AVMON         APPLNET  NETCTRL-1      06504186 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:50:41
AVMON         APPLNET  NETCTRL-1      06533928 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:53:39
AVMON         APPLNET  NETCTRL-1      06563697 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:56:36
AVMON         APPLNET  NETCTRL-1      06594064 0096196  11000000  LOG      60 00   -TSO HAS BAD RESOPNSE TIME AT 06:59:40
AVMON         APPLNET  NETCTRL-1      07004408 0096196  11000000  LOG      43 00   +AVAILABILITY WITHIN 018  MINUTES - 7:00:44
AVMON         APPLNET  NETCTRL-1      07004408 0096196  11000000  LOG      20 00   +NETVIEW 018 TSO 018
AVMON         APPLNET  NETCTRL-1      07004408 0096196  11000000  LOG      43 00   ***  END OF MONITORING -  7:00:44  07/14/02
```

Figure 37. Loglist report of AVMON run

Generating a summary report with the Response Time Utility

The response time utility uses the information in the log data set to compile a set of reports that give a clear picture of the response times observed in the monitoring run.

You can use the following response time utility parameters to provide a set of response time reports. You might want to modify these parameters to meet your specific requirements.

```
PROCESS ACTUAL
TGRAPH  INTERVAL=600,INCR=2,THRESH=50
GRAPH   2
PERCENT 90,95,99
REPORT  LEVEL=TERM,
        SUMMARY=(NOLIST,NOGRAPH,NOCGRAPH,NOTGRAPH,NOTRANS),
        TERMGRP=(NOLIST,NOTRANS,NOCGRAPH,NOGRAPH,NOTGRAPH),
        TERM=(NOLIST,NOTRANS,CGRAPH,GRAPH,TGRAPH)
BTRANS  TYPE=EXEC,TEXT=(EXEC),SCAN=YES,TIME=START
ETRANS  TEXT=(READY),SCAN=YES
RUN
END
```

You can enter these parameters from the console or provide them in a data set referenced on the SYSIN statement of the response time utility job stream. For more information about the use of the response time utility, refer to *WSim Utilities*

Guide.

Figure 38. Response Time Utility report of AVMON run

```
--------------------------------------------------------------------------------------------------------------------------
TERMINAL REPORT    NETWORK  ALL NETWORKS      PROCESS  ACTUAL      TIME LIMITS ALL               TRANSACTION *ALL*
                   VTAMAPPL APPLT1            EXIT                 START TIME  062351
                   TERMINAL TSO1-1            TERMTYPE LU2         END TIME    072421
--------------------------------------------------------------------------------------------------------------------------
MEAN   RESPONSE             4.98    MESSAGES SENT           29    NUMBER OF RESPONSES         23  SNA RESP SENT        63
MEDIAN RESPONSE             4.51       AVERAGE LENGTH       33       PER MINUTE                0  SNA RESP RECV         1
MODE   RESPONSE             4.51       PER MINUTE            0    RESPONSES DISCARDED          0
LOW    RESPONSE             2.60    MESSAGES RECEIVED       63    VARIANCE                7.1095
HIGH   RESPONSE            13.33       AVERAGE LENGTH       80    95 PERCENT CI               --
AVERAGE QUEUE TIME          0.44       PER MINUTE            1
PERCENTILE    RESPONSE TIME        AVERAGE
    90                 6.48           4.26
    95                11.72           4.60
    99                13.33           4.98
                                                  RESPONSE TIME FREQUENCY DISTRIBUTION
NETWORK              100 |---------------------------------------------------------------------------------------------------------|
VTAMAPPL APPLT1          |                                                                                                         |
TERMINAL TSO1-1          |                                                                                                         |
TRANSACT *ALL*           |                                                                                                         |
                         |                                                                                                         |
                      90 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      80 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      70 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      60 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
          PERCENTAGE  50 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
              OF         |                                                                                                         |
                         |                                                                                                         |
          RESPONSES      |                                                                                                         |
                      40 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      30 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      20 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |               *                                                                                         |
                         |               *                                                                                         |
                      10 |---------------*-----------------------------------------------------------------------------------------|
                         |           *   *           * *     *                                                                     |
                         |           *   *           * *     *                                                                     |
                         |         * * * *   *   *   * *   * * * *   * *   *                                                   *   |
                         |         * * * *   *   *   * *   * * * *   * *   *                                                   *   |
                        ____________________________________________________________________________________________________________
                             | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + +
                            2.0       3.0       4.0       5.0       6.0       7.0       8.0       9.0      10.0      11.0
                                                            RESPONSE TIME (SECONDS)          INCREMENT =  0.2 SECONDS
                                                                                             PERCENTAGE ABOVE LAST INCREMENT = 4.3

                                                  CUMULATIVE RESPONSE TIME DISTRIBUTION
NETWORK              100 |---------------------------------------------------------------------------------------------------------|
VTAMAPPL APPLT1          |                                                                                                         |
TERMINAL TSO1-1          |                                                                                                    *    |
TRANSACT *ALL*           |                                                                                                         |
                         |                                                  * * * * * * * * * * * * * * * * * * * * * * * * *      |
                      90 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                              * *                                                        |
                         |                                                                                                         |
                         |                                            *                                                            |
                      80 |---------------------------------------------------------------------------------------------------------|
                         |                                        * *                                                              |
                         |                                                                                                         |
                         |                                      *                                                                  |
                         |                                                                                                         |
                      70 |------------------------------------*--------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      60 |---------------------------------*-----------------------------------------------------------------------|
                         |                                                                                                         |
                         |                             * *                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
          PERCENTAGE  50 |---------------------------------------------------------------------------------------------------------|
                         |                           *                                                                             |
              OF         |                                                                                                         |
                         |                                                                                                         |
          RESPONSES      |                                                                                                         |
                      40 |-----------------------*-*-------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                   * *                                                                                   |
                         |                                                                                                         |
                      30 |---------------*-*---------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |                                                                                                         |
                      20 |---------------------------------------------------------------------------------------------------------|
                         |             *                                                                                           |
                         |                                                                                                         |
                         |           *                                                                                             |
                         |                                                                                                         |
                      10 |---------------------------------------------------------------------------------------------------------|
                         |                                                                                                         |
                         |                                                                                                         |
                         |         *                                                                                               |
                         |                                                                                                         |
                         ___________________________________________________________________________________________________________
                             | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + + | + + + +
                            2.0       3.0       4.0       5.0       6.0       7.0       8.0       9.0      10.0      11.0
                                                            RESPONSE TIME (SECONDS)          INCREMENT =  0.2 SECONDS

NETWORK                                              TIME GRAPH OF RESPONSES                                        < MINIMUM
VTAMAPPL APPLT1                                                                              INTERVAL =   180 SEC   * AVERAGE
TERMINAL TSO1-1                                                                              INCREMENT=   0.2 SEC   > MAXIMUM
TRANSACT *ALL*            RESPONSE TIME (SECONDS)
                        |                                                                                                     |
   TIME     NUMBER OF   |                                                  1         1         1         1         1         2|
            RESPONSES   |0----+----2----+----4----+----6----+----8----+----0----+----2----+----4----+----6----+----8----+----0|
  6.26.00       1       |                         T                                 *                                         |
  6.29.00       1       |                      *  T                                                                           |
  6.32.00       1       |                   *     T                                                                           |
  6.35.00       1       |                      *  T                                                                           |
  6.38.00       1       |                       * T                                                                           |
  6.41.00       1       |                         T  *                                                                        |
  6.44.00       1       |                   *     T                                                                           |
  6.47.00       1       |              *          T                                                                           |
  6.50.00       1       |                         T                                        *                                  |
  6.53.00       1       |                 *       T                                                                           |
  6.56.00       1       |                         T    *                                                                      |
  6.59.00       1       |              *          T                                                                           |
  7.02.00       1       |                       * T                                                                           |
  7.05.00       1       |               *         T                                                                           |
  7.08.00       1       |                         T                                                                           |
  7.11.00       1       |                *        T                                                                           |
  7.14.00       1       |                         T     *                                                                     |
  7.17.00       1       |               *         T                                                                           |
  7.20.00       1       |                         T *                                                                         |
  7.23.00       1       |                         T      *                                                                    |
  7.26.00       1       |                         T                                                                           |
```

AVMON decks

The following examples are the message generation decks for the sample AVMON configuration. Refer to *WSim User's Guide* for the location of these files on the WSim distribution tape.

AVMON VTAMAPPL configuration

```
AVMON    NTWRK HEAD='AVAILABILITY MONITOR',
**********************************************************************
* AVMON is a set of WSim scripts that may be used to monitor the     *
* availability and status of network subsystems and resources.       *
*                                                                    *
* This example of AVMON monitors two subsystems: TSO and NetView.    *
* More subsystems may be monitored by modifying the AMONTSO and      *
* ACHKTSO decks to meet the requirements of the additional subsystems.*
*                                                                    *
* AVMON is configured to run as a VTAMAPPL network.  It is not       *
* necessary to dedicate any communications controllers to this       *
* monitoring activity.  Running as a VTAMAPPL configuration does not *
* restrict the use of this tool only to the subsystems resident in   *
* the host processor that contains WSim.  Cross domain resources may *
* be monitored as well providing the network is capable of routing   *
* across domains.                                                    *
*                                                                    *
* Each message deck is preceded with a comment box that outlines     *
* the basic purpose of the deck and what WSim counters, switches,    *
* and areas are referenced in the deck.  Throughout each deck are    *
* comment blocks that describe the processing flow of the script.     *
*                                                                     *
* Certain aspects of this network will need to be tailored to meet    *
* your specific requirements.  These areas are underscored with       *
* "====" and are labeled with the word "CHANGE".                      *
***********************************************************************
               DELAY=F180,
*                    ==== <--------------------------------------CHANGE
               DISPLAY=(24,80),
               INIT=SEC,
               REPORT=NONE,
               OPTIONS=(NORRLOG),
               MSGTRACE=NO,
*                       == <-------------------------------------CHANGE
               STLTRACE=NO,
*                       == <-------------------------------------CHANGE
               NETUSER=400,
               USERAREA=200,
               BUFSIZE=3000,
               UTI=100

***********************************************************************
         INCLUDE AFORTIME
1        PATH  ACHKNETV
2        PATH  ACHKTSO
***********************************************************************
APPLNET  VTAMAPPL
NETCTRL  LU    LUTYPE=LU0,
               FRSTTXT=ACTRLNET
*---------------------------------------------------------------------*
APPLTC   VTAMAPPL
TSOCTRL  LU    LUTYPE=LU0,
               FRSTTXT=AMONTSO
*---------------------------------------------------------------------*
APPLT1   VTAMAPPL
TS01     LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),
                 MAXSESS=(0,1),DISPLAY=(24,80)
APPLT2   VTAMAPPL
TS02     LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),
                 MAXSESS=(0,1),DISPLAY=(24,80)
APPLT3   VTAMAPPL
TS03     LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),
                 MAXSESS=(0,1),DISPLAY=(24,80)
*---------------------------------------------------------------------*
APPLNC   VTAMAPPL
NETVCTRL LU    LUTYPE=LU0,
               FRSTTXT=AMONNETV

***********************************************************************
* The value of DELAY= will determine how quickly NV01, NV02, or NV03  *
* will cycle through the list of resources in ACHKNETV.  In this      *
* example there are only two resources (TSO and NetView), so a value  *
* of DELAY=F90 will mean each resource will be checked every three    *
* minutes.                                                            *
***********************************************************************
APPLN1   VTAMAPPL
NV01     LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),SAVEAREA=(1,6000),
                 MAXSESS=(0,1),DISPLAY=(24,80),DELAY=F90
*                                                    === <-------CHANGE
APPLN2   VTAMAPPL
NV02     LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),SAVEAREA=(1,6000),
                 MAXSESS=(0,1),DISPLAY=(24,80),DELAY=F90
*                                                    === <-------CHANGE
APPLN3   VTAMAPPL
NV03     LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),SAVEAREA=(1,6000),
                MAXSESS=(0,1),DISPLAY=(24,80),DELAY=F90
*                                                  === <-------CHANGE
**********************************************************************
```

ACTRLNET message generation deck

```
ACTRLNET MSGTXT
**********************************************************************
* This message deck is used by the LU named NETCTRL and acts as a     *
* network controller.                                                 *
*                                                                     *
* Comment sections in this deck are boxed with asterisks (*).         *
*                                                                     *
* The following switches, counters, and areas are referenced in this  *
* message deck:                                                       *
*                                                                     *
*       Switches:  NSW32  (request by monitoring terminal for         *
*                          special message print)                     *
*                                                                     *
*       Counters:  NC1    (counts minutes of elapsed time)            *
*                  NC2    (counts minutes of avail. for NetView)      *
*                  NC3    (counts minutes of availability for TSO)    *
*                  NC4    (additional subsystem avail. counter)       *
*                  NC5    (additional subsystem avail. counter)       *
*                  NC6    (additional subsystem avail. counter)       *
*                  NC7    (additional subsystem avail. counter)       *
*                  NC8    (additional subsystem avail. counter)       *
*                  NC9    (additional subsystem avail. counter)       *
*                  NC10   (additional subsystem avail. counter)       *
*                  DSEQ   (triggers printing of message to log)       *
*                                                                     *
*       Areas:     N+0    (network special message area, length 60)   *
*                  N+100  (NetView special message area, length 60)   *
*                  N+200  (TSO special message area, length 60)       *
*                  U+10   (reformatted time of day, length 8)         *
*                                                                     *
* Message generation statements which may need changing to match your *
* requirements are marked with the word "CHANGE".                     *
**********************************************************************
* When EVENT=SPECMSG is observed, special messages are output to the  *
* the operator screens and to the log.                                *
*                                                                     *
* When EVENT=PUTAV is observed, the availability summary is sent to   *
* the screen and log.                                                 *
*                                                                     *
* The two OPCMND statements ensure the network resources are          *
* synchronized with NETCTRL LU.                                       *
**********************************************************************
         DATASAVE  AREA=N+0,TEXT=( )
         DATASAVE  AREA=N+1,TEXT=($RECALL,N+0,299$)
         ON        EVENT=SPECMSG,THEN=E-LSPECMSG
         ON        EVENT=PUTAV,THEN=E-PUTAV
         OPCMND    (A AVMON,QUIESCE)
         OPCMND    (A NETCTRL,RELEASE)

**********************************************************************
* Deck AFORTIME reformats the standard time of day (HHMMSS) into a    *
* more readable format (HH:MM:SS).                                    *
*                                                                     *
* The TEXT=( ) on the IF statement specifies when system availability *
* monitoring is to start.  If TOD is greater than that specified,     *
* processing with branch to RELEASE and monitoring will begin, else   *
* check again in 10 seconds.                                          *
**********************************************************************
CHKSERVB LABEL
```

```
         CALL   NAME=AFORTIME
000      IF     WHEN=IMMED,LOC=U+10,TEXT=(07:30),COND=GE,THEN=B-RELEASE
         WAIT   TIME=10                    ===== <----------------CHANGE
         BRANCH LABEL=CHKSERVB

***********************************************************************
* Availability monitoring has begun.  A message indicating time and   *
* date is written to the operator screen and to the log.              *
***********************************************************************
RELEASE  LABEL
         CALL  NAME=AFORTIME
         WTO   ( )
         LOG   ( )
         WTO   (***  BEGIN MONITORING - $RECALL,U+10,8$, $MONTH$/$DAY$/*
 $YEAR$)
         LOG   (***  BEGIN MONITORING - $RECALL,U+10,8$, $MONTH$/$DAY$/*
 $YEAR$)
***********************************************************************
* The LU's that act as controllers to the system monitoring pools are *
* now activated.                                                      *
***********************************************************************
         OPCMND  (A TSOCTRL,RELEASE)
         OPCMND  (A NETVCTRL,RELEASE)
***********************************************************************
* The other control terminals are given 10 seconds to increment their *
* availability counters.                                              *
***********************************************************************
         WAIT  TIME=10

***********************************************************************
* The LU waits for 3 minutes, gets the time of day, increments the    *
* elapsed time counter (NC1), increments the number of 3 minute time  *
* increments that have passed (DSEQ), then evaluates the following    *
* logic tests:                                                        *
*                                                                     *
*      IF 001: stops WSim if the end of availability monitoring has   *
*              been reached or exceeded.                              *
*      IF 002: prints the availability status if another control      *
*              terminal has requested it (by setting NSW32).          *
*      IF 003: prints the availability status when the specified      *
*              time interval has elapsed.  DSEQ counts in 3 minute    *
*              intervals.  One hour would be DSEQ=20 (20x3).          *
*                                                                     *
* If no THEN actions are taken, processing loops back to the 3 minute *
* wait at label "WAITLAP".                                            *
***********************************************************************
WAITLAP  WAIT   TIME=180
         CALL   NAME=AFORTIME
         SET    NC1=+3
         SET    DSEQ=+1
001      IF    WHEN=IMMED,LOC=U+10,TEXT=(18:00),COND=GE,THEN=C-STOPWSIM
*                                          ===== <------------------CHANGE
002      IF     WHEN=IMMED,LOC=NSW32,THEN=C-PUTAV
003      IF     WHEN=IMMED,LOC=DSEQ,TEXT=20,THEN=C-PUTAV
*                                          == <---------------------CHANGE
         BRANCH LABEL=WAITLAP

***********************************************************************
* If an availability status print was requested as determined by      *
* IF 002 or IF 003, then the status is written to the operator screen *
* and the log.  Once completed, NSW32 (request to print status by     *
* another controlling terminal) is set OFF and DSEQ is reset to zero  *
* (DSEQ counts the number of 3 minute intervals between normal status *
* prints).                                                            *
*                                                                     *
* Processing then returns to the point from which "PUTAV" was called  *
***********************************************************************
PUTAV    LABEL
 WTO (+AVAILABILITY WITHIN $CNTR,NC1,3$ MINUTES - $RECALL,U+10,8$)
 WTO (+NETVIEW $CNTR,NC2,3$ TSO $CNTR,NC3,3$)
 LOG (+AVAILABILITY WITHIN $CNTR,NC1,3$ MINUTES - $RECALL,U+10,8$)
 LOG (+NETVIEW $CNTR,NC2,3$ TSO $CNTR,NC3,3$)
************************************************************************
* If logging to tape is to be performed, then comment out or delete   *
* each OPCMND (E) statement.  They are intended for use on logging to *
* disk runs only.  These statements are included to preserve the      *
* log dataset at regular intervals.                                   *
*                                                                     *
* When logging to disk, ensure the LOGDD statement on the execution   *
* JCL specifies DISP=MOD, otherwise the file will be closed and re-   *
* opened at the beginning (thus destroying all the previous logging). *
************************************************************************
         OPCMND (E) <--------------------------------------------CHANGE
         OPCMND (E) <--------------------------------------------CHANGE
         SETSW  NSW32=OFF
         SET    DSEQ=0
         RETURN
************************************************************************
* The following sections is EXECUTED (performed immediately) whenever *
* the event "SPECMSG" is signalled by one of the other controlling    *
* terminals.  The special message is written to the operator screen   *
* and to the log, then copied to the NetView special msg. area N+100. *
* EVENT NETVMSG is signalled, which writes the message to a NetView   *
* operator.  The network special message area is then cleared and     *
* the on EVENT=SPECMSG is reset.  Processing then returns to the      *
* point where the signalled event was detected.                       *
************************************************************************
LSPECMSG LABEL
         WTO      ($RECALL,N+0,60$)
         LOG      ($RECALL,N+0,60$)
         DATASAVE AREA=N+100,TEXT=($RECALL,N+0,100$)
         EVENT    SIGNAL=NETVMSG
         DATASAVE AREA=N+0,TEXT=( )
         DATASAVE AREA=N+1,TEXT=($RECALL,N+0,99$)
         ON       EVENT=SPECMSG,THEN=E-LSPECMSG
         RETURN
************************************************************************
* WSim is stopped when the availability monitoring is no longer       *
* desired as determined by IF 001.  The time of day is determined     *
* and a message is written to the operator screen and the log.  Then  *
* WSim is stopped with the OPCMND (Z END).                            *
************************************************************************
STOPWSIM LABEL
         CALL    LABEL=PUTAV
         CALL    NAME=AFORTIME
         WTO   (***  END OF MONITORING - $RECALL,U+10,8$  $MONTH$/$DAY$*
 /$YEAR$)
         LOG   (***  END OF MONITORING - $RECALL,U+10,8$  $MONTH$/$DAY$*
 /$YEAR$)
         OPCMND  (Z END)
         RETURN
         ENDTXT
************************************************************************
```

AFORTIME message generation deck

```
AFORTIME MSGTXT
************************************************************************
* This deck reformats the time of day data field option into a more   *
* readable format (from HHMMSS to HH:MM:SS).                          *
************************************************************************
         DATASAVE  AREA=U+12,TEXT=($TOD,6$)
         DATASAVE  AREA=U+10,
             TEXT=($RECALL,U+12,2$:$RECALL,U+14,2$:$RECALL,U+16,2$)
         RETURN
         ENDTXT
***********************************************************************
```

AMONNETV message generation deck

```
AMONNETV MSGTXT
***********************************************************************
* This message deck is used by the LU named NETVCTRL and acts as a    *
* controller of the 3 NetView terminals that check the availability   *
* of the named system resources.                                      *
*                                                                     *
* Comment sections in this deck are boxed with asterisks (*).         *
*                                                                     *
* The following switches, counters, and areas are referenced in this  *
* message deck:                                                       *
*                                                                     *
*       Switches:  NSW1   (set by one of the NetView terminals,       *
*                          it indicates the state of the NetView      *
*                          system)                                    *
*                                                                     *
*       Counters:  NC2    (counts minutes of avail. for NetView)      *
*                  DSEQ   (tracks which NetView terminal is active)   *
*                                                                     *
*       Areas:     N+0    (network special message area, length 60)   *
*                  U+10   (reformatted time of day, length 8)         *
*                                                                     *
* Message generation statements which may need changing to match your *
* requirements are marked with the word "CHANGE".                     *
***********************************************************************
* An initial wait of one second is provided to allow NETCTRL to       *
* quiesce all LU's and establish synchronization.  DSEQ is            *
* incremented to track which NetView terminal from the pool is        *
* currently active.  The first NetView terminal is released from its  *
* quiesced state.                                                     *
***********************************************************************
        WAIT  TIME=1
        SET   DSEQ=1
        OPCMND  (A NV01,RELEASE)

***********************************************************************
* NETVCTRL goes into a 3 minute wait while the NetView terminal       *
* under its control continues to issue D NET commands (and thus       *
* monitor the status of NetView).  After 3 minutes have expired the   *
* time of day is reformatted (AFORTIME) and a logic test of NSW1      *
* is evaluated.  If NSW1 is on, then processing will branch to        *
* label "OK", indicating the NetView terminal is satisfied with the   *
* response from the D NET command.                                    *
***********************************************************************
CHECK   LABEL
        WAIT  TIME=180
CHECK1  LABEL
        CALL  NAME=AFORTIME
0       IF    LOC=NSW1,WHEN=IMMED,THEN=B-OK,ELSE=CONT

***********************************************************************
* Processing of this section indicates NSW1 was off for the previous  *
* IF.  Two more tests of NSW1 will occur, each one minute apart.  If  *
* after 2 minutes NSW1 is not turned on, then the NetView             *
* terminal is assumed inactive and steps are taken to start the next  *
* LU in the NetView terminal pool.                                    *
***********************************************************************
        DATASAVE AREA=N+0,
        TEXT=(-NETV-$CNTR,DSEQ,2$ IS NOT RESPONDING - $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
        WAIT  TIME=60
```

```
1       IF    LOC=NSW1,WHEN=IMMED,THEN=B-NOWOK
        WAIT  TIME=60
2       IF    LOC=NSW1,WHEN=IMMED,ELSE=B-REINIT
***********************************************************************
* Previous tests of NSW1 were met, indicating NetView is operational. *
* If NSW1 was on for IF 1, then "NOWOK" is processed: a special       *
* message is written to indicate the NetView terminal is again        *
* responding.  If NSW1 was on all along, then "OK" is processed: NSW1 *
* is turned off, and the NetView availability counter is incremented  *
* by 3 minutes.                                                       *
***********************************************************************
NOWOK   LABEL
        CALL  NAME=AFORTIME
        DATASAVE AREA=N+0,
       TEXT=(+NETV-$CNTR,DSEQ,2$ IS RESPONDING AGAIN - $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
OK      LABEL
        SETSW NSW1=OFF
        SET NC2=+3
        BRANCH LABEL=CHECK
***********************************************************************
* NSW1 was never determined to be on, so the NetView terminal is      *
* assumed to be inactive at this point.  It will be quiesced and the  *
* next LU in the terminal pool will be released from its quiesced     *
* state.  This is controlled according to the value of DSEQ, which    *
* was tracking which NetView terminal was currently in operation.     *
* If no more LU's are available from the pool, then processing        *
* branches to "NOMORE".                                               *
***********************************************************************
REINIT  LABEL
3       IF    LOC=DSEQ,TEXT=1,WHEN=IMMED,ELSE=B-N1
        OPCMND  (A NV01,QUIESCE)
        OPCMND  (A NV02,RELEASE)
        BRANCH LABEL=ENDUP
N1      LABEL
4       IF    LOC=DSEQ,TEXT=2,WHEN=IMMED,ELSE=B-NOMORE
        OPCMND  (A NV02,QUIESCE)
        OPCMND  (A NV03,RELEASE)
        BRANCH LABEL=ENDUP
***********************************************************************
* The next NetView terminal has been released and the value of DSEQ   *
* is incremented by one.  NETVCTRL waits for 180 seconds and          *
* processing branches to "CHECK1".                                    *
***********************************************************************
ENDUP   LABEL
        SET   DSEQ=+1
        WAIT  TIME=180
        BRANCH LABEL=CHECK1
***********************************************************************
* The NetView terminal pool is exhausted.  Event PUTAV is signalled   *
* write the availability summary. A special message                   *
* is written to the network special message area and then EVENT       *
* SPECMSG is signalled to write it out to the operator screen and to  *
* the log.  Then this WSim network (AVMON) is CANCELLED, INITIALIZED, *
* and STARTED via the OPCMND's.                                       *
***********************************************************************
NOMORE   LABEL
         CALL  NAME=AFORTIME
         EVENT SIGNAL=PUTAV
         DATASAVE AREA=N+0,
         TEXT=(*** NETV POOL EXHAUSTED, REINIT AVMON - $RECALL,U+10,8$)
         EVENT SIGNAL=SPECMSG
5        IF LOC=U+10,WHEN=IMMED,COND=GE,TEXT=(17:30),THEN=B-NOREST
*                                            ===== <------------CHANGE
         DATASAVE AREA=N+0,
         TEXT=(*** RESTART OF AVMON BY $LUID$ - $RECALL,U+10,8$)
         EVENT SIGNAL=SPECMSG
         OPCMND (C AVMON)
         OPCMND (I AVMON)
         OPCMND (S AVMON)
         BRANCH LABEL=WAIT
NOREST   LABEL
         DATASAVE AREA=N+0,
         TEXT=(*** NO RESTART OF AVMON AT THIS TIME - $RECALL,U+10,8$)
         EVENT SIGNAL=SPECMSG
WAIT     WAIT
         BRANCH LABEL=WAIT
         ENDTXT
***********************************************************************
```

ALOGNETV message generation deck

```
ALOGNETV MSGTXT
***********************************************************************
* The following deck is used to logon to NetView.  The userid's to    *
* be used are assumed to be WSIM01, WSIM02, and WSIM03.               *
*                                                                     *
* Comment sections in this deck are boxed with asterisks (*).         *
*                                                                     *
* The following switches, counters, and areas are referenced in this  *
* message deck:                                                       *
*                                                                     *
*       Switches:         (there are no switches referenced)          *
*                                                                     *
*       Counters:         (there are no counters referenced)          *
*                                                                     *
*       Areas:     U+0    (userid work area, length 6)                *
*                                                                     *
* Message generation statements which may need changing to match your *
* requirements are marked with the word "CHANGE".                     *
***********************************************************************
* The following section composes the userid (concatenating "WSIM"     *
* and the last 2 digits of the LU name) and issues the CMND           *
* that will establish the session.                                    *
***********************************************************************
         WAIT  TIME=10
         DATASAVE AREA=U+2,TEXT=($ID,4$)
         DATASAVE AREA=U+0,TEXT=(WSIM)
         CMND  COMMAND=INIT,MODE=D6327802,RESOURCE=NETV
*                                ========          ==== <--------CHANGE

***********************************************************************
* The following IF holds up processing until the text "PASSWORD" is   *
* encountered.                                                        *
***********************************************************************
0        IF    LOC=B+0,SCAN=YES,TEXT=(PASSWORD),THEN=CONT,ELSE=WAIT,
               STATUS=HOLD,DELAY=CANCEL

***********************************************************************
* The next section delivers the necessary information to logon to     *
* NetView.  The first TEXT statement delivers the userid as composed  *
* earlier in this logon deck.                                         *
*                                                                     *
* The second TEXT statement delivers the password for this particular *
* userid.  The password you provide for each userid (WSIM01, WSIM02,  *
* and WSIM03) should end with the same two digit suffix as the        *
* userid.  The value of "yourpw" you provide below, coupled with the  *
* last two digits of the userid name ($RECALL,U+4,2$) will comprise   *
* the password.  Ensure that all passwords begin with the value       *
* supplied at "yourpw".                                               *
***********************************************************************
         TEXT  ($RECALL,U+0,6$)
         TAB
         TEXT  (yourpw$RECALL,U+4,2$)
```

```
*              ====== <------------------------------------------CHANGE
         TAB
         TAB
         TAB
         TEXT  (NO)
         ENTER
***********************************************************************
* This last section waits until NetView is fully functional before    *
* continuing.  By using IF 0, the previous IF named 0 is deactivated  *
* and this IF takes its place, looking for the underscore line        *
* (-----) as an indication of NetView operation.  The time of day     *
* is then reformatted (AFORTIME) and a message is written to the      *
* operator screen and to the log telling of the new LU in session     *
* with NetView.                                                       *
***********************************************************************
         DELAY TIME=F1
0        IF    LOC=B+0,SCAN=YES,TEXT=(-----),THEN=B-RETURN,DELAY=CANCEL
WAIT     WAIT
         BRANCH LABEL=WAIT
RETURN   CALL  NAME=AFORTIME
         WTO  (+NETVIEW IS NOW RESPONDING - $RECALL,U+10,8$)
         LOG  (+NETVIEW IS NOW RESPONDING - $RECALL,U+10,8$)
         ENDTXT
***********************************************************************
```

ACHKNETV message generation deck

```
ACHKNETV MSGTXT
**********************************************************************
* This message deck is used by the NetView terminal to monitor       *
* the status of network resources by issuing a DISPLAY NET command   *
* from the NetView terminal.                                         *
*                                                                    *
* Comment sections in this deck are boxed with asterisks (*).        *
*                                                                    *
* The following switches, counters, and areas are referenced in this *
* message deck:                                                      *
*                                                                    *
*        Switches:  NSW1   (Used by the LU executing this deck to    *
*                           indicate to NETVCTRL that it's still     *
*                           active)                                  *
*                                                                    *
*        Counters:  DC1    (Used to track the number of special      *
*                           messages stacked in the save area but    *
*                           not yet sent to NetView)                 *
*                                                                    *
*        Areas:     N+0    (network special message area, length 60) *
*                   N+100  (The NetView special msg area, length 60) *
*                   U+10   (Reformatted time of day, length 8)       *
*                   U+20   (resource name work area, length 16)      *
*                   U+40   (resource name setup area, length 16)     *
*                   U+60   (resource name message area, length 16)   *
*                   U+80   (status message area, length 14)          *
*                   U+100  (VTAM message capture area, length 65)    *
*                                                                    *
*                                                                    *
*                                                                    *
* Message generation statements which may need changing to match your*
* requirements are marked with the word "CHANGE".                    *
**********************************************************************
* The VTAM message capture area (U+100) is cleared of any data and   *
* AUTOWRAP is set on.                                                *
**********************************************************************
         DATASAVE  AREA=U+100,TEXT=()
         TEXT  (AUTOWRAP YES)
         ENTER
```

```
**********************************************************************
* ON EVENT=NETVMSG is set to execute SNETVMSG whenever NETVMSG is    *
* signalled.  This will concatenate the special message to the       *
* special message queue in savearea 1.                               *
*                                                                    *
* ON EVENT=NETVMSGL is set to call LNETVMSG whenever NETVMSGL is     *
* signalled.  NETVMSGL is signalled in SNETVMSG.  This will send     *
* the special messages from savearea 1 to the NetView console.       *
*                                                                    *
* IF 0 executes DATASAV1 whenever the VTAM message IST486I is        *
* observed.  This will be the status response to a valid resource    *
* requested with the D NET command later in this deck.               *
*                                                                    *
* IF 1 and 2 executes DATASAV1 whenever IST088I or IST453I is        *
* observed.  IST088I is the VTAM response when D NET on an unknown   *
* or inactive resource is performed with VTAM R2 or before.  IST453I *
* results with VTAM R3 or later.                                     *
*                                                                    *
* IF 3 executes DATASAV2 whenever the string STATUS= is observed.    *
* "STATUS=" is part of the IST486 message for pre-VTAM 3.2.          *
*                                                                    *
* IF 4 executes DATASAV3 whenever the string STATE is observed.      *
* "STATE" is part of the IST486 message for VTAM 3.2.                *
**********************************************************************
         ON    EVENT=NETVMSG,THEN=E-SNETVMSG
         ON    EVENT=NETVMSGL,THEN=C-LNETVMSG
0        IF    TEXT=(IST486I),LOC=D+0,SCAN=YES,THEN=E-DATASAV1,
               STATUS=HOLD
1        IF    TEXT=(IST088I),LOC=D+0,SCAN=YES,THEN=E-DATASAV1,
               STATUS=HOLD
2        IF    TEXT=(IST453I),LOC=D+0,SCAN=YES,THEN=E-DATASAV1,
               STATUS=HOLD
3        IF    TEXT=(STATUS=),LOC=D+0,SCAN=YES,THEN=E-DATASAV2,
               STATUS=HOLD
4        IF    TEXT=(STATE),LOC=D+0,SCAN=YES,THEN=E-DATASAV3,
               STATUS=HOLD
         BRANCH  LABEL=NODELAY
**********************************************************************
* The following section is used to determine what resource will be   *
* named in the D NET command.  The TEXT operand of the DATASAVE      *
* statement names the resource.  More resources may be monitored by  *
* replicating the DATASAVE and CALL statements and changing the      *
* TEXT operand.  Data following the resource may be included to      *
* provide a brief comment to the operator.                           *
**********************************************************************
REPEAT   LABEL
         SETSW NSW1=ON
NODELAY  LABEL
         DATASAVE  AREA=U+20,TEXT=(ITPECHO         )
         CALL  LABEL=NETVDIS
         DATASAVE  AREA=U+20,TEXT=(APPL5           )
         CALL  LABEL=NETVDIS
         BRANCH  LABEL=REPEAT
**********************************************************************
* NETVDIS first formats the time of day in a more readable format    *
* (AFORTIME), moves resource data in the USERAREA and then branches  *
* to either NOMSG (no VTAM message is stored in U+100), IST486I      *
* (a status response to the D NET is stored in U+100), or IST088I/   *
* IST453I (the resource requested was inactive or unknown to VTAM).  *
**********************************************************************
NETVDIS  LABEL
         CALL  NAME=AFORTIME
         DATASAVE  AREA=U+60,TEXT=($RECALL,U+40,20$)
         DATASAVE  AREA=U+40,TEXT=($RECALL,U+20,20$)
4        IF    TEXT=('00'),LOC=U+100,WHEN=IMMED,THEN=B-NOSMSG
```

```
5        IF    TEXT=(IST486I),LOC=U+100,WHEN=IMMED,THEN=B-IST486I
6        IF    TEXT=(IST088I),LOC=U+100,WHEN=IMMED,THEN=B-IST088I
7        IF    TEXT=(IST453I),LOC=U+100,WHEN=IMMED,THEN=B-IST453I
         BRANCH  LABEL=NOSMSG
***********************************************************************
* The following section is called when immediate IF 6 above detects  *
* VTAM message IST486I stored in U+100.  NSW1 is set on to indicate  *
* a valid response.  Immediate IF 8 checks if the status was "ACTIV",*
* and if so then processing branches to NOMSG.  If not, then a       *
* message is copied to the network special message area (N+0)        *
* and processing branches to SIGNALSP.                               *
***********************************************************************
IST486I  LABEL
         SETSW NSW1=ON
8        IF    TEXT=(ACTIV),LOC=U+100,WHEN=IMMED,SCAN=50,
               THEN=B-NOSMSG
         DATASAVE  AREA=N+0,
         TEXT=(+$RECALL,U+60,16$ $RECALL,U+80,14$ $RECALL,U+10,8$    )
         BRANCH  LABEL=SIGNALSP
***********************************************************************
* The following section is processed when immediate IFs 7 or 8       *
* detected VTAM message IST088I or IST453I was returned from the     *
* D NET command.  A message is copied to the network special message *
* area (N+0) and processing branches to SIGNALSP.                    *
***********************************************************************
IST088I  LABEL
IST453I  LABEL
         SETSW NSW1=ON
         DATASAVE  AREA=N+0,
         TEXT=(-$RECALL,U+60,16$ NOT ACTIVATED - $RECALL,U+10,8$  )
         BRANCH  LABEL=SIGNALSP
***********************************************************************
* The following section clears the VTAM message save area (U+100)    *
* and signals the EVENT SPECMSG (which is processed by ACTRLNET).    *
***********************************************************************
SIGNALSP LABEL
         DATASAVE  AREA=U+100,TEXT=('00')
         DATASAVE  AREA=U+101,TEXT=($RECALL,U+100,99$)
         EVENT  SIGNAL=SPECMSG
***********************************************************************
* The following section issues the D NET command to display the      *
* status of the resource whose name is currently stored in U+40.     *
* First the VTAM message area is cleared, a TAB is processed to      *
* return the cursor to the beginning of the NetView input area,      *
* and EREOF is processed to clear the input field, and the D NET     *
* command is entered with the resource name being that located in    *
* U+40.                                                              *
***********************************************************************
NOSMSG   LABEL
         DATASAVE  AREA=U+100,TEXT=('00')
         DATASAVE  AREA=U+101,TEXT=($RECALL,U+100,99$)
         TAB
         EREOF
         TEXT  (D NET,NONE,ID=$RECALL,U+40,16$  )
         ENTER
         STOP
         RETURN
***********************************************************************
* DATASAV1 is executed on the THEN leg of IFs 0, 1, and 2.  This     *
* will occur whenever VTAM message IST486I, IST088I, or IST453I      *
* is detected.  DATASAV1 saves the entire VTAM message into U=100.   *
***********************************************************************
DATASAV1 LABEL
         DATASAVE AREA=U+100,LENG=065,LOC=*
         RETURN
```

```
***********************************************************************
* DATASAV2 is executed on the THEN leg of IF 3.  This will occur     *
* when the pre-VTAM 3.2 message IST486I with the STATUS= of the      *
* named resource is detected in the inbound datastream.  WSim saves  *
* away the status of the named resource at location U+80.            *
***********************************************************************
DATASAV2 LABEL
         DATASAVE AREA=U+80,LENG=014,LOC=*
         RETURN
***********************************************************************
* DATASAV3 is executed on the THEN leg of IF 4.  This will occur     *
* when the VTAM 3.2 message IST486I with the current state of the    *
* named resource is detected in the inbound datastream.  WSim saves  *
* away the current state of the resource at location U+80.           *
***********************************************************************
DATASAV3 LABEL
         DATASAVE AREA=U+80,LENG=013,LOC=*
         RETURN
***********************************************************************
* LNETVMSG issues the SPECMSG message to the NetView operator when-  *
* ever EVENT SNETVMSG is signalled.  Change the operator ID to match *
* the local system configuration.                                     *
*                                                                     *
* The special message queue in savearea 1 is reduced, one message at *
* a time, until the counter DC1=0.  This means no messages are left. *
***********************************************************************
LNETVMSG LABEL
         TAB
         EREOF
         TEXT  (MSG OPER,$RECALL,1+0,60$)
*                   ==== <---------------------------------------CHANGE
         ENTER
         DELAY TIME=F5
         DATASAVE  AREA=1,TEXT=($RECALL,1+60,6000$)
         SET DC1=-1
         IF LOC=DC1,TEXT=0,ELSE=B-LNETVMSG,WHEN=IMMED
         ON EVENT=NETVMSGL,THEN=C-LNETVMSG
         RETURN
***********************************************************************
* SNETVMSG is executed when NETVMSG is signalled.  This concatenates *
* new messages to the end of existing messages in the message queue  *
* (savearea 1).  The message area N+100 is cleared, DC1 is           *
* incremented to reflect the new message in the queue, the event     *
* NETVMSGL is signalled (to begin the process of actually sending    *
* the messages to NetView), the ON condition is reset, and processing*
* returns.                                                            *
***********************************************************************
SNETVMSG DATASAVE AREA=1,TEXT=($RECALL,1$$RECALL,N+100,60$)
         DATASAVE AREA=N+100,TEXT=('00')
         DATASAVE AREA=N+101,TEXT=($RECALL,N+100,99$)
         SET DC1=+1
         EVENT SIGNAL=NETVMSGL
         ON EVENT=NETVMSG,THEN=E-SNETVMSG
         RETURN
         ENDTXT
************************************************************************
```

AMONTSO message generation deck

```
AMONTSO MSGTXT
************************************************************************
* THIS MESSAGE DECK IS USED BY THE LU NAMED TSOCTRL AND ACTS AS A     *
* CONTROLLER OF THE 3 TSO TERMINALS THAT CHECK THE AVAILABILITY       *
* OF THE TSO SYSTEM.                                                   *
*                                                                      *
* COMMENT SECTIONS IN THIS DECK ARE BOXED WITH ASTERISKS (*).         *
*                                                                      *
* THE FOLLOWING SWITCHES, COUNTERS, AND AREAS ARE REFERENCED IN THIS  *
* MESSAGE DECK:                                                        *
*                                                                      *
*       SWITCHES:  NSW2   (SET BY ONE OF THE TSO TERMINALS, IT         *
*                          INDICATES THE STATE OF THE TSO SYSTEM)      *
*                                                                      *
*       COUNTERS:  NC3    (COUNTS MINUTES OF AVAILABILITY FOR TSO)     *
*                  DSEQ   (TRACKS WHICH TSO TERMINAL IS ACTIVE)        *
*                                                                      *
*       AREAS:     N+0    (NETWORK SPECIAL MESSAGE AREA, LENGTH 60)   *
*                  U+10   (REFORMATTED TIME OF DAY, LENGTH 8)          *
*                                                                      *
* MESSAGE GENERATION STATEMENTS WHICH MAY NEED CHANGING TO MATCH YOUR *
* REQUIREMENTS ARE MARKED WITH THE WORD "CHANGE".                      *
************************************************************************
* AN INITIAL WAIT OF ONE SECOND IS PROVIDED TO ALLOW NETCTRL TO        *
* QUIESCE ALL LU'S AND ESTABLISH SYNCHRONIZATION.  DSEQ IS             *
* INCREMENTED TO TRACK WHICH TSO TERMINAL FROM THE POOL IS             *
* CURRENTLY ACTIVE.  THE FIRST TSO TERMINAL IS RELEASED FROM ITS       *
* QUIESCED STATE.                                                      *
************************************************************************
        WAIT  TIME=1
        SET   DSEQ=1
        OPCMND  (A TS01,RELEASE)

************************************************************************
* TSOCTRL GOES INTO A 3 MINUTE WAIT WHILE THE TSO TERMINAL UNDER       *
* ITS CONTROL CONTINUES TO MONITOR THE STATUS OF TSO.  AFTER 3         *
* MINUTES HAVE EXPIRED THE TIME OF DAY IS REFORMATTED (AFORTIME) AND  *
* A LOGIC TEST ON NSW2 IS EVALUATED.  IF NSW2 IS ON, THEN PROCESSING  *
* WILL BRANCH TO LABEL "OK", INDICATING THE TSO TERMINAL IS SATIS-     *
* FIED WITH THE RESPONSE IT RECEIVED FROM TSO                          *
************************************************************************
CHECK   LABEL
        WAIT  TIME=180
CHECK1  LABEL
        CALL  NAME=AFORTIME
        IF    LOC=NSW2,WHEN=IMMED,THEN=B-OK,ELSE=CONT

************************************************************************
* PROCESSING OF THIS SECTION INDICATES NSW1 WAS OFF FOR THE PREVIOUS  *
* IF.  TWO MORE TESTS OF NSW2 WILL OCCUR, EACH ONE MINUTE APART.  IF  *
* AFTER 2 MINUTES NSW2 IS NOT TURNED ON, THEN THE TSO TERMINAL IS      *
* ASSUMED INACTIVE AND STEPS ARE TAKEN TO ATTEMPT TO RECOVER THE       *
* TERMINAL.                                                            *
************************************************************************
        DATASAVE AREA=N+0,
        TEXT=(-TSO-$CNTR,DSEQ,2$ IS NOT RESPONDING - $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
        WAIT  TIME=60
        IF    LOC=NSW2,WHEN=IMMED,THEN=B-NOWOK,ELSE=CONT
        WAIT  TIME=60
        IF    LOC=NSW2,WHEN=IMMED,THEN=B-NOWOK
        BRANCH LABEL=ATTENT

************************************************************************
* PREVIOUS TESTS OF NSW2 WERE MET, INDICATING TSO IS OPERATIONAL.      *
* IF NSW2 WAS ON AFTER THE 2ND MINUTE THEN "NOWOK" IS PROCESSED: A     *
* SPECIAL MESSAGE IS WRITTEN TO INDICATE THE TSO TERMINAL IS AGAIN     *
* RESPONDING.  IF NSW2 WAS ON ALL ALONG, THEN "OK" IS PROCESSED: NSW2 *
* IS TURNED OFF, AND THE TSO AVAILABILITY COUNTER IS INCREMENTED BY    *
* 3 MINUTES.                                                           *
************************************************************************
NOWOK   LABEL
        CALL  NAME=AFORTIME
        DATASAVE AREA=N+0,
        TEXT=(+TSO-$CNTR,DSEQ,2$ IS RESPONDING AGAIN - $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
```

```
OK      LABEL
        SETSW NSW2=OFF
        SET   NC3=+3
        BRANCH LABEL=CHECK
***********************************************************************
* NSW2 WAS NEVER DETERMINED TO BE ON, SO THE TSO TERMINAL IS ASSUMED  *
* TO BE INACTIVE AT THIS POINT.  IT WILL BE QUIESCED AND THE NEXT LU  *
* IN THE TERMINAL POOL IS RELEASED FROM ITS QUIESCED STATE.  THIS     *
* IS CONTROLLED ACCORDING TO THE VALUE OF DSEQ, WHICH WAS TRACKING    *
* WHICH TSO TERMINAL WAS CURRENTLY IN OPERATION.  IF NO MORE LU'S     *
* ARE AVAILABLE FROM THE POOL, THEN PROCESSING BRANCHES TO "NOMORE".  *
***********************************************************************
REINIT  LABEL
        DEACT ONEVENTS=(TSOREC)
        IF    LOC=DSEQ,TEXT=1,WHEN=IMMED,ELSE=B-N1
        OPCMND  (A TS01,QUIESCE)
        OPCMND  (A TS02,RELEASE)
        BRANCH LABEL=ENDUP
N1      LABEL
        IF    LOC=DSEQ,TEXT=2,WHEN=IMMED,ELSE=B-NOMORE
        OPCMND  (A TS02,QUIESCE)
        OPCMND  (A TS03,RELEASE)
        BRANCH LABEL=ENDUP
***********************************************************************
* THIS SECTION PROMPTS THE TSO TERMINAL TO ISSUE AN "ATTN" TO TRY TO  *
* RECOVER.  A 60 SECOND WAIT IS IMPOSED TO SEE IF THIS WORKED.  IF    *
* NOT, THE PROCESSING BRANCHES TO REINIT.                             *
***********************************************************************
ATTENT  LABEL
        EVENT SIGNAL=ATTENT
        ON EVENT=TSOREC,THEN=B-NOWOK
        WAIT TIME=60
        BRANCH LABEL=REINIT
***********************************************************************
* THE NEXT TSO TERMINAL HAS BEEN RELEASED AND THE VALUE OF DSEQ IS    *
* INCREMENTED BY ONE.  TSOCTRL WAITS FOR 180 SECONDS AND PROCESSING   *
* BRANCHES TO "CHECK1".                                               *
***********************************************************************
ENDUP   LABEL
        SET   DSEQ=+1
        WAIT  TIME=180
        BRANCH LABEL=CHECK1
***********************************************************************
* THE TSO TERMINAL POOL HAS BEEN EXHAUSTED.  EVENT PUTAV IS SIGNALLED *
* TO WRITE THE AVAILABILITY SUMMARY REPORT.  A SPECIAL MESSAGE IS     *
* WRITTEN TO THE NETWORK SPECIAL MESSAGE AREA AND THEN EVENT SPECMSG  *
* IS SIGNALLED TO WRITE IT OUT THE THE OPERATOR SCREEN AND TO THE     *
* LOG.  THEN THIS WSim NETWORK (AVMON) IS CANCELLED, INITIALIZED,     *
* AND STARTED VIA THE OPCMND'S.                                       *
***********************************************************************
NOMORE  LABEL
        CALL  NAME=AFORTIME
        EVENT  SIGNAL=PUTAV
        DATASAVE AREA=N+0,
        TEXT=(*** TSO POOL EXHAUSTED, REINIT AVMON - $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
        CALL  NAME=AFORTIME
0       IF    TEXT=(17:30),LOC=U+10,WHEN=IMMED,COND=GE,THEN=B-NOREST
*                  ===== <--------------------------------------CHANGE
        DATASAVE AREA=N+0,
        TEXT=(*** RESTART OF AVMON BY $LUID$ - $RECALL,U+10,8$)
        EVENT  SIGNAL=SPECMSG
        OPCMND (C AVMON)
        OPCMND (I AVMON)
        OPCMND (S AVMON)
        BRANCH  LABEL=WAIT
```

```
NOREST  LABEL
        DATASAVE AREA=N+0,
        TEXT=(*** NO RESTART OF AVMON AT THIS TIME - $RECALL,U+10,8$)
        EVENT  SIGNAL=SPECMSG
WAIT    WAIT
        BRANCH LABEL=WAIT
        ENDTXT
```

ALOGTSO message generation deck

```
ALOGTSO  MSGTXT
***********************************************************************
* The following deck is used to logon to TSO.  The userid's to be     *
* used are assumed to be WSIM01, WSIM02, and WSIM03.                  *
*                                                                     *
* Comment sections in this deck are boxed with asterisks (*).         *
*                                                                     *
* The following switches, counters, and areas are referenced in this  *
* message deck:                                                       *
*                                                                     *
*       Switches:  NSW2   (tells TSOCTRL the terminal is still        *
*                          active.  Used in this deck just in case    *
*                          logon is unusually time consuming)         *
*                                                                     *
*       Counters:         (there are no counters referenced)          *
*                                                                     *
*       Areas:     U+0    (userid work area, length 6)                *
*                                                                     *
* Message generation statements which may need changing to match your *
* requirements are marked with the word "CHANGE".                     *
***********************************************************************
* The first section composes the TSO userid by placing the last two   *
* digits of the LU name on the end of "WSIM", thus the TSO userid's   *
* are WSIM01, WSIM02, and WSIM03.                                     *
***********************************************************************
         WAIT  TIME=10
         DATASAVE AREA=U+2,TEXT=($ID,4$)
         DATASAVE AREA=U+0,TEXT=(WSIM)

***********************************************************************
* The following IF checks for the end of page indicator (***) and if  *
* seen, then CLEAR is called to press enter to wrap the page.         *
***********************************************************************
0        IF    TEXT=('08'***),LOC=C-5,THEN=C-CLEAR,
               STATUS=HOLD,DELAY=CANCEL

***********************************************************************
* The CMND is processed to establish the session between the status   *
* LU and TSO.  Change the MODE and RESOURCE to match local system     *
* requirements.                                                       *
***********************************************************************
         CMND  COMMAND=INIT,DATA=($RECALL,U+0,6$),
               MODE=D6327802,RESOURCE=TSO
*                   ========          === <----------------------CHANGE

***********************************************************************
* The next section checks for an incoming message with the text       *
* "PASSWORD".  When it arrives, processing branches to PSWGO.  For    *
* all other incoming messages (specifically incoming BIND images,     *
* etc.), processing holds up at WAIT1.  The BRANCH LABEL=WAIT1 is     *
* included because an incoming bind will reset any active waits.      *
***********************************************************************
CHKPSWD  LABEL
1        IF    LOC=B+0,SCAN=YES,TEXT=(PASSWORD),THEN=B-PSWGO,
               STATUS=HOLD,DELAY=CANCEL
WAIT1    WAIT
         BRANCH LABEL=WAIT1
```

```
***********************************************************************
* The following section enters the userid password.  The password is  *
* constructed using the character string provided at "yourpw" and the *
* last two digits of the userid.  Thus if the letters WSIM were       *
* placed at "yourpw", WSIM02's password would be "WSIM02".            *
*                                                                     *
* NOTE: if the installation of TSO about to be logged onto auto-      *
* matically enters SESSION MANAGER MODE, have your systems programmer *
* provide a TSO PROCEDURE that will turn SESSION MANAGER OFF.         *
***********************************************************************
PSWGO    LABEL
         TEXT  (WSIM$RECALL,U+4,2$)
*               ====== <-----------------------------------------CHANGE
         ENTER
         SETSW NSW1=ON

***********************************************************************
* The following IF statements determine whether the logon was         *
* successful (READY), or unsuccessful (LOGON REJECTED), and           *
* processing will branch accordingly.  If a message arrives that      *
* contains neither of the two, then the LU logging on will continue   *
* to wait at WAIT2.                                                   *
***********************************************************************
1        IF    LOC=B+0,SCAN=YES,TEXT=(READY),THEN=B-ENDLOG,
               STATUS=HOLD,DELAY=CANCEL
2        IF    LOC=B+0,SCAN=YES,TEXT=(LOGON REJECTED,),THEN=B-RECON,
               STATUS=HOLD,DELAY=CANCEL
WAIT2    WAIT
         BRANCH LABEL=WAIT2

***********************************************************************
* If the logon was rejected then an attempt to reconnect will be      *
* processed.  Processing branches back to CHKPSWD.                    *
***********************************************************************
RECON    TEXT  (LOGON $RECALL,U+0,6$ RECONNECT)
         ENTER
         BRANCH LABEL=CHKPSWD

***********************************************************************
* In the event a page end is detected (***), then IF 0 calls the      *
* following segment to clear the screen.                              *
***********************************************************************
CLEAR    LABEL
         CLEAR
         RETURN

***********************************************************************
* If a READY was received then processing branches to ENDLOG, which   *
* reformats the time of day (AFORTIME) and writes a message out to    *
* the operator screen and log indicating that TSO is responding.      *
***********************************************************************
ENDLOG   LABEL
         CALL  NAME=AFORTIME
         WTO  (+TSO IS NOW RESPONDING - $RECALL,U+10,8$)
         LOG  (+TSO IS NOW RESPONDING - $RECALL,U+10,8$)
         ENDTXT
***********************************************************************
```

ACHKTSO message generation deck

```
ACHKTSO MSGTXT
***********************************************************************
* THIS MESSAGE DECK IS USED BY THE TSO TERMINALS TO MONITOR THE       *
* AVAILABILITY OF THE TSO SYSTEM.                                     *
*                                                                     *
* COMMENT SECTIONS IN THIS DECK ARE BOXED WITH ASTERISKS (*).         *
*                                                                     *
* THE FOLLOWING SWITCHES, COUNTERS, AND AREAS ARE REFERENCED IN THIS  *
* MESSAGE DECK:                                                       *
*                                                                      *
*        SWITCHES:  NSW2   (SET BY THE LU EXECUTING THIS DECK, IT      *
*                           INDICATES THE STATUS OF THE TSO SYSTEM)    *
*                   SW1    (SET WHEN "READY" IS RETURNED FROM TSO)     *
*                   SW2    (SET WHEN RESPONSE TIME WAS GREATER THAN    *
*                           DEFINED SERVICE LEVEL COMMITMENT)          *
*                                                                      *
*        COUNTERS:         (THERE ARE NO COUNTERS USED IN THIS DECK)   *
*                                                                      *
*        AREAS:     N+0    (NETWORK SPECIAL MESSAGE AREA, LENGTH 60)   *
*                   U+10   (REFORMATTED TIME OF DAY, LENGTH 8)         *
*                                                                      *
* MESSAGE GENERATION STATEMENTS WHICH MAY NEED CHANGING TO MATCH YOUR *
* REQUIREMENTS ARE MARKED WITH THE WORD "CHANGE".                      *
************************************************************************
* THE FOLLOWING ON CONDITION IS SET TO PREPARE THE LU IN THE EVENT     *
* IT HANGS.  ATTENT WILL CAUSE AN ATTN KEY PROCESSING TO OCCUR.        *
************************************************************************
        ON    EVENT=ATTENT,THEN=B-ATTNKEY

************************************************************************
* THE FOLLOWING IF WILL CLEAR THE SCREEN WHENEVER THE INCOMING         *
* IS "***".                                                            *
************************************************************************
0       IF    TEXT=('08'***),LOC=C-5,THEN=C-CLEAR,
                   STATUS=HOLD,DELAY=CANCEL

************************************************************************
* THE FOLLOWING SECTION MEASURES THE RESPONSE TIME OF THE TSO SYSTEM   *
* TO AN EXEC STATEMENT THAT INVOKES THE CLIST "TIMECHK".  THE CLIST    *
* IS SIMPLY:                                                           *
*                   PROC 0 COUNT(5)                                    *
*                   DO WHILE &COUNT > 0                                *
*                   SET COUNT = &COUNT-1                               *
*                   END                                                *
*                   TIME                                               *
*                                                                      *
* ENSURE THIS CLIST IS IN A DATASET CORRESPONDING TO THE NAME GIVEN    *
* ON THE TEXT STATEMENT.                                               *
*                                                                      *
* PROCESSING OF THIS DECK IS AS FOLLOWS:                               *
*                                                                      *
*      - NSW2 IS SET ON TO TELL TSOCTLR THAT THE TSO TERMINAL IS       *
*        STILL ACTIVE.                                                 *
*      - ALL THE DEVICE SWITCHES ARE SET OFF BY SETSW SW=OFF.  THIS    *
*        EFFECTS ONLY THE SWITCHES FOR THE TSO TERMINALS, NOT TSOCTRL. *
*      - EVENT TIMELAPT IS RESET.                                      *
*      - EVENT TIMELAPT IS POSTED WITH TIME=5.  THE VALUE FOR TIME IS  *
*        MAXIMUM RESPONSE TIME ALLOWED ACCORDING TO THE LOCAL SERVICE  *
*        COMMITMENT.                                                   *
*      - IF 1 SETS SW1 IF THE MESSAGE RETURNED IS THE EXPECTED READY.  *
*      - IF 2 SETS SW2 IF EVENT=TIMELAPT HAS BEEN POSTED.  THIS WOULD  *
*        MEAN THE MAXIMUM RESPONSE TIME EXPIRED BEFORE THE RESPONSE    *
*        WAS RECEIVED.                                                 *
*      - IF 3 EXECUTES BADMSG IF BOTH SW1 (READY RECEIVED) AND         *
*        SW2 (TIMELAPT POSTED: RESPONSE TOO SLOW)                      *
*      - IF 4 BRANCHES BACK TO "REPEAT" IF THE SW1 IS ON, INDICATING   *
*        THE "READY" WAS RECEIVED WITHIN THE TIME LIMIT IMPOSED.       *
*      - IF 5 GUARDS AGAINST THE RARE OCCURANCE OF RECEIVING THE END   *
*        PAGE "***" AFTER THE "READY".                                 *
************************************************************************
REPEAT  LABEL
        TEXT  (EXEC ''WSIM.AVMON.CLIST((TIMECHK))'')
*                    =========================== <---------------CHANGE
        ENTER
        SETSW NSW2=ON
        SETSW SW=OFF
        CANCEL EVENTTAG=TIMELAPT
        EVENT RESET=TIMELAPT
        EVENT POST=TIMELAPT,TIME=5
*                             = <-----------------------------CHANGE
1       IF    TEXT=(READY),LOC=RU+0,SCAN=YES,THEN=SW1(ON)
2       IF    EVENT=TIMELAPT,THEN=SW2(ON)
3       IF    LOC=SW1&SW2,THEN=E-BADMSG
4       IF    LOC=SW1,THEN=B-REPEAT
WAIT    WAIT
5       IF    LOC=SW1,WHEN=IMMED,THEN=B-REPEAT
        BRANCH  LABEL=WAIT
***********************************************************************
* "BADMSG" IS EXECUTED WHEN THE RESPONSE TIME WAS TOO SLOW.  IT IS    *
* EXECUTED ON THE "THEN" LEG OF IF 3.                                 *
***********************************************************************
BADMSG  LABEL
        DATASAVE  AREA=U+12,TEXT=($TOD,6$)
        DATASAVE  AREA=U+10,
        TEXT=($RECALL,U+12,2$:$RECALL,U+14,2$:$RECALL,U+16,2$)
        DATASAVE AREA=N+0,
        TEXT=(-TSO HAS BAD RESPONSE TIME AT $RECALL,U+10,8$)
        EVENT SIGNAL=SPECMSG
        SETSW SW2=OFF
        EVENT RESET=TIMELAPT
        RETURN
***********************************************************************
* WHEN PROMPTED TO PERFORM THESE ACTIONS, THE LU SENDS THE ATTN KEY   *
* (COMMAND=SIGNAL).  IT THEN LOOKS FOR THE "READY" TO BE RETURNED.    *
* IF IT IS, THEN THE TERMINAL HAS RECOVERED AND THE TSO CONTROLLING   *
* LU IS ALERTED OF THIS VIA TSOREC.                                   *
***********************************************************************
ATTNKEY CMND COMMAND=SIGNAL,SENSE='00010000'
6       IF    TEXT=(READY),LOC=RU+0,SCAN=YES,THEN=CONT,STATUS=HOLD
        DELAY TIME=F1
        WAIT
        DEACT IFS=(6)
        EVENT SIGNAL=TSOREC
        ON EVENT=ATTENT,THEN=B-ATTNKEY
        BRANCH LABEL=REPEAT
        RETURN
***********************************************************************
* "CLEAR" IS CALLED WHENEVER THE END OF PAGE INDICATOR ("***") IS     *
* DETECTED BY IF 0.                                                   *
***********************************************************************
CLEAR   LABEL
        CLEAR
        RETURN
        ENDTXT
```

AVMON STL procedures

The following examples are STL procedures for the sample AVMON configuration. Refer to *WSim User's Guide* for the location of these data sets on the WSim distribution tape.

```
/*-------------------------------------------------------------------*/
/* AVMON Overview                                                    */
/*                                                                   */
/* This version of AVMON is a collection of STL procedures that      */
/* may be used to monitor the performance and status of your         */
/* network resources.  AVMON does this by having simulated           */
/* terminals log onto the resources and check the status on a        */
/* periodic basis.  For example, AVMON as it appears here logs       */
/* onto TSO and NetView; the terminals in session with TSO check     */
/* the response time of TSO, the terminals in session with           */
/* NetView issue D NET,ID=resource_name commands and check the       */
/* status that is returned.                                           */
/*                                                                    */
/* AVMON works on three levels:                                       */
/*                                                                    */
/* 1) Monitoring LUs go into session with the systems and perform    */
/*    their checking functions.  This example of AVMON checks        */
/*    only TSO and NetView, but you may modify AVMON to monitor      */
/*    more systems if you'd like.  Only one LU is in session with    */
/*    an application at any given time, but AVMON provides a pool    */
/*    of three LUs for each application.  If an LU hangs,            */
/*    for whatever reason, AVMON will detect this and start the      */
/*    next LU in the pool.  If a pool is exhausted, AVMON will       */
/*    detect this as well and refresh the pools by restarting        */
/*    the whole network.                                              */
/*                                                                    */
/* 2) The system pools are controlled by "controlling LUs", one      */
/*    for every system being monitored.  These LUs do not go into    */
/*    session with anything, they simply check the condition of      */
/*    the monitoring LUs and take action when problems arise.        */
/*                                                                    */
/* 3) A network-wide controlling LU keeps watch on the entire        */
/*    AVMON network, managing the activities of the simulation       */
/*    at this high level.                                             */
/*                                                                    */
/* It is also important to note that the information logged to the   */
/* WSim log data set is essentially the same between the two          */
/* versions of AVMON. Therefore, this manuals instructions on how to */
/* run ITPLL and ITPRESP are applicable in any case.                  */
/*                                                                    */
/* This STL program is comprised of several sections:                 */
/*                                                                    */
/*   1) A constants declaration section.  This will allow you to     */
/*      configure AVMON to your requirements without having to       */
/*      make modifications to the STL code itself.  An explanation   */
/*      of each constant is provided in comment blocks in the        */
/*      constants sections.                                           */
/*                                                                    */
/*   2) A variable declaration section.  This allows the STL         */
/*      translator program to know how it should allocate the        */
/*      variables named in the program.  These should not be         */
/*      modified unless you have a solid understanding of the STL    */
/*      program, the STL language, and the STL Translator.           */
/*                                                                    */
/*   3) A table definition section.  Many aspects of AVMON will      */
/*      differ from user to user.  The STL version of AVMON puts     */
/*      as many of these user-specific items in tables for easier    */
/*      modification.  These tables are discussed in comment blocks  */
/*      in the table section.                                         */
/*                                                                    */
/*   4) Individual STL procedures.  These procedures should not      */
/*      require modification unless you're tailoring your AVMON to   */
/*      something special.  Take the time to understand the overall  */
/*      flow of AVMON before you make modification.  Slight changes  */
/*      can have drastic results.                                     */
/*                                                                    */
/* Please take note:                                                  */
/*                                                                    */
/*                                                                    */
/*   o  This version of AVMON works in a manner different from the   */
/*      version written in the WSim scripting language.  Do not      */
/*      attempt to mix any portion of this program with the          */
/*      original AVMON.                                               */
/*                                                                    */
/*   o  If you make any modifications to this STL program, be sure   */
/*      to translate it before you use it in your WSim simulations.  */
/*      Changes made to STL must be translated before they will be   */
/*      seen in your simulation run.                                  */
/*                                                                    */
/*   o  If you choose to write some additional procedures for this   */
/*      program, be sure to include them in this file before you     */
/*      translate them using the STL Translator.  Separate trans-    */
/*      lations may result in WSim resources, such as counters or    */
/*      save areas, being used for different reasons.  This could    */
/*      easily upset the operation of AVMON.                          */
/*                                                                    */
/* WSim network configuration requirements:                           */
/*                                                                    */
/*   The network configuration that you supply to run AVMON should   */
/*   be patterned after the following example.  This example is a    */
/*   VTAMAPPL configuration, but you may configure a domain          */
/*   simulation if you'd prefer. The name on the NTWRK statement     */
/*   MUST be "AVMON".                                                 */
/*                                                                    */
/*                                                                    */
/*   AVMON    NTWRK HEAD='AVAILABILITY MONITOR',                      */
/*   *-------------------------------------------------------------* */
/*   * The NTWRK statement operands follow:                         * */
/*   *-------------------------------------------------------------* */
/*                  DELAY=F1,                                         */
/*                  UTI=100,                                          */
/*                  INIT=SEC,                                         */
/*                  MSGTRACE=NO, <--- Change if desired               */
/*                  STLTRACE=NO, <--- Change if desired               */
/*                  REPORT=NONE,                                      */
/*                  OPTIONS=(NORRLOG),                                */
/*                  BUFSIZE=3000                                      */
/*   *-------------------------------------------------------------* */
/*   * The INCLUDE and PATH statements are next:                    * */
/*   *-------------------------------------------------------------* */
/*            INCLUDE SAVEMSG,NETVMSG,REFTIME,AVSTATS,SPECMSG,        */
/*                    REINIT,NOWOK,CLEARIT,ATTNKEY,PARSPROC,TIMER    */
/*   1        PATH  ACHKNETV                                          */
/*   2        PATH  ACHKTSO                                           */
/*   *-------------------------------------------------------------* */
/*   * The network controller LU is defined in this manner.  The   * */
/*   * name on the LU must be "NETCTRL".  The name of the VTAMAPPL * */
/*   * may be any valid VTAM APPL name and must be defined to      * */
/*   * VTAM and activated before running AVMON.                    * */
/*   *-------------------------------------------------------------* */
/*   APPLNET  VTAMAPPL                                                */
/*   NETCTRL  LU    LUTYPE=LU0,                                       */
/*                  FRSTTXT=ACTRLNET                                  */
/*   *-------------------------------------------------------------* */
/*   * The NetView controller LU.  The name on the LU must be      * */
/*   * "NETVCTRL".  The name of the VTAMAPPL may be any valid VTAM * */
/*   * APPL name and must be defined to VTAM and activated before  * */
/*   * running AVMON.                                               * */
/*   *-------------------------------------------------------------* */
/*   APPLNC   VTAMAPPL                                                */
/*   NETVCTRL LU    LUTYPE=LU0,                                       */
/*                  FRSTTXT=AMONNETV                                  */
/*   *-------------------------------------------------------------* */
/*   * The NetView monitoring LU pool follows.  The LU names you   * */
/*   * use must match the NetView operator IDs WSim will log on to.* */
/*   * These names must also be provided in the table "netvtble"   * */
/*   * along with the associated passwords (see "netvtble" for     * */
/*   * instructions).  The VTAMAPPL names may be of your choosing, * */
/*   * and should be defined to VTAM and activated prior to running* */
/*   * AVMON.                                                       * */
/*   *-------------------------------------------------------------* */
/*   APPLN1   VTAMAPPL                                                */
/*   NVOPER1  LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),             */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*   APPLN2   VTAMAPPL                                                */
/*   NVOPER2  LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),             */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*   APPLN3   VTAMAPPL                                                */
/*   NVOPER3  LU    LUTYPE=LU2,FRSTTXT=ALOGNETV,PATH=(1),             */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*   *-----------------------------------------------------------* */
/*   * The TSO controller LU.  The name on the LU must be          * */
/*   * "TSOCTRL".  The name of the VTAMAPPL may be any valid VTAM  * */
/*   * APPL name and must be defined to VTAM and activated before  * */
/*   * running AVMON.                                              * */
/*   *-----------------------------------------------------------* */
/*   APPLTC   VTAMAPPL                                                */
/*   TSOCTRL  LU    LUTYPE=LU0,                                       */
/*                  FRSTTXT=AMONTSO                                   */
/*   *-----------------------------------------------------------* */
/*   * The TSO monitoring LU pool follows.  The LU names you       * */
/*   * use must match the TSO IDs names WSim will log on to.       * */
/*   * These names must also be provided in the table "tsotable"   * */
/*   * along with the associated passwords (see "tsotable" for     * */
/*   * instructions).  The VTAMAPPL names may be of your choosing, * */
/*   * and should be defined to VTAM and activate prior to running * */
/*   * AVMON.                                                      * */
/*   *-----------------------------------------------------------* */
/*   APPLT1   VTAMAPPL                                                */
/*   TSOID1   LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),              */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*   APPLT2   VTAMAPPL                                                */
/*   TSOID2   LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),              */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*   APPLT3   VTAMAPPL                                                */
/*   TSOID3   LU    LUTYPE=LU2,FRSTTXT=ALOGTSO,PATH=(2),              */
/*                    MAXSESS=(0,1),DISPLAY=(24,80)                   */
/*                                                                    */
/*--------------------------------------------------------------------*/
/*--------------------------------------------------------------------*/
/* The following STL statement, along with an "STLTRACE=YES" in       */
/* the network configuration, will place STL trace records into your */
/* log dataset.  This should be done for debugging purposes only:    */
/* when you put AVMON into production, you should NOT log STL trace  */
/* records.  Coding "STLTRACE=NO" will inhibit the logging of        */
/* STL trace records.  You need not change the "@program" statement. */
/*--------------------------------------------------------------------*/
```

Constant declarations

```
 @program=avmontrc
/*--------------------------------------------------------------------*/
/* Constants in AVMON STL dictate how AVMON will operate.  Between   */
/* these constants and the table definition section, AVMON can be    */
/* completely configured.  No modification of the STL procedures is  */
/* necessary.                                                         */
/*                                                                    */
/* Constant values are either integer or character in nature.  AVMON */
/* does not use any binary, or bit, constants.  Be sure to express   */
/* integer constants as number values, and enclose string constants  */
/* in single quotes.                                                  */
/*--------------------------------------------------------------------*/
                                           /*------------------------*/
 constant achktso_suspend_time  180        /* Time (in seconds)      */
                                           /* between TSO terminal   */
                                           /* checks of TSO system.  */
                                           /*------------------------*/
```

```
                                         /*-------------------------*/
constant max_response_time       3       /* Maximum TSO response   */
                                         /* time allowed (in       */
                                         /* seconds).              */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant message_log_device   'DISK'     /* Method of message log- */
                                         /* ging.  Enter either    */
                                         /* DISK or TAPE.          */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant netview_logon_appl   'CNM01'    /* APPL to be used when   */
                                         /* logging on to NetView. */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant netview_mode_table   'D4A32782' /* LOGMODE table to be    */
                                         /* used when logging on   */
                                         /* to NetView.            */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant netview_msg_operator 'OPER5'    /* NetView operator where */
                                         /* AVMON special messages */
                                         /* should be sent.        */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant reinit_stop_time     '17:30'    /* Time beyond which no   */
                                         /* 'reinit' of AVMON will */
                                         /* be performed.          */
                                         /* (00:01 to 24:00)       */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant resource_loop         180       /* Total time for NetView */
                                         /* monitoring LU to cycle */
                                         /* through all the re-    */
                                         /* sources in RESTBLE     */
                                         /* (in seconds).          */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant start_time           '08:00'    /* Time at which you wish */
                                         /* AVMON to begin moni-   */
                                         /* toring.  Must be in    */
                                         /* the 'HH:MM' format.    */
                                         /* (00:01 to 24:00)       */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant stats_msg_cycle_time   60       /* Time (in minutes)      */
                                         /* between availability   */
                                         /* status report.         */
                                         /*-------------------------*/

                                         /*-------------------------*/
constant stop_time            '18:00'    /* Time at which you wish */
                                         /* AVMON to stop monitor- */
                                         /* ing.  Must be in the   */
                                         /* 'HH:MM' format.        */
                                         /* (00:01 to 24:00)       */
                                         /*-------------------------*/
```

```
                                           /*------------------------*/
constant tso_logon_appl       'TSO'        /* APPL to be used when   */
                                           /* logging on to TSO.     */
                                           /*------------------------*/

                                           /*------------------------*/
constant tso_mode_table       'D4A32782'   /* LOGMODE table to be    */
                                           /* used when logging on   */
                                           /* to TSO.                */
                                           /*------------------------*/
```

Variable declarations

```
/*---------------------------------------------------------------*/
/* The variable declaration section is used to tell STL how to      */
/* allocate the resources named by STL variables.  For example,     */
/* if you want a counter to be shared across all resources, you     */
/* would declare the integer variable as "shared" so that STL       */
/* would translate the variable into a WSim network counter.        */
/*                                                                  */
/* These declarations should NOT be modified.  Their present        */
/* allocation is critical to the operation of AVMON.                */
/*---------------------------------------------------------------*/
```

Integer variables

```
                                           /*------------------------*/
integer unshared status_offset             /* Used when determining  */
                                           /* the offset into the    */
                                           /* VTAM IST486I message   */
                                           /* the keyword "STATUS="  */
                                           /* (VTAM 3.1.1 or before).*/
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared state_offset              /* Used when determining  */
                                           /* the offset into the    */
                                           /* VTAM IST486I message   */
                                           /* the keyword "STATE="   */
                                           /* (VTAM 3.2 and after).  */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared table_index               /* Used to control the    */
                                           /* selection of the       */
                                           /* resource name from     */
                                           /* RESTBLE.               */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared stats_msg_counter         /* Used to track the time */
                                           /* since the last print-  */
                                           /* ing of the status      */
                                           /* report.                */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared terminal_suspend_time     /* Passed to procedure    */
                                           /* NOWOK to control       */
                                           /* suspend after special  */
                                           /* message.               */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared tso_recovery_wait         /* DO loop control value  */
                                           /* used when checking to  */
                                           /* see if TSO recovered   */
                                           /* after ATTENTION used.  */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared total_elapsed_seconds     /* Total seconds between  */
                                           /* two time stamps passed */
                                           /* to "timer" procedure.  */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared start_hr                  /* Hour value of time of  */
                                           /* day when TSO message   */
                                           /* transmitted.           */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared start_min                 /* Minute value of time   */
                                           /* of day when TSO message*/
                                           /* transmitted.           */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared start_sec                 /* Second value of time   */
                                           /* day when TSO message   */
                                           /* transmitted.           */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared stop_hr                   /* Hour value of time of  */
                                           /* day when READY received*/
                                           /* by TSO terminal.       */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared stop_min                  /* Minute value of time of*/
                                           /* day when READY received*/
                                           /* by TSO terminal.       */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared stop_sec                  /* Second value of time of*/
                                           /* day when READY received*/
                                           /* by TSO terminal.       */
                                           /*------------------------*/

                                           /*------------------------*/
integer unshared line_offset               /* Used in procedure      */
                                           /* SAVEMSG to determine   */
                                           /* the offset into the    */
                                           /* screen of the NetView  */
                                           /* separator line (----). */
                                           /*------------------------*/

                                           /*------------------------*/
integer shared overall_time                /* Used to track the      */
                                           /* total time AVMON has   */
                                           /* been in operation.     */
                                           /*------------------------*/

                                           /*------------------------*/
integer shared netview_pool_lu_number      /* Used to indicate which */
                                           /* LU from the NetView    */
                                           /* pool is currently      */
                                           /* active.                */
                                           /*------------------------*/

                                           /*------------------------*/
```

```
integer shared netview_availability       /* Used to track the      */
                                          /* total time the NetView */
                                          /* LUs have been active.  */
                                          /*------------------------*/

                                          /*------------------------*/
integer shared tso_pool_lu_number         /* Used to indicate which */
                                          /* LU from the TSO pool   */
                                          /* is currently active.   */
                                          /*------------------------*/

                                          /*------------------------*/
integer shared tso_availability           /* Used to track the      */
                                          /* total time the TSO     */
                                          /* LUs have been active.  */
                                          /*------------------------*/
```

String variables

```
                                          /*------------------------*/
string unshared resource_status           /* Used to store the      */
                                          /* status of the resource */
                                          /* after the NetView LU   */
                                          /* has issued the D NET   */
                                          /* command.               */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared temp                      /* Used as a temporary    */
                                          /* variable for special   */
                                          /* message processing.    */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared termtype                  /* Used by procedure      */
                                          /* NOWOK to specify what  */
                                          /* terminal type has      */
                                          /* regained activity.     */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared start                     /* Used by procedure      */
                                          /* TIMER to calculate     */
                                          /* TSO response time.     */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared stop                      /* Used by procedure      */
                                          /* TIMER to calculate     */
                                          /* TSO response time.     */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared token_string              /* UTBL entry passed to   */
                                          /* PARSPROC procedure to  */
                                          /* parse out id and pass- */
                                          /* word.                  */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared id_token                  /* Application ID value   */
                                          /* parsed from utbl entry.*/
                                          /* Created by PARSPROC.   */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared message_area              /* Used in SAVEMSG to     */
                                          /* determine if the       */
                                          /* current screen line    */
                                          /* contains a VTAM message*/
                                          /* of concern.            */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared vtam_message              /* The entire VTAM message*/
                                          /* is saved in this.      */
                                          /*------------------------*/

                                          /*------------------------*/
string unshared password_token            /* ID password value      */
                                          /* parsed from utbl entry.*/
                                          /* Created by PARSPROC.   */
                                          /*------------------------*/

                                          /*------------------------*/
string shared time_of_day                 /* Used to store the      */
                                          /* current time of day.   */
                                          /*------------------------*/

                                          /*------------------------*/
string shared close_log_indicator         /* Used to store the      */
                                          /* content of the constant*/
                                          /* "message_log_device"   */
                                          /* so that logic testing  */
                                          /* may be performed.      */
                                          /*------------------------*/

                                          /*------------------------*/
string shared queue                       /* Special message queue. */
                                          /*------------------------*/

                                          /*------------------------*/
string shared netv_queue                  /* Special message to be  */
                                          /* sent to NetView oper-  */
                                          /* ator queue.            */
                                          /*------------------------*/
```

Bit variables

```
                                          /*------------------------*/
bit unshared msg_located                  /* Used to indicate the   */
                                          /* VTAM message was found.*/
                                          /*------------------------*/

                                          /*------------------------*/
bit shared netview_still_active           /* Used to indicate to    */
                                          /* AMONNETV that the      */
                                          /* NetView LU is still    */
                                          /* active.                */
                                          /*------------------------*/

                                          /*------------------------*/
bit shared tso_still_active               /* Used to indicate to    */
                                          /* AMONTSO that the TSO   */
                                          /* LU is still active.    */
                                          /*------------------------*/

                                          /*------------------------*/
bit shared tso_recovered                  /* Used to indicate to    */
                                          /* AMONTSO that the TSO   */
                                          /* LU has recovered after */
                                          /* using ATTENTION key.   */
                                          /*------------------------*/
```

Table declarations

```
/*-------------------------------------------------------------------*/
/* MSGUTBLs are used by AVMON to provide flexibility.  You may add   */
/* to the tables if you wish, and AVMON will automatically know the  */
/* size.  This is another way STL makes WSim scripting easier.       */
/*                                                                   */
/* All tables contain string data.  Therefore all entries must be    */
/* enclosed in single quotes.  What may go in each entry is really   */
/* determined by what it will be used for.  For example, the         */
/* table "restble" contains the names of VTAM resources to be        */
/* monitored.  These entries should be valid resource names.         */
/*-------------------------------------------------------------------*/

/*-------------------------------------------------------------------*/
/* The table "restble" contains the names of network resources you   */
/* want AVMON to periodically check.  AVMON does this by issuing     */
/* a D NET,ID=resource_name,NONE command and seeing what the result  */
/* is.                                                               */
/*                                                                   */
/* You may put as many resource names in this table as you wish.     */
/* Please be aware that the time it takes to cycle through this      */
/* entire list is determined by the constant "resource_loop".        */
/* Ensure the value for "resource_loop" is at least that of the      */
/* number of names in this table.  Otherwise the delay between       */
/* messages will be zero.                                            */
/*-------------------------------------------------------------------*/

restble:   msgutbl
           'TSO01'
           'CNM01'
           'IMS01'
           'CICS01'
           'APPL1'
           'APPL2'
           endutbl


/*-------------------------------------------------------------------*/
/* The table titled "ctrltble" is used to list the names of the      */
/* LUs that control the NetView and TSO LU pools.  If you modify     */
/* AVMON to monitor more systems than that, you may add the names    */
/* of these new controlling LUs to this table.                       */
/*                                                                   */
/* The LU that provides overall control of AVMON uses this table to  */
/* release the subsystem controller LUs from their initial quiesced  */
/* state.                                                            */
/*-------------------------------------------------------------------*/

ctrltble:  msgutbl
           'TSOCTRL'
           'NETVCTRL'
           endutbl


/*-------------------------------------------------------------------*/
/* The table "netvtble" contains the NetView operator IDs and        */
/* passwords that will be used by the LUs that actually log onto     */
/* NetView and issue the D NET commands.  This configuration of      */
/* AVMON provides three LUs in this pool.  If one goes inactive for  */
/* any reason, AVMON will quiesce it and use the next LU in the pool.*/
/* If you wish to add more LUs to the pool you may do so.  Simply    */
/* provide more VTAMAPPL-LU groups to your NTWRK configuration, and  */
/* add to this list.                                                 */
/*                                                                   */
/* The format of this table is:                                      */
/*                                                                   */
/*            'operator_id,password'                                */
/*                                                                  */
/* It must be in this order, and a comma must separate the values   */
/* within the table entry.                                          */
/*                                                                  */
/* Once again, please make sure WSim LU names you coded in your     */
/* network configuration match these operator IDs.                  */
/*------------------------------------------------------------------*/

netvtble:  msgutbl
           'NVOPER1,NVPASS1'
           'NVOPER2,NVPASS2'
           'NVOPER3,NVPASS3'
           endutbl


/*------------------------------------------------------------------*/
/* The table "tsotable" operates in an identical manner to the table */
/* called "netvtble".  Please review the explanation of that table   */
/* for information about this table.                                 */
/*------------------------------------------------------------------*/

tsotable:  msgutbl
           'TSOID1,TSOPASS1'
           'TSOID2,TSOPASS2'
           'TSOID3,TSOPASS3'
           endutbl
```

ACTLRNET procedure

```
/*------------------------------------------------------------------*/
/* The STL procedures follow from here to the end of this program.   */
/*------------------------------------------------------------------*/

/*------------------------------------------------------------------*/
/* Procedure Name: ACTLRNET                                          */
/* Used by: Network controller LU (NETCTRL)                          */
/*                                                                  */
/* The procedure "actlrnet" is used to provide overall AVMON         */
/* coordination and control.  It's primary function is to            */
/* start and stop AVMON at the specified times, and to provide the   */
/* status report at the specified intervals.                         */
/*------------------------------------------------------------------*/
actrlnet:  msgtxt
           /*------------------------------------------------------*/
           /* Begins by transferring constant value to variable value*/
           /* for logic testing later, and setting up the special    */
           /* message "on signaled" condition.  Once done, it calls  */
           /* the procedure "reftime" to check the current time of   */
           /* day.                                                   */
           /*------------------------------------------------------*/
           close_log_indicator=message_log_device
           on signaled('SPECMSG') then execute specmsg
           call reftime

           /*------------------------------------------------------*/
           /* AVMON will not start monitoring until the specified    */
           /* start time has passed (constant "start_time").  It     */
           /* checks to see if it's okay to begin, and if not then   */
           /* it calculates the delay and suspends until that time.  */
           /*------------------------------------------------------*/
           if substr(time_of_day,1,5)<start_time then
            do
             start=tod(6)
```

```
  stop=substr(start_time,1,2)||substr(start_time,4,2)||'00'
  call timer
  suspend(total_elapsed_seconds)
 end
/*------------------------------------------------------*/
/* Monitoring has now begun.  A message is put out to the */
/* console and log saying this is so.                   */
/*------------------------------------------------------*/
call reftime
say 'Begin Monitoring at' time_of_day,
     month()||'/'|| day()||'/'|| year()
log 'Begin Monitoring at' time_of_day,
     month()||'/'|| day()||'/'|| year()
/*------------------------------------------------------*/
/* A quick loop through "ctrltble" releases all the sub-  */
/* system controller LUs.                               */
/*------------------------------------------------------*/
do i=0 to utblmax(ctrltble)
 opcmnd 'a '||utbl(ctrltble,i)||',release'
end
suspend(10)
call reftime
/*------------------------------------------------------*/
/* The following "do while" loop checks to see if it is   */
/* time to stop AVMON.  If not, then the AVMON controller */
/* waits 180 seconds (3 minutes).  Upon exiting the wait  */
/* it updates the overall time counter and the statistics */
/* message counter by 3 minutes.                        */
/*------------------------------------------------------*/
do while substr(time_of_day,1,5)<stop_time
 suspend(180)
 overall_time=overall_time+3
 stats_msg_counter=stats_msg_counter+3
 /*-----------------------------------------------------*/
 /* A check to see if it's time to print out the        */
 /* statistics message.  If so, then the time is re-    */
 /* formatted and "avstats" is called.  If not, then    */
 /* processing simply loops back to the check for end   */
 /* of monitoring.                                      */
 /*-----------------------------------------------------*/
 if stats_msg_counter>=stats_msg_cycle_time then
  do
   call reftime
   call avstats
  end
 else nop
 call reftime
end
/*------------------------------------------------------*/
/* The current time of day has passed the designated stop */
/* time, so AVMON will shut down.  One last message is    */
/* issued and AVMON is stopped.                         */
/*------------------------------------------------------*/
call reftime
say 'End Monitoring at' time_of_day,
     month()||'/'||day()||'/'||year()
log 'End Monitoring at' time_of_day,
     month()||'/'||day()||'/'||year()
/*------------------------------------------------------*/
/* The following command will end the WSim job.          */
/*------------------------------------------------------*/
opcmnd 'zend'
quiesce
endtxt
```

AMONNETV procedure

```
/*------------------------------------------------------------------*/
/* Procedure Name: AMONNETV                                         */
/* Used by: NetView pool controller LU (NETVCTRL)                   */
/*                                                                  */
/* The procedure called 'amonnetv' is used to monitor and           */
/* control the NetView LU pool.  The primary purpose of this        */
/* procedure is to keep tabs on the LU pool, checking to see if the */
/* current LU is still active, and if not then releasing the next   */
/* LU from the pool.  If the pool becomes exhausted, this procedure */
/* will request the reinitialization of the entire network (thus    */
/* refreshing the pool).                                            */
/*------------------------------------------------------------------*/
amonnetv:  msgtxt
           /*------------------------------------------------------*/
           /* NETVCTRL is initially quiesced so that NETCTRL can    */
           /* get control of the overall timing.  When NETCTRL      */
           /* releases NETVCTRL, the variable that tracks which LU  */
           /* from the LU pool is currently active is initialized to */
           /* zero (necessary because user tables are zero-based).  */
           /* The type of terminals being controlled by this device */
           /* is designated as "NetView".  The NetView ID name      */
           /* and password are pulled from the table and set for    */
           /* "parsproc" to parse out the values separately.        */
           /* NETVCTRL then releases the first LU from the pool.    */
           /*------------------------------------------------------*/
           quiesce
           netview_pool_lu_number=0
           termtype='NetView'
           table_string=utbl(netvtble,netview_pool_lu_number)
           call parsproc
           opcmnd 'a '||id_token||',release'
           /*------------------------------------------------------*/
           /* NETVCTRL now goes into a loop that checks the status  */
           /* of the monitoring LU.  The loop repeats itself every  */
           /* 180 seconds (3 minutes).                              */
           /*------------------------------------------------------*/
           do forever
            suspend(180)
            /*-----------------------------------------------------*/
            /* The variable 'netview_still_active' is a switch that */
            /* is turned on by the monitoring LU to indicate it is  */
            /* still active.  If the switch is on then the count of */
            /* NetView availability is incremented by 3 minutes and */
            /* the switch is turned off.  If NETVCTRL returns and   */
            /* the switch is back on, then the monitoring LU must   */
            /* still be active.                                     */
            /*-----------------------------------------------------*/
            if netview_still_active=on then
             do
              netview_availability=netview_availability+3
              netview_still_active=off
             end
            else
            /*-----------------------------------------------------*/
            /* The switch was off, so the monitoring LU is con-     */
            /* sidered to be inactive at this time.  The following  */
            /* actions are now taken:                               */
            /*  o A message telling of the inactivity is placed     */
            /*    on the message queue (with a two-byte hex length  */
            /*    header in front of each message).                 */
            /*  o 'SPECMSG' is signalled, which triggers NETCTRL    */
            /*    to begin special message processing               */
            /*  o A 60-second wait is then started.                 */
            /*-----------------------------------------------------*/
```

```
do
 call reftime
 temp='NetView term',
      id_token,
      'not responding',
      time_of_day
 queue=queue||hex(length(temp),2)||temp
 signal 'SPECMSG'
 suspend(60)
 /*-----------------------------------------------------*/
 /* A check is again made to see if the monitoring LU   */
 /* is active.  If so, then an integer value of 120     */
 /* seconds is set and the procedure "nowok" is called. */
 /*-----------------------------------------------------*/
 if netview_still_active=on then
  do
   terminal_suspend_time=120
   call nowok
  end
 else
 /*-----------------------------------------------------*/
 /* The monitoring LU was still not active after the    */
 /* first one minute wait, so it will be given another  */
 /* one minute wait to become active.  If after the     */
 /* wait it's active again, then an integer value of    */
 /* 60 seconds is set and "nowok" is called.            */
 /*-----------------------------------------------------*/
  do
   suspend(60)
   if netview_still_active=on then
    do
     terminal_suspend_time=60
     call nowok
    end
   else
    /*--------------------------------------------------*/
    /* At this point the monitoring LU is considered    */
    /* permanently inactive.  It is time to quiesce     */
    /* this LU and release the next in the pool.        */
    /*                                                  */
    /* A check is made to see if the pool is exhausted. */
    /* If not, then the following actions are taken:    */
    /*  - a special message is put out                  */
    /*  - the inactive terminal is quiesce              */
    /*  - the pool LU number is incremented             */
    /*  - the next ID/password string is extracted      */
    /*  - "parsproc" splits the tokens out              */
    /*  - the next LU is released                       */
    /*--------------------------------------------------*/
    do
     if netview_pool_lu_number<utblmax(netvtble) then
      do
       call reftime
       temp='NetView term',
            id_token,
            'inactive',
            time_of_day
       queue=queue||hex(length(temp),2)||temp
       signal 'SPECMSG'
       opcmnd 'a '||id_token||',quiesce'
       netview_pool_lu_number=netview_pool_lu_number+1
       table_string=utbl(netvtble,netview_pool_lu_number)
       call parsproc
       opcmnd 'a '||id_token||',release'
      end
     else
```

```
              /*-------------------------------------------------*/
              /* The pool was found to be exhausted, so a        */
              /* reinitialization of AVMON is required to        */
              /* refresh the pool.  "avstats" is called to put   */
              /* out a status report before reinitialization,    */
              /* a special message is added to the queue, and    */
              /* the procedure "reinit" is called.               */
              /*-------------------------------------------------*/
               do
                call reftime
                call avstats
                temp='NetView Pool Exhaused. Reinit AVMON',
                     time_of_day
                call reinit
               end
               /*------------------------------------------------*/
               /* End reinitialization processing.               */
               /*------------------------------------------------*/
             end
             /*--------------------------------------------------*/
             /* End permanently inactive LU processing           */
             /*--------------------------------------------------*/
           end
           /*----------------------------------------------------*/
           /* End temporarily inactive LU processing, 2nd wait.  */
           /*----------------------------------------------------*/
         end
         /*------------------------------------------------------*/
         /* End temporarily inactive LU processing, 1st wait.    */
         /*------------------------------------------------------*/
       end
       /*--------------------------------------------------------*/
       /* End "do forever" loop.                                 */
       /*--------------------------------------------------------*/
       endtxt
```

ALOGNETV procedure

```
/*-------------------------------------------------------------------*/
/* Procedure Name: ALOGNETV                                          */
/* Used by: NetView monitoring LUs.                                  */
/*                                                                   */
/* The procedure called 'alognetv' is used by the monitoring         */
/* LUs to log on to NetView.                                         */
/*-------------------------------------------------------------------*/


alognetv:  msgtxt
           /*--------------------------------------------------------*/
           /* The LU falls into a quiesced state to begin with.  When*/
           /* it's released by the controlling LU it sends an        */
           /* initself.                                              */
           /*--------------------------------------------------------*/
           quiesce
           initself(netview_logon_appl,netview_mode_table)
           /*--------------------------------------------------------*/
           /* The following loop periodically check to see if the    */
           /* NetView logon screen has been received.  When it has,  */
           /* then processing continues.                             */
           /*--------------------------------------------------------*/
           do while index(screen,'ENTER LOGON INFORMATION')=0
            suspend(3)
           end
           /*--------------------------------------------------------*/
           /* The appropriate ID/password string is pulled from the  */
           /* table and passed to "parsproc", which separates and    */
```

```
          /* returns the individual components.  The ID and password*/
          /* values are then typed on the screen, along with the    */
          /* "NO" for running the initial NetView procedure.  The   */
          /* screen is then transmitted and this LU waits until     */
          /* either the NetView separator line (-----), or the      */
          /* string "ALREADY LOGGED ON" is returned.                */
          /*--------------------------------------------------------*/
          table_string=utbl(netvtble,netview_pool_lu_number)
          call parsproc
          type id_token
          nl
          type password_token
          nl
          nl
          nl
          type 'NO'
          transmit using enter and wait until onin,
            index(screen,'--------')>0 |,
            index(screen,'ALREADY LOGGED ON')>0

          /*--------------------------------------------------------*/
          /* A check is made to see if "ALREADY LOGGED ON" was      */
          /* returned.  If so, this LU can do no more.  A message   */
          /* is put out and the LU is quiesced.                     */
          /*--------------------------------------------------------*/
          if index(screen,'ALREADY LOGGED ON')>0 then
           do
            call reftime
            say id_token 'unable to logon.  Time:' time_of_day
            log id_token 'unable to logon.  Time:' time_of_day
            quiesce
           end
          else nop

          /*--------------------------------------------------------*/
          /* The LU is now logged onto NetView.  "AUTOWRAP" is      */
          /* typed and transmitted, with a wait for the NetView     */
          /* message "DSI082I", which indicates autowrap has been   */
          /* started.  When this is returned, a message is put      */
          /* out, and processing continues to the procedure         */
          /* "achknetv".                                            */
          /*--------------------------------------------------------*/
          type 'AUTOWRAP YES'
          transmit using enter and wait until onin,
            index(screen,'DSI082I')>0
          call reftime
          say 'NetView terminal' id_token 'logged on' time_of_day
          log 'NetView terminal' id_token 'logged on' time_of_day
          endtxt
```

ACHKNETV procedure

```
/*---------------------------------------------------------------------*/
/* Procedure Name: ACHKNETV                                            */
/* Used by: NetView monitoring LUs.                                    */
/*                                                                     */
/* The procedure 'achknetv' is executed by the monitoring LUs          */
/* to check the status of the resources you've named in 'restble'.    */
/* Its primary function is to issue a 'D NET,ID=resource' command     */
/* to NetView and report on the status it sees in return.              */
/*---------------------------------------------------------------------*/


achknetv:  msgtxt
           /*--------------------------------------------------------*/
           /* The 'on signaled' condition 'NETVMSG' is activated.    */
           /* This will cause the special message to be sent to the  */
           /* designated NetView operator.  The resource table index */
           /* number is initialized to zero.  A check is made to see */
           /* if the special message queue has any messages left     */
           /* over from a prior pool LU that could not complete the  */
           /* sending of all messages.  If messages remain, "netvmsg"*/
           /* is called to process them.                             */
           /*--------------------------------------------------------*/
           on signaled('NETVMSG') then call netvmsg
           table_index=0
           if length(netv_queue)>0 then call netvmsg
           do forever
            /*-------------------------------------------------------*/
            /* The monitoring LU has entered a "do forever" loop that*/
            /* will process the D NET messages until told to quit.   */
            /* A D NET command is typed on the screen and transmitted*/
            /* to NetView.  It waits until it sees either the 314I   */
            /* message indicating the end of that display group, or  */
            /* the 453I message indicating a name unknown to VTAM.   */
            /* When either is returned, the "still active" switch is */
            /* turned on and the procedure "savemsg" is called.      */
            /* "savemsg" interrogates the screen and saves the line  */
            /* that has the status indicator.                        */
            /*-------------------------------------------------------*/
            type 'd net,id='||utbl(restble,table_index)||',none'
            transmit using enter and wait until onin,
                             index(ru,'IST314I')>0 |,
                             index(ru,'IST453I')>0
            netview_still_active=on
            call savemsg
            /*-------------------------------------------------------*/
            /* Upon return from "savemsg" we now have the results of */
            /* the D NET command.  Because VTAM changed the message  */
            /* format of IST486I between VTAM 3.1.1 and VTAM 3.2,    */
            /* we need to check for either "STATUS=" in VTAM 3.1.1   */
            /* or "CURRENT STATE" in VTAM 3.2                        */
            /*-------------------------------------------------------*/
            status_offset=index(vtam_message,'STATUS=')
            state_offset=index(vtam_message,'CURRENT STATE')
            if status_offset>0 | state_offset>0 then
            /*-------------------------------------------------------*/
            /* Somewhere in the saved VTAM message there is the      */
            /* status indicator.  Because of the message format      */
            /* difference, where it's located depends on the level   */
            /* of VTAM that returned the message.  A check is made   */
            /* and the status indicator is saved.                    */
            /*-------------------------------------------------------*/
             do
              if status_offset>0 & state_offset=0 then,
                resource_status=substr(vtam_message,status_offset+8,10)
              else
                resource_status=substr(vtam_message,state_offset+16,10)
              /*-----------------------------------------------------*/
              /* A check is made to see if "ACTIV" was NOT the value */
              /* of the status indicator.  A special message needs   */
              /* to be issued.                                       */
              /*-----------------------------------------------------*/
              if resource_status¬='ACTIV' then
               do
                call reftime
                temp='Status of',
                     utbl(restble,table_index),
                     ':' resource_status 'Time:' time_of_day
                queue=queue||hex(length(temp),2)||temp
                signal 'SPECMSG'
               end
              else nop
             end
            else
            /*-----------------------------------------------------*/
            /* Neither "STATUS=" or "CURRENT STATE" was found, so    */
            /* this means the resource is unknown to VTAM.  Usually  */
            /* this implies the resource is not activated.  A special*/
            /* message is generated.                                 */
            /*-----------------------------------------------------*/
             do
              call reftime
              temp=utbl(restble,table_index),
                   'not activated.  Time:' time_of_day
              queue=queue||hex(length(temp),2)||temp
              signal 'SPECMSG'
             end
            /*-----------------------------------------------------*/
            /* Some post-D NET processing is required.  The string   */
            /* variables are cleaned out, the suspend time is        */
            /* calculated and invoked, and if the end of the resource*/
            /* table has been reached, the index number is reset     */
            /* to zero to start over again.                          */
            /*-----------------------------------------------------*/
            vtam_message=''
            resource_status=''
            suspend_time=resource_loop/(utblmax(restble)+1)
            suspend(suspend_time)
            table_index=table_index+1
            if table_index>utblmax(restble) then table_index=0
           end
           endtxt
```

AMONTSO procedure

```
/*-----------------------------------------------------------------*/
/* Procedure Name: AMONTSO                                           */
/* Used by: TSO pool controller LU (TSOCTRL)                         */
/*                                                                   */
/* The procedure 'amontso' is used to monitor and control the TSO    */
/* LU pool.  The primary purpose of this procedure is to keep tabs   */
/* on the LU pool, checking to see if the current LU is still active,*/
/* and if not then releasing the next LU from the pool.  If the pool */
/* becomes exhausted, this procedure will request the reinitial-     */
/* ization of the entire network (thus refreshing the pool).         */
/*-----------------------------------------------------------------*/


amontso:   msgtxt
           /*-----------------------------------------------------*/
           /* TSOCTRL is initially quiesced so that NETCTRL can     */
           /* get control of the overall timing.  When NETCTRL      */
           /* releases TSOCTRL, the variable that tracks which LU   */
           /* from the LU pool is currently active is initialized to */
           /* zero (necessary because user tables are zero-based).  */
           /* The type of terminals being controlled by this device */
           /* is designated as "TSO".  The TSO ID name              */
           /* and password are pulled from the table and passed to  */
           /* "parsproc" to parse out the values separately.        */
           /* TSOCTRL then releases the first LU from the pool.     */
           /*-----------------------------------------------------*/
           quiesce
           tso_pool_lu_number=0
           termtype='TSO'
```

```
table_string=utbl(tsotable,tso_pool_lu_number)
call parsproc
opcmnd 'a '||id_token||',release'
/*------------------------------------------------------*/
/* TSOCTRL now goes into a loop that checks the status    */
/* of the monitoring LU.  The loop repeats itself every   */
/* 180 seconds (3 minutes).                               */
/*------------------------------------------------------*/
do forever
 suspend(180)
 /*-----------------------------------------------------*/
 /* The variable 'tso_still_active' is a switch that      */
 /* is turned on by the monitoring LU to indicate it is   */
 /* still active.  If the switch is on then the count of  */
 /* TSO availability is incremented by 3 minutes and      */
 /* the switch is turned off.  If TSOCTRL returns and     */
 /* the switch is back on, then the monitoring LU must    */
 /* still be active.                                      */
 /*-----------------------------------------------------*/
 if tso_still_active=on then
  do
   tso_availability=tso_availability+3
   tso_still_active=off
  end
 else
 /*-----------------------------------------------------*/
 /* The switch was off, so the monitoring LU is con-      */
 /* sidered to be inactive at this time.  The following   */
 /* actions are now taken:                                */
 /*  o A message telling of the inactivity is placed      */
 /*    on the message queue (with a two-byte hex length   */
 /*    header in front of each message).                  */
 /*  o 'SPECMSG' is signalled, which triggers NETCTRL     */
 /*    to begin special message processing                */
 /*  o A 60-second wait is then started.                  */
 /*-----------------------------------------------------*/
  do
   call reftime
   temp='TSO term',
        id_token,
        'not responding',
        time_of_day
   queue=queue||hex(length(temp),2)||temp
   signal 'SPECMSG'
   suspend(60)
   /*----------------------------------------------------*/
   /* A check is again made to see if the monitoring LU   */
   /* is active.  If so, then a parameter value of 120    */
   /* seconds is set and the procedure "nowok" is called. */
   /*----------------------------------------------------*/
   if tso_still_active=on then
    do
     terminal_suspend_time=120
     call nowok
    end
   else
   /*----------------------------------------------------*/
   /* The monitoring LU was still not active after the    */
   /* first one minute wait, so it will be given another  */
   /* one minute wait to become active.  If after the     */
   /* wait its active again, then a parameter value of    */
   /* 60 seconds is set and "nowok" is called.            */
   /*----------------------------------------------------*/
    do
     suspend(60)
```

```
if tso_still_active=on then
 do
  terminal_suspend_time=60
  call nowok
 end
else
/*-------------------------------------------------*/
/* The monitoring LU was still not active, so a      */
/* final measure will be taken.  The signal "ATTENT" */
/* is given to get the TSO LU to issue its attention */
/* key.  The controlling LU checks every 10 seconds  */
/* to see if the monitoring LU has recovered.        */
/*-------------------------------------------------*/
 do
  signal 'ATTENT'
  do tso_recovery_wait = 10 to 60 by 10
   suspend(10)
   if tso_recovered=on then leave
  end
  tso_recovered=off
  /*-----------------------------------------------*/
  /* Did TSOCTRL leave the loop because the monitor- */
  /* ing LU became active, or because the loop time  */
  /* expired?  If the total recovery wait time was   */
  /* greater than or equal to 60 seconds, then       */
  /* the loop time expired and the monitoring LU     */
  /* must still be inactive.  If it is inactive,     */
  /* then it will be considered permanently inactive,*/
  /* and TSOCTRL will attempt to start another LU    */
  /* from the pool.                                  */
  /*-----------------------------------------------*/
  if tso_recovery_wait>=60 then
   do
    if tso_pool_lu_number<utblmax(tsotable) then
     do
      /*-------------------------------------------*/
      /* The TSO LU pool still has some available    */
      /* LUs, so TSOCTRL will start the next LU from */
      /* the pool.  A special message is put out     */
      /* saying this.  The inactive LU is quiesced,  */
      /* the pool index number is incremented, the   */
      /* next ID/password value is retrieved, and    */
      /* the next LU is released from its quiesced   */
      /* state.                                      */
      /*-------------------------------------------*/
      call reftime
      temp='TSO term',
           id_token,
           'inactive',
           time_of_day
      queue=queue||hex(length(temp),2)||temp
      signal 'SPECMSG'
      opcmnd 'a '||id_token||',quiesce'
      tso_pool_lu_number=tso_pool_lu_number+1
      table_string=utbl(tsotable,tso_pool_lu_number)
      call parsproc
      opcmnd 'a '||id_token||',release'
     end
    else
     do
      /*-------------------------------------------*/
      /* The TSO LU pool is exhausted, so all of     */
      /* AVMON will need to be reinitialized to      */
      /* refresh the pool.  A status report is       */
      /* requested, a special message is built,      */
      /* and the procedure "reinit" is called.       */
```

```
                  /*----------------------------------------------*/
                  call reftime
                  call avstats
                  temp='TSO Pool Exhaused. Reinit AVMON',
                       time_of_day
                  call reinit
                 end
                 /*-----------------------------------------------*/
                 /* End reinitialization processing.             */
                 /*-----------------------------------------------*/
               end
               /*-------------------------------------------------*/
               /* End permanently inactive LU processing.        */
               /*-------------------------------------------------*/
              else
               do
                /*------------------------------------------------*/
                /* The TSO LU recovered after issuing its        */
                /* attention key, so processing passes to        */
                /* the procedure "nowok" with a suspend time     */
                /* of zero.                                      */
                /*------------------------------------------------*/
                terminal_suspend_time=0
                call nowok
               end
               /*-------------------------------------------------*/
               /* End recovered by attention key processing.     */
               /*-------------------------------------------------*/
             end
             /*---------------------------------------------------*/
             /* End attention key processing.                    */
             /*---------------------------------------------------*/
           end
           /*-----------------------------------------------------*/
           /* End temporarily inactive LU, 2nd wait.             */
           /*-----------------------------------------------------*/
         end
         /*-------------------------------------------------------*/
         /* End temporarily inactive LU, 1st wait.               */
         /*-------------------------------------------------------*/
       end
       /*---------------------------------------------------------*/
       /* End "do forever" loop.                                 */
       /*---------------------------------------------------------*/
       endtxt
```

ALOGTSO procedure

```
/*------------------------------------------------------------------*/
/* Procedure Name: ALOGTSO                                          */
/* Used by: TSO monitoring LUs                                      */
/*                                                                  */
/* The procedure "alogtso" is used by the TSO monitoring LUs to log  */
/* onto the system.                                                 */
/*------------------------------------------------------------------*/


alogtso:   msgtxt
           /*-----------------------------------------------------*/
           /* The monitoring LU is initially quiesced so that overall*/
           /* timing can be established.  It will be released by the */
           /* LU controlling the TSO pool.  The onin condition is    */
           /* established to protect against the end-of-page         */
           /* condition.  The first ID/password string is retrieved  */
           /* and passed to "parsproc".  An initself is sent and     */
           /* this LU waits until it sees the password entry screen. */
```

```
/*-----------------------------------------------------------*/
quiesce
onin substr(screen,coff()-4,3)='***' &,
     substr(screen,coff()-5,1)¬='*' then call clearit
table_string=utbl(tsotable,tso_pool_lu_number)
call parsproc
initself(tso_logon_appl,tso_mode_table,id_token)
do while index(screen,'PASSWORD')=0
 suspend(2)
end
/*-----------------------------------------------------------*/
/* The password screen has arrived, so the password is    */
/* entered.  The LU will wait until either the "ready"    */
/* indicator or "rejected" indicator is returned.         */
/*-----------------------------------------------------------*/
type password_token
transmit using enter and wait until onin,
   index(screen,'READY')>0 |,
   index(screen,'REJECTED')>0
if index(screen,'REJECTED')>0 then
 do
  /*---------------------------------------------------------*/
  /* The LU has experienced a rejection of its logon      */
  /* attempt, so it will try a reconnect.  It waits for   */
  /* the password screen to come back and it resends      */
  /* with the reconnect indicator.                        */
  /*---------------------------------------------------------*/
  type 'LOGON '||id_token||' RECONNECT'
  transmit using enter and wait until onin,
     index(screen,'PASSWORD')>0
  type password_token
  transmit using enter and wait until onin,
     index(screen,'READY')>0 |,
     index(screen,'REJECTED')>0
  if index(screen,'REJECTED')>0 then
   do
    /*-------------------------------------------------------*/
    /* The LU was rejected a second time, so therefore    */
    /* the ID must be in use somewhere else.  A special   */
    /* message is built and SPECMSG is signaled.  This    */
    /* LU is quiesced.  The controlling LU will eventually*/
    /* detect this and start the next LU in the pool.     */
    /*-------------------------------------------------------*/
    call reftime
    temp=id_token 'unable to logon.  Time:' time_of_day
    queue=queue||hex(length(temp),2)||temp
    signal 'SPECMSG'
    quiesce
   end
  else nop
 end
else nop
/*-----------------------------------------------------------*/
/* The LU is now logged on to TSO.  A message is sent     */
/* indicating this, and processing goes to "achktso".     */
/*-----------------------------------------------------------*/
call reftime
say 'TSO terminal' id_token 'logged on' time_of_day
log 'TSO terminal' id_token 'logged on' time_of_day
endtxt
```

ACHKTSO procedure

```
/*------------------------------------------------------------------*/
/* Procedure Name: ACHKTSO                                          */
/* Used by: TSO monitoring LUs                                      */
/*                                                                  */
/* The procedure "achktso" is used by the TSO monitoring LUs to     */
/* check the status of TSO by invoking a simple CLIST that indicates */
/* the time of day.  WSim does not supply this CLIST with AVMON,    */
/* but it's simple enough to add.  Here's what the CLIST should     */
/* look like:                                                       */
/*                                                                  */
/*          PROC 0 COUNT(5)                                         */
/*          DO WHILE &COUNT > 0                                     */
/*          SET COUNT = &COUNT-1                                    */
/*          END                                                     */
/*          TIME                                                    */
/*                                                                  */
/* Be sure to put this in the dataset specified on the "type"       */
/* statement.                                                       */
/*                                                                  */
/* The time that elapses between the sending of the message and     */
/* the receipt of the "ready" indicator is compared against the     */
/* value of the "max_response_time" constant.  If the response time  */
/* is within acceptable limits, no message is issued.  Longer times  */
/* will be flagged.                                                 */
/*------------------------------------------------------------------*/


achktso:   msgtxt
           /*------------------------------------------------------*/
           /* The onin condition to hit clear when the end-of-page   */
           /* indicator is seen is established.  The "on signaled"   */
           /* action whenever "ATTENT" is seen is established.  The  */
           /* LU then enters a "do forever" loop that continues to   */
           /* check the status of TSO until told to stop.            */
           /*------------------------------------------------------*/
           onin substr(screen,coff()-4,3)='***' &,
                substr(screen,coff()-5,1)¬='*' then call clearit
           on signaled('ATTENT') then call attnkey
           do forever
            /*-----------------------------------------------------*/
            /* The command to execute the CLIST is typed,            */
            /* but not yet sent.  The start time-of-day stamp is set,*/
            /* and the message is sent to TSO with the instructions  */
            /* to wait until it sees "READY".  When "READY" is       */
            /* returned, the stop time stamp is set and the switch   */
            /* indicating continued activity is set on.  The         */
            /* procedure "timer" is called to calculate the time     */
            /* difference between the start and stop time stamps.    */
            /*-----------------------------------------------------*/
            type "exec 'wsim.avmon.clist(timechk)'"
            start=tod(6)
            transmit using enter and wait until,
                onin index(ru,'READY')>0
            stop=tod(6)
            tso_still_active=on
            call timer
            /*-----------------------------------------------------*/
            /* A check is made to see if the response time is greater*/
            /* than acceptable.  If so, then a special message is    */
            /* sent indicating this.                                 */
            /*-----------------------------------------------------*/
            if total_elapsed_seconds > max_response_time then
             do
              call reftime
              temp='TSO has bad response time at' time_of_day
              queue=queue||hex(length(temp),2)||temp
              signal 'SPECMSG'
             end
            else nop
            /*------------------------------------------------------*/
            /* This LU is suspended for the specified period of time.*/
            /*------------------------------------------------------*/
            suspend(achktso_suspend_time)
           end
           endtxt
```

AVMON utility procedures

```
/*-----------------------------------------------------------------*/
/* The following STL procedures are called or executed by the      */
/* primary procedures shown above.  They perform a variety of      */
/* activities.                                                     */
/*-----------------------------------------------------------------*/
```

SAVEMSG

```
/*-----------------------------------------------------------------*/
/* Procedure Name: SAVEMSG                                         */
/* Used by: NetView pool controller LU (NETVCTRL)                  */
/* Called or executed from: ACHKNETV                               */
/*                                                                 */
/* The procedure 'savemsg' locates the VTAM message IST088I,       */
/* IST453I, or IST486I and saves it away for later processing      */
/*                                                                 */
/* This is a fairly complex task on a NetView screen because the   */
/* screen wraps bottom to top.  Thus it's possible to have two     */
/* messages of concern on the screen at the same time: the most    */
/* recent and the one prior to that.  There's no way of knowing    */
/* which one is the most recent except that it should be nearest   */
/* the separator line (------).  This is how this procedure works: */
/* it locates the separator line and works backwards from there    */
/* until it finds the appropriate message.                         */
/*-----------------------------------------------------------------*/


savemsg:   msgtxt
           /*------------------------------------------------------*/
           /* The separator line is located on the screen and the   */
           /* offset is saved.  If no line is found, it's assumed   */
           /* the line is off the end of the page (this will happen */
           /* when the information fills the screen, but does not   */
           /* need to wrap...the line simply drops off the screen). */
           /* In this case the string "???" is used to start the    */
           /* backwards process.                                    */
           /*                                                       */
           /* The cursor is moved two places to the left of the start*/
           /* of the line.  It's on this column that the messages   */
           /* appear.  A loop is begun that will continue until     */
           /* the message is found.                                 */
           /*------------------------------------------------------*/
           line_offset=index(screen,'------')
           if line_offset=0 then line_offset=index(screen,'???')
           cursor(line_offset-2)
           do while msg_located=off
            /*------------------------------------------------------*/
            /* The cursor is moved up one line and a check is made   */
            /* for any of the three pertinent messages.  If found,   */
            /* then the whole line is saved.                         */
            /*------------------------------------------------------*/
            cursor("up")
            message_area=substr(screen,coff(),19)
```

```
          if substr(message_area,1,7)='IST088I' |,
             substr(message_area,1,7)='IST486I' |,
             substr(message_area,13,7)='IST453I' then
           do
            vtam_message=substr(screen,coff(),80)
            msg_located=on
           end
          else if line_offset=coff()+2 then
           do
           /*----------------------------------------------------*/
           /* The cursor has looped around the screen and is back  */
           /* to the starting point.  This means none of the       */
           /* messages could be found.  A message is issued saying */
           /* this, and "msg_located" is turned on so that the LU  */
           /* doesn't enter an infinite loop.                      */
           /*----------------------------------------------------*/
            call reftime
            say luid() 'could not find VTAM message at' time_of_day
            log luid() 'could not find VTAM message at' time_of_day
            msg_located=on
           end
          else nop
         end
         /*------------------------------------------------------*/
         /* The message has been found and the LU has left the     */
         /* loop.  "msg_located" is turned off and the cursor is   */
         /* returned to the home position.                         */
         /*------------------------------------------------------*/
         msg_located=off
         home
         return
         endtxt
```

SPECMSG

```
/*------------------------------------------------------------------*/
/* Procedure Name: SPECMSG                                          */
/* Used by: NETCTRL                                                 */
/* Called or executed from: ACTRLNET.  Any LU may issue the signal  */
/*                          that causes NETCTRL to execute this     */
/*                          procedure.                              */
/*                                                                  */
/* The msgtxt procedure 'specmsg' is executed frequently whenever   */
/* a special message needs processing.  The messages are placed on  */
/* a queue with a 2-byte hex header that identifies the length of   */
/* the following message.  This procedure strips the message from   */
/* the queue, issues it in the form of a console message and log    */
/* message, then places the message on the back of the NetView      */
/* message queue.  'NETVMSG' is then signaled to tell the NetView   */
/* monitoring LU to process the special message.  The message is    */
/* then removed from the queue.                                     */
/*------------------------------------------------------------------*/


specmsg:   msgtxt
           say substr(queue,3,c2d(substr(queue,1,2)))
           log substr(queue,3,c2d(substr(queue,1,2)))
           netv_queue=netv_queue||,
                      substr(queue,1,c2d(substr(queue,1,2))+2)
           queue=substr(queue,(c2d(substr(queue,1,2))+3))
           signal 'NETVMSG'
           on signaled('SPECMSG') then execute specmsg
           return
           endtxt
```

NETVMSG

```
/*------------------------------------------------------------------*/
/* Procedure Name: NETVMSG                                          */
/* Used by: NetView monitoring LUs                                  */
/* Called or executed from: ACHKNETV.  The signal that causes this  */
/*                          procedure to be called is issued from   */
/*                          the procedure "specmsg".                */
/*                                                                  */
/* The procedure 'netvmsg' is used by one of the NetView            */
/* monitoring LUs whenever the event 'NETVMSG' is signaled by       */
/* NETCTRL.  This generally occurs whenever a special message needs */
/* to be sent to the designated NetView operator.                   */
/*                                                                  */
/* The procedure works in this fashion:                             */
/*   o A 'do while' loop is set up to process while the length      */
/*     of the message queue is greater than zero.                   */
/*   o The special message is taken from the front of the queue     */
/*     and sent to the NetView operator.                            */
/*   o The message is removed from the front of the queue.          */
/*------------------------------------------------------------------*/


netvmsg:   msgtxt
           do while length(netv_queue)>0
            tab
            ereof
            type 'msg '||netview_msg_operator||' '||,
                  substr(netv_queue,3,c2d(substr(netv_queue,1,2)))
            transmit using enter
            suspend(1)
            netv_queue=substr(netv_queue,(c2d(substr(netv_queue,1,2))+3))
           end
           on signaled('NETVMSG') then call netvmsg
           post 'MSGCOMP'
           return
           endtxt
```

REFTIME

```
/*------------------------------------------------------------------*/
/* Procedure Name: REFTIME                                          */
/* Used by: all LUs                                                 */
/* Called or executed from: ACTLNET, AMONNETV, ALOGNETV, ACHKNETV,  */
/*                          AMONTSO, ALOGTSO, ACHKNETV, SAVEMSG,    */
/*                          NOWOK                                   */
/*                                                                  */
/* The procedure 'reftime' is called frequently throughout all      */
/* of AVMON.  It simply reformats the current time of day from the  */
/* standard "HHMMSS" to "HH:MM:SS".                                 */
/*------------------------------------------------------------------*/


reftime:   msgtxt
           time_of_day=substr(TOD(6),1,2)||':'||,
                       substr(TOD(6),3,2)||':'||,
                       substr(TOD(6),5,2)
           return
           endtxt
```

AVSTATS

```
/*------------------------------------------------------------------*/
/* Procedure Name: AVSTATS                                          */
/* Used by: NETCTRL, NETVCTRL, TSOCTRL                              */
/* Called or executed from: ACTLNET, AMONNETV, AMONTSO              */
/*                                                                  */
/* The procedure 'avstats' simply puts out an availability          */
/* report whenever:                                                 */
/*   o NETCTRL determines the time has come based on the value of    */
/*     'stats_msg_counter', or                                       */
/*   o One of the LU pools has been exhausted and AVMON reinitial-   */
/*     ization is necessary.                                         */
/*                                                                   */
/* NOTE: This msgtxt procedure may require changes if you add more   */
/*       subsystem monitoring responsibilities to your AVMON         */
/*       configuration.                                              */
/*-------------------------------------------------------------------*/

avstats:   msgtxt
           say '-------------------------------------'
           say ' AVMON Availability Report -' time_of_day
           say ' Total time of AVMON monitoring:' char(overall_time)
           say ' '
           say ' Subsystem availability times:'
           say '    Netview' char(netview_availability)
           say '    TSO' char(tso_availability)
           say '-------------------------------------'
           log '-------------------------------------'
           log ' AVMON Availability Report -' time_of_day
           log ' Total time of AVMON monitoring:' char(overall_time)
           log ' '
           log ' Subsystem availability times:'
           log '    Netview' char(netview_availability)
           log '    TSO' char(tso_availability)
           log '-------------------------------------'

           /*------------------------------------------------------*/
           /* The following IF determines the type of device you're  */
           /* logging to.  If you're logging to disk, then two       */
           /* successive 'E' commands are issued that save the       */
           /* dataset on a periodic basis.                           */
           /*                                                        */
           /* NOTE: make sure you specify DISP=MOD on your execution */
           /*       JCL for the LOGDD dataset.                       */
           /*                                                        */
           /* If you were logging to tape, you would not want to     */
           /* issue an 'E' command because that would cause the      */
           /* tape to rewind, destroying previously logged activity. */
           /*------------------------------------------------------*/
           if close_log_indicator='DISK' |,
              close_log_indicator='disk' then
            do
             opcmnd 'E'
             opcmnd 'E'
            end
           else nop
           stats_msg_counter=0
           return
           endtxt
```

REINIT

```
/*-------------------------------------------------------------------*/
/* Procedure Name: REINIT                                            */
/* Used by: NETVCTRL, TSOCTRL                                        */
/* Called or executed from: AMONNETV, AMONTSO                        */
/*                                                                   */
/* The procedure 'reinit' is called when AVMON needs to be           */
/* reinitialized to refresh the LU pools.                            */
/*-------------------------------------------------------------------*/

reinit:    msgtxt
           queue=queue||hex(length(temp),2)||temp
           signal 'SPECMSG'
           if substr(time_of_day,1,5)>=reinit_stop_time then
            do
```

```
  /*-----------------------------------------------------*/
  /* No restart of AVMON will be attempted because the    */
  /* time of day is past the latest restart time.  A      */
  /* special message is processed and the controller      */
  /* LU simply waits.                                     */
  /*-----------------------------------------------------*/
  temp='*** No restart of AVMON at this time:' time_of_day
  queue=queue||hex(length(temp),2)||temp
  signal 'SPECMSG'
  wait
 end
else
 do

  /*-----------------------------------------------------*/
  /* A restart of AVMON will be attempted.  A special     */
  /* message is processed indicating this.  The post and  */
  /* wait statements are allowing the NetView terminal    */
  /* to clean out its special message queue before the    */
  /* reinitialization.  Once done, AVMON is cancelled,    */
  /* initialized, and restarted.                          */
  /*-----------------------------------------------------*/
  temp='Restart of AVMON by' LUID() ':' time_of_day
  queue=queue||hex(length(temp),2)||temp
  reset 'MSGCOMP'
  signal 'SPECMSG'
  post 'MSGCOMP' after 10
  wait until posted('MSGCOMP')
  opcmnd 'c avmon'
  opcmnd 'i avmon'
  opcmnd 's avmon'
  wait
 end
return
endtxt
```

NOWOK

```
/*------------------------------------------------------------------*/
/* Procedure Name: NOWOK                                            */
/* Used by: NETVCTRL, TSOCTRL                                       */
/* Called or executed from: AMONNETV, AMONTSO                       */
/*                                                                  */
/* The procedure "nowok" is called whenever the controlling LU has  */
/* previously seen one of its monitoring LUs as inactive, but now   */
/* sees it has regained activity.  It's "now okay", hence the name. */
/* The controlling LU calling this procedure passes it two          */
/* parameters: the terminal type (via "termtype") and the amount of */
/* time the controlling LU should wait before returning to the      */
/* procedure "amonnetv" or "amontso".                               */
/*------------------------------------------------------------------*/

nowok:     msgtxt
           /*-------------------------------------------------------*/
           /* A select group is used to determine what kind of      */
           /* terminal type is now okay.  When the condition is     */
           /* met, the system availability counter is incremented   */
           /* by the 3 minutes that have expired, the "still-active" */
           /* switch is turned off, and a special message is issued. */
           /* The LU is then suspended for the period of time       */
           /* specified in the parameter passed to this procedure.  */
           /*-------------------------------------------------------*/
           select
            when termtype='netview' |,
                 termtype='NetView' |,
                 termtype='NETVIEW' then
             do
              /*----------------------------------------------------*/
              /* NetView terminal now okay                          */
              /*----------------------------------------------------*/
              netview_availability=netview_availability+3
              netview_still_active=off
              call reftime
              temp='NetView term',
                   id_token,
                   'responding again',
                   time_of_day
              queue=queue||hex(length(temp),2)||temp
             end

            when termtype='tso' | termtype='TSO' then
             do
              /*----------------------------------------------------*/
              /* TSO terminal now okay                              */
              /*----------------------------------------------------*/
              tso_availability=tso_availability+3
              tso_still_active=off
              call reftime
              temp='TSO term' id_token 'responding again',
                   time_of_day
              queue=queue||hex(length(temp),2)||temp
             end

            otherwise
             do
              /*----------------------------------------------------*/
              /* Terminal type now known to this procedure          */
              /*----------------------------------------------------*/
              temp='Undefined NOWOK termtype:' termtype
              queue=queue||hex(length(temp),2)||temp
             end
            end
           signal 'SPECMSG'
           suspend(terminal_suspend_time)
           return
           endtxt
```

CLEARIT

```
/*-------------------------------------------------------------------*/
/* Procedure Name: CLEARIT                                           */
/* Used by: TSO monitoring LUs                                       */
/* Called or executed from: ALOGTSO, ACHKTSO                         */
/*                                                                   */
/* The procedure "clearit" simply clears the display screen.        */
/*-------------------------------------------------------------------*/

clearit:   msgtxt
           transmit using clear
           return
           endtxt
```

ATTNKEY

```
/*-------------------------------------------------------------------*/
/* Procedure Name: ATTNKEY                                           */
/* Used by: TSO monitoring LUs                                       */
/* Called or executed from: ACHKTSO                                  */
/*                                                                   */
/* The procedure "attnkey" simply simulates the attention key, then  */
/* waits for the receipt of "ready" from TSO.  When "ready" is       */
/* received, the terminal is active again and the switch indicating  */
/* the TSO terminal has recovered is turned on.                      */
/*-------------------------------------------------------------------*/

attnkey:   msgtxt
```

```
           snacmnd(signal,'00010000'x)
           wait until onin index(ru,'READY')>0
           tso_recovered=on
           return
           endtxt
```

PARSPROC

```
/*-------------------------------------------------------------------*/
/* Procedure Name: PARSPROC                                          */
/* Used by: NETVCTRL, TSOCTRL, NetView monitors, TSO monitors        */
/* Called or executed from: AMONNETV, ALOGNETV, AMONTSO, ALOGTSO     */
/*                                                                   */
/* The procedure "parsproc" receives a string taken from a table     */
/* that contains the ID and the associated password.  This procedure */
/* then strips each component, or token, from that string based on   */
/* the delimiter, which is a comma.                                  */
/*-------------------------------------------------------------------*/

parsproc:  msgtxt
           id_token=substr(table_string,1,index(table_string,',')-1)
           password_token=substr(table_string,index(table_string,',')+1,,
                          length(table_string))
           return
           endtxt
```

TIMER

```
/*-------------------------------------------------------------------*/
/* Procedure Name: TIMER                                             */
/* Used by: TSO monitoring LUs                                       */
/* Called or executed from: ACHKTSO                                  */
/*                                                                   */
/* The procedure "timer" calculates the difference between a start   */
/* time stamp and a stop time stamp.  It receives "start" and "stop" */
/* from the calling procedure and calculates the difference, even    */
/* if the time interval crossed minute, hour, or even day boundaries.*/
/*                                                                   */
/* The input to this procedure is START and STOP, both with format   */
/* HHMMSS.  The output from this procedure is TOTAL_ELAPSED_SECONDS, */
/* which is the total number of seconds between START and STOP.      */
/*-------------------------------------------------------------------*/

timer:  msgtxt
        start_hr=e2d(substr(start,1,2),2)
        start_min=e2d(substr(start,3,2),2)
        start_sec=e2d(substr(start,5,2),2)
        stop_hr=e2d(substr(stop,1,2),2)
        stop_min=e2d(substr(stop,3,2),2)
        stop_sec=e2d(substr(stop,5,2),2)
        /*-----------------------------------------------------------*/
        /* If the time interval crossed midnight then 24 must be     */
        /* added to the stop hour to allow for the later subtraction */
        /* of START_HR from STOP_HR.                                 */
        /*-----------------------------------------------------------*/
        if stop_hr<start_hr then
           stop_hr=stop_hr+24
         else
           nop
        /*-----------------------------------------------------------*/
        /* If STOP_MIN is less than START_MIN (as would occur if the */
        /* time interval crossed an hour) then 1 hour must be        */
        /* borrowed from STOP_HR and 60 minutes carried to STOP_MIN. */
        /*-----------------------------------------------------------*/
        if stop_min<start_min then
           do
            stop_hr=stop_hr-1
```

```
    stop_min=stop_min+60
   end
 else
   nop
/*-----------------------------------------------------------*/
/* If STOP_SEC is less than START_SEC (as would occur is the */
/* time interval crossed a minute) then 1 minute must be     */
/* borrowed from STOP_MIN and 60 seconds carried to STOP_SEC.*/
/*-----------------------------------------------------------*/
if stop_sec<start_sec then
   do
    stop_min=stop_min-1
    stop_sec=stop_sec+60
   end
 else
   nop
/*-----------------------------------------------------------*/
/* Simple subtraction of the hours, minutes, and seconds is  */
/* performed, then the total elapsed seconds is calculated.  */
/*-----------------------------------------------------------*/
total_elapsed_seconds=((stop_hr-start_hr)*3600)+,
                      ((stop_min-start_min)*60)+,
                      (stop_sec-start_sec)
return
endtxt
```

Chapter 26. Loglist examples

This topic contains loglists for selected samples in this book. All of the loglists in this topic were run with the RUN Loglist Utility command, unless otherwise noted.

INSTALL1 loglist

Figure 39 contains a portion of a loglist for the INSTALL1 sample, described in “WSim as a VTAM application (INSTALL1)” on page 281.

Figure 39. INSTALL1 loglist output

```
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME          NAME              NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
                                                 12021630 0096056  11000000  CNSL  0800 000000     75
  Workload Simulator (WSim) VERSION 1, RELEASE 1.0
--------------------------------------------------------------------------------------------------------------------------------------
                                                 12033054 0096056  11000000  CNSL  0800 000000     10
  I INSTALL1
--------------------------------------------------------------------------------------------------------------------------------------
                                                 12033159 0096056  11000000  CNSL  0800 000000     52
  ITP029I INITIALIZATION COMPLETE FOR NETWORK INSTALL1
--------------------------------------------------------------------------------------------------------------------------------------
                                                 12034359 0096056  11000000  CNSL  0800 000000      1
  S
--------------------------------------------------------------------------------------------------------------------------------------
                                                 12034700 0096056  11000000  CNSL  0800 000000     32
  ITP006I NETWORK INSTALL1 STARTED
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12034704 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00         0
  ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00003 OF DECK INSTMTXT
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035300 12035587 12035294 +XMIT  8000 880020     34   E2   INSTMTXT  00         2
   XMIT INITIATE SELF REQUEST
       TH   2C0000010001         FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=00   OAF=01   ODAI=0   SEQUENCE=1
       RH   0B8000   REQUEST    FM DATA-FM HEADER     ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   01068101 C4F4C1F3 F2F7F8F2 F308C9E3    D7C5C3C8 D6400000 00                  *..A.D4A327823.ITPECHO ...       *
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035587 12035587 12035587 +RECV  8000 080020     12   E2   INSTMTXT  00         3
   RECV INITIATE SELF RESPONSE
       TH   2C0001000001         FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=00   ODAI=0   SEQUENCE=1
       RH   8B8000   RESPONSE   FM DATA-FM HEADER     ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   010681                                                                   *..A                             *
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035587 0096056  11000000  MTRC  0100 083000     81   E2   INSTMTXT  00         4
  ITP421I INPUT     IF  0   (INSTMTXT 00001) NOT EVALUATED - NO SPECIFICATION MATCH
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035706 12035706 12035706  RECV  8000 080000    115   E2   INSTMTXT  00         5
   RECV BIND SESSION REQUEST
       TH   2D0001010000         FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=0
       RH   6B8000   REQUEST     SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   31010303 B1903080 008087C7 80000200    00000000 18500000 7E000007 C9E3D7C5   *..........GG.........&;.=...ITPE*
 00000020   C3C8D600 05000212 102008E6 E2C9D4C1    D7D7D360 12DF279F E4CFDE98 5409D5C5   *CHO........WSIMAPPL-....U..Q..NE*
 00000040   E3C14BC1 F0F1D40E 0DF3D5C5 E3C14BC9    E3D7C5C3 C8D62C0A 01084040 40404040   *TA.A01M..3NETA.ITPECHO....      *
 00000060   40402D09 08C4F4C1 F3F2F7F8 F2                                                *  ...D4A32782                   *

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME          NAME              NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
        BIND SESSION   FORMAT 0   TYPE=NON-NEGOTIABLE   FM PROFILE 3     TS PROFILE 3
           PRIMARY PROTOCOLS:   RU CHAINING=MULTIPLE   REQUEST MODE=IMMEDIATE   RESPONSE REQUESTED=DEF OR EXC   END BRACKET SENT
           SECONDARY PROTOCOLS: RU CHAINING=MULTIPLE   REQUEST MODE=IMMEDIATE   RESPONSE REQUESTED=EXCEPTION    END BRACKET NOT SENT
           COMMON PROTOCOLS:    SEGMENTS SUPPORTED     FM HEADERS NOT ALLOWED   BRACKETS RESET BETB      BRACKET TERMINATION RULE 1
                                ALTERNATE CODE SET NOT USED   HALF-DUPLEX FLIP-FLOP    RECOVERY RESPONSIBILITY=PRIMARY
                                CONTENTION WINNER=SECONDARY
           SECONDARY SEND PACING COUNT=NONE        SECONDARY RECEIVE PACING COUNT=NONE   ADAPTIVE SESSION PACING SUPPORTED
           SECONDARY MAXIMUM RU SEND SIZE=1024      PRIMARY MAXIMUM RU SEND SIZE=1536
           PRIMARY SEND PACING COUNT=NONE           PRIMARY RECEIVE PACING COUNT=NONE
           LU TYPE 2   DEFAULT SCREEN SIZE=024,080  ALTERNATE SCREEN SIZE=NONE
           PRIMARY LU NAME=ITPECHO                  CRYPTOGRAPHIC FIELD=NONE
           URC=0002121020
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035706 0096056  11000000  MTRC  0100 083000     74   E2   INSTMTXT  00         6
  ITP429I INPUT     IF  0   (INSTMTXT 00001) NOT MET - ELSE ACTION NOT CODED
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035725 12035768 12035706  XMIT  8000 880000     10   E2   INSTMTXT  00         7
   XMIT BIND SESSION RESPONSE
       TH   2D0001010000         FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=0
       RH   EB8000   RESPONSE    SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   31                                                                       *.                               *
--------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL         WSIMLU-1        12035775 12035775 12035775  RECV  8000 080000     10   E2   INSTMTXT  00         8
   RECV START DATA TRAFFIC REQUEST
       TH   2D0001010078         FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=120
       RH   6B8000   REQUEST     SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   A0                                                                       *.                               *
```

```
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12035775 0096056  11000000  MTRC  0100 083000     74   E2   INSTMTXT  00         9
  ITP429I INPUT     IF  0   (INSTMTXT 00001) NOT MET - ELSE ACTION NOT CODED

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12035775 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         0
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|                                                                                | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|                                                                                |13
14|                                                                                |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW(  1) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12035775 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        11
  ITP447I MSG GEN ENTERED: STMT# 00003 OF DECK INSTMTXT
  ITP450I BRANCH FROM STMT# 00003 OF DECK INSTMTXT TO STAY     AT 00001 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00003 OF DECK INSTMTXT
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12035775 12035818 12035775  XMIT  8000 880000     10   E2   INSTMTXT  00        14
   XMIT START DATA TRAFFIC RESPONSE
       TH   2D0001010078          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=120
       RH   EB8000   RESPONSE    SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   A0                                                                    *.                                *
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12035824 12035825 12035825  RECV  8000 080000    134   E2   INSTMTXT  00        15
   RECV (DATA) REQUEST
       TH   2C0001010001          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   0390C0   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   BEGIN-END BRACKET
       RU   F5C7114E 7F1D68E6 C5D3C3D6 D4C540E3    D640C9E3 D7C5C3C8 D64B1D60 40C5D5E3   *5G.+"..WELCOME TO ITPECHO..- ENT*
 00000020   C5D97EC5 C3C8D640 4040C3D3 C5C1D97E    D9C5E2E3 D6D9C540 4040F561 F67EE2E3   *ER=ECHO   CLEAR=RESTORE   5/6=ST*
 00000040   D9C9D5C7 40D9C5D7 C5C1E340 4040F97E    D9C5D7C5 C1E31150 50C5D5E3 C5D940C4   *RING REPEAT   9=REPEAT.&&ENTER D*
 00000060   C1E3C140 E3D640C5 C3C8D640 C2C5D3D6    E67A11D1 5F1D4013 115D7F1D F0         *ATA TO ECHO BELOW:.J–. ..)".0   *
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12035825 0096056  11000000  MTRC  0100 083000    130   E2   INSTMTXT  00        16
  ITP427I INPUT     IF  0   (INSTMTXT 00001) MET - THEN ACTION TAKEN: BRANCH FROM 00003 OF INSTMTXT TO GO       AT 00004 OF INSTMTXT

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12040288 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         1
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|                                                                                | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040288 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        18
  ITP447I MSG GEN ENTERED: STMT# 00004 OF DECK INSTMTXT
-----------------------------------------------------------------------------------------------------------------------------------------
                                                 12040288 0096056  11000000  CNSL  0800 000000     58
  ITP137I INSTALL1 WSIMLU  -00001 - NOW LOGGED ON TO ITPECHO
-----------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040288 0096056  11000000  MTRC  0100 083020     53   E2   INSTMTXT  00        19
  ITP448I MSG GEN ENDED:   STMT# 00009 OF DECK INSTMTXT
```

```
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12040288 0096056  11000000  DSPY  0200 080040     24   E2   INSTMTXT  00         2
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|                                                                                | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|THIS IS A SIMPLE MESSAGE                                                        |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN( 25)   AID: ENTER KEY                  WHEN LOGGED: END OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040363 12040398 12040288  XMIT  8000 880000     39   E2   INSTMTXT  00        21
   XMIT (DATA) REQUEST
       TH   2C0001010001          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   0390A0   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   BEGIN BRACKET
                     INDICATORS=        CHANGE DIRECTION
       RU   7DD1F811 D160E3C8 C9E240C9 E240C140    E2C9D4D7 D3C540D4 C5E2E2C1 C7C5       *'J8.J-THIS IS A SIMPLE MESSAGE  *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040403 12040403 12040403  RECV  8000 080000     52   E2   INSTMTXT  00        22
   RECV (DATA) REQUEST
       TH   2C0001010002          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=2
       RH   039040   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   END BRACKET
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40E2C9D4   *1C.J-..  .+"..  .  THIS IS A SIM*
 00000020   D7D3C540 D4C5E2E2 C1C7C5                                                    *PLE MESSAGE                     *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040403 0096056  11000000  MTRC  0100 083000     89   E2   INSTMTXT  00        23
  ITP427I INPUT     IF  1   (INSTMTXT 00006) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12040906 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         3
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A SIMPLE MESSAGE                                                        | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040906 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        25
  ITP447I MSG GEN ENTERED: STMT# 00009 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00013 OF DECK INSTMTXT

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12040906 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         4
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A SIMPLE MESSAGE                                                        | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|THIS IS A 50 CHARACTER MESSAGE                                                  |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN( 51)   AID: ENTER KEY                  WHEN LOGGED: END OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040913 12040921 12040906  XMIT  8000 880000     65   E2   INSTMTXT  00        28
   XMIT (DATA) REQUEST
       TH   2C0001010002          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=2
       RH   0390A0   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   BEGIN BRACKET
                     INDICATORS=        CHANGE DIRECTION
       RU   7DD2D211 D160E3C8 C9E240C9 E240C140    F5F040C3 C8C1D9C1 C3E3C5D9 40D4C5E2   *'KK.J-THIS IS A 50 CHARACTER MES*
 00000020   E2C1C7C5 40404040 40404040 40404040    40404040 40404040                     *SAGE                             *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040921 12040921 12040921  RECV  8000 080000     78   E2   INSTMTXT  00        29
   RECV (DATA) REQUEST
       TH   2C0001010003          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=3
       RH   039040   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   END BRACKET
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40F5F040   *1C.J-..  .+"..  .  THIS IS A 50 *
 00000020   C3C8C1D9 C1C3E3C5 D940D4C5 E2E2C1C7    C5404040 40404040 40404040 40404040   *CHARACTER MESSAGE                *
 00000040   40404040 40                                                                  *                                 *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12040921 0096056  11000000  MTRC  0100 083000     89   E2   INSTMTXT  00        30
  ITP427I INPUT     IF  2   (INSTMTXT 00010) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12041423 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         5
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A 50 CHARACTER MESSAGE                                                  | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041423 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        32
  ITP447I MSG GEN ENTERED: STMT# 00013 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00017 OF DECK INSTMTXT

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12041423 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         6
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A 50 CHARACTER MESSAGE                                                  | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|THIS IS A 200 CHARACTER MESSAGE                                                 |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
```

```
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 17) COLUMN( 41)   AID: ENTER KEY                  WHEN LOGGED: END OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041424 12041432 12041423  XMIT  8000 880000    215   E2   INSTMTXT  00        35
   XMIT (DATA) REQUEST
       TH   2C0001010003          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=3
       RH   0390A0   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   BEGIN BRACKET
                     INDICATORS=        CHANGE DIRECTION
       RU   7DD4E811 D160E3C8 C9E240C9 E240C140    F2F0F040 C3C8C1D9 C1C3E3C5 D940D4C5   *'MY.J-THIS IS A 200 CHARACTER ME*
 00000020   E2E2C1C7 C5404040 40404040 40404040    40404040 40404040 40404040 40404040   *SSAGE                           *
 00000040   40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040   *                                *
        TO NEXT LINE SAME AS ABOVE
 000000C0   40404040 40404040 40404040 4040                                              *                                *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041432 12041433 12041433  RECV  8000 080000    228   E2   INSTMTXT  00        36
   RECV (DATA) REQUEST
       TH   2C0001010004          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=4
       RH   039040   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   END BRACKET
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40F2F0F0   *1C.J-..  .+"..  .  THIS IS A 200*
 00000020   40C3C8C1 D9C1C3E3 C5D940D4 C5E2E2C1    C7C54040 40404040 40404040 40404040   * CHARACTER MESSAGE              *
 00000040   40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040   *                                *
        TO NEXT LINE SAME AS ABOVE
 000000C0   40404040 40404040 40404040 40404040    40404040 40404040 404040              *                                *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041433 0096056  11000000  MTRC  0100 083000     89   E2   INSTMTXT  00        37
  ITP427I INPUT     IF  3   (INSTMTXT 00014) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1                                         12041921 0096056  11000000  MARK  1000 000000      0

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12041941 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         7
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A 200 CHARACTER MESSAGE                                                 | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT   |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041941 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        39
  ITP447I MSG GEN ENTERED: STMT# 00017 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00021 OF DECK INSTMTXT

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12041941 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         8
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|THIS IS A 200 CHARACTER MESSAGE                                                 | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT   |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|10                                                                              |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  3)   AID: PROGRAM FUNCTION 5         WHEN LOGGED: END OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041955 12041973 12041941  XMIT  8000 880000     17   E2   INSTMTXT  00        42
   XMIT (DATA) REQUEST
       TH   2C0001010004          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=4
       RH   0390A0   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   BEGIN BRACKET
                     INDICATORS=        CHANGE DIRECTION
       RU   F5D1E211 D160F1F0                                                    *5JS.J-10                      *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041973 12041973 12041973  RECV  8000 080000     38   E2   INSTMTXT  00        43
   RECV (DATA) REQUEST
       TH   2C0001010005          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=5
       RH   039040   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP   END BRACKET
       RU   F1C311D1 60131140 403C4E7F 00114040    124040C1 C2C3C4C5 C6C7C8C9 D1         *1C.J-..  .+"..  .  ABCDEFGHIJ   *
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12041973 0096056  11000000  MTRC  0100 083000     89   E2   INSTMTXT  00        44
  ITP427I INPUT     IF  4   (INSTMTXT 00018) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
INSTALL1        WSIMAPPL          WSIMLU-1       12042475 0096056  11000000  DSPY  0200 080060   2236   E2   INSTMTXT  00         9
            1         2         3         4         5         6         7         8
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
   --------------------------------------------------------------------------------
 1|ABCDEFGHIJ                                                                      | 1
 2|                                                                                | 2
 3|                                                                                | 3
 4|                                                                                | 4
 5|                                                                                | 5
 6|                                                                                | 6
 7|                                                                                | 7
 8|                                                                                | 8
 9|                                                                                | 9
10|                                                                                |10
11|                                                                                |11
12|                                                                                |12
13|WELCOME TO ITPECHO.  ENTER=ECHO   CLEAR=RESTORE   5/6=STRING REPEAT   9=REPEAT  |13
14|ENTER DATA TO ECHO BELOW:                                                       |14
15|                                                                                |15
16|                                                                                |16
17|                                                                                |17
18|                                                                                |18
19|                                                                                |19
20|                                                                                |20
21|                                                                                |21
22|                                                                                |22
23|                                                                                |23
24|                                                                                |24
   --------------------------------------------------------------------------------
   12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
      CURSOR: ROW( 15) COLUMN(  1)   AID: NULL DISPLAY AID           WHEN LOGGED: BEGINNING OF MSG GEN
      DIMENSIONS: ( 24, 80)
------------------------------------------------------------------------------------------------------------------------------------
INSTALL1        WSIMAPPL          WSIMLU-1       12042475 0096056  11000000  MTRC  0100 083060     53   E2   INSTMTXT  00        46
  ITP447I MSG GEN ENTERED: STMT# 00021 OF DECK INSTMTXT
  ITP448I MSG GEN ENDED:   STMT# 00025 OF DECK INSTMTXT
```

WSIM application loglist

Figure 40 contains a portion of a loglist used to run the sample WSim application, described in “WSim as an application” on page 289.

Figure 40. WSim Application loglist output

```
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
                                                 15422703 0096058  11000000  CNSL  0800 000000     75
  Workload Simulator (WSim) VERSION 1, RELEASE 1.0
------------------------------------------------------------------------------------------------------------------------------------
                                                 15424097 0096058  11000000  CNSL  0800 000000     10
  I ECHO,S,L
------------------------------------------------------------------------------------------------------------------------------------
                                                 15425737 0096058  11000000  CNSL  0800 000000     48
  ITP029I INITIALIZATION COMPLETE FOR NETWORK ECHO
------------------------------------------------------------------------------------------------------------------------------------
                                                 15425941 0096058  11000000  CNSL  0800 000000     28
  ITP006I NETWORK ECHO STARTED
------------------------------------------------------------------------------------------------------------------------------------
ECHO                                             15432704 0096058  11000000  MARK  1000 000000      0
------------------------------------------------------------------------------------------------------------------------------------
ECHO                                             15442705 0096058  11000000  MARK  1000 000000      0
------------------------------------------------------------------------------------------------------------------------------------
ECHO            VA1               PLU-1          15444522 15444522 15444522  RECV  8000 080000    227   E0   PLUECHO   00         0
   RECV CONTROL INITIATE REQUEST
       TH   2D0001000000          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=00   ODAI=0   SEQUENCE=0
       RH   0B8000   REQUEST    FM DATA-FM HEADER     ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   81060100 0A000000 0000002A 010303B1    90308000 0087C780 00020000 00000018   *A....................GG.........*
 00000020   5000007E 000008E6 E2C9D4C1 D7D7D300    05000212 1020F305 E6E2C9D4 F1000000   *&;.=...WSIMAPPL.......3.WSIM1...*
 00000040   000E01C0 6D000000 80000018 5000007E    000D24C4 F4C1F3F2 F7F8F240 40404040   *...._.......&;.=...D4A32782     *
 00000060   40404013 00010800 00010002 00030004    00050006 00070015 14000000 01000A00   *   .............................*
 00000080   00000101 04D5C5E3 C1404040 400E0EF3    D5C5E3C1 E6E2C9D4 D4C1D7D7 D30E0BF3   *.....NETA    ..3NETA.WSIMAPPL..3*
 000000A0   D5C5E3C1 E6E2C9D4 D4F12C0A 01084040    40404040 40402D09 08C4F4C1 F3F2F7F8   *NETA.WSIM1....        ...D4A3278*
 000000C0   F26012DF 279FE4D0 3AB51409 D5C5E3C1    4BC1F0F1 D42F0303 8000                *2-....U.....NETA.A01M.....      *
------------------------------------------------------------------------------------------------------------------------------------
```

```
ECHO           VA1              PLU-1         15444566 15444566 15444522 +XMIT  8000 880020     14   E0   PLUECHO   00         1
   XMIT CONTROL INITIATE RESPONSE
       TH   2D0000010000          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=00   OAF=01   ODAI=0   SEQUENCE=0
       RH   8B8000   RESPONSE   FM DATA-FM HEADER     ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   810601FE 00                                                          *A....                           *
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444567 15444840 15444522  XMIT  8000 880000     53   E0   PLUECHO   00         2
   XMIT BIND SESSION REQUEST
       TH   2D0001010001          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   6B8000   REQUEST     SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   31010303 B1903080 000087C7 80000200    00000000 18500000 7E000008 E6E2C9D4   *..........GG.........&;.=...WSIM*
 00000020   C1D7D7D3 00050002 12102000                                           *APPL........                    *
        BIND SESSION   FORMAT 0   TYPE=NON-NEGOTIABLE   FM PROFILE 3     TS PROFILE 3
           PRIMARY PROTOCOLS:   RU CHAINING=MULTIPLE   REQUEST MODE=IMMEDIATE   RESPONSE REQUESTED=DEF OR EXC   END BRACKET SENT
           SECONDARY PROTOCOLS: RU CHAINING=MULTIPLE   REQUEST MODE=IMMEDIATE   RESPONSE REQUESTED=EXCEPTION    END BRACKET NOT SENT
           COMMON PROTOCOLS:    SEGMENTS SUPPORTED     FM HEADERS NOT ALLOWED   BRACKETS RESET BETB      BRACKET TERMINATION RULE 1
                                ALTERNATE CODE SET NOT USED   HALF-DUPLEX FLIP-FLOP    RECOVERY RESPONSIBILITY=PRIMARY
                                CONTENTION WINNER=SECONDARY
           SECONDARY SEND PACING COUNT=NONE         SECONDARY RECEIVE PACING COUNT=NONE   ADAPTIVE SESSION PACING NOT SUPPORTED

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME          TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
           SECONDARY MAXIMUM RU SEND SIZE=1024      PRIMARY MAXIMUM RU SEND SIZE=1536
           PRIMARY SEND PACING COUNT=NONE           PRIMARY RECEIVE PACING COUNT=NONE
           LU TYPE 2   DEFAULT SCREEN SIZE=024,080  ALTERNATE SCREEN SIZE=NONE
           PRIMARY LU NAME=WSIMAPPL                 CRYPTOGRAPHIC FIELD=NONE
           URC=0002121020
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444840 15444840 15444840  RECV  8000 080000     10   E0   PLUECHO   00         3
   RECV BIND SESSION RESPONSE
       TH   2D0001010001          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   EB8000   RESPONSE    SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   31                                                                   *.                               *
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444848 15444861 15444840  XMIT  8000 880000     10   E0   PLUECHO   00         4
   XMIT START DATA TRAFFIC REQUEST
       TH   2D000101008E          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=142
       RH   6B8000   REQUEST     SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   A0                                                                   *.                               *
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444861 15444861 15444861  RECV  8000 080000     10   E0   PLUECHO   00         5
   RECV START DATA TRAFFIC RESPONSE
       TH   2D000101008E          FID=2   WHOLE SEGMENT  EXPEDITED FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=142
       RH   EB8000   RESPONSE    SESSION CONTROL      ONLY IN CHAIN   RESPONSE TYPE=DEF1
       RU   A0                                                                   *.                               *
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444861 0096058  11000000  MTRC  0100 083060     53   E0   PLUECHO   00         6
  ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00001) NOT MET - ELSE ACTION NOT CODED
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444861 0096058  11000000  INFO  4000 082000     67   E0   PLUECHO   00        18
  ITP410I DATASAVE LENGTH ERROR, DATA MAY BE TRUNCATED IN SAVEAREA N1
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444861 0096058  11000000  MTRC  0100 083020    130   E0   PLUECHO   00        19
  ITP427I IMMEDIATE IF      (PLUECHO  00006) MET - THEN ACTION TAKEN: BRANCH FROM 00007 OF PLUECHO  TO BUILDN1  AT 00004 OF PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00006) NOT MET - ELSE ACTION NOT CODED
  ITP451I CALL FROM STMT# 00008 OF DECK PLUECHO  TO  THE  BEGINNING   OF DECK FIRSTMSG
  ITP448I MSG GEN ENDED:   STMT# 00001 OF DECK FIRSTMSG
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444931 0096058  11000000  MTRC  0100 083060     53   E0   FIRSTMSG  00        23
  ITP447I MSG GEN ENTERED: STMT# 00001 OF DECK FIRSTMSG
  ITP452I RETURN FROM STMT# 00002 OF DECK FIRSTMSG TO STMT# 00009 OF DECK PLUECHO

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME          TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
ECHO           VA1              PLU-1         15444931 0096058  11000000  INFO  4000 082000     42   E0   PLUECHO   00        25
  ITP407I RECALL SAVEAREA 1 CONTAINS NO DATA
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444931 0096058  11000000  MTRC  0100 083020     53   E0   PLUECHO   00        26
  ITP448I MSG GEN ENDED:   STMT# 00014 OF DECK PLUECHO
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444938 15444938 15444931  XMIT  8000 880000    130   E0   PLUECHO   00        27
   XMIT (DATA) REQUEST
       TH   2C0001010001          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   0380A0   REQUEST        FM DATA          ONLY IN CHAIN   RESPONSE TYPE=DEF1   BEGIN BRACKET
                  INDICATORS=        CHANGE DIRECTION
       RU   F5C7114E 7F1DF8E6 C5D3C3D6 D4C540E3    D640C9E3 D7C5C3C8 D64B1D60 40C5D5E3   *5G.+".8WELCOME TO ITPECHO..- ENT*
 00000020   C5D97EC5 C3C8D640 4040C3D3 C5C1D97E    D9C5E2E3 D6D9C540 4040F57E E2E3D9C9   *ER=ECHO   CLEAR=RESTORE   5=STRI*
 00000040   D5C740D9 C5D7E340 4040F97E D9C5D7C5    C1E31150 50C5D5E3 C5D940C4 C1E3C140   *NG REPT   9=REPEAT.&&ENTER DATA *
 00000060   E3D640C5 C3C8D640 C2C5D3D6 E67A11D1    5F1D4013 115D7F1D F0                  *TO ECHO BELOW:.J–. ..)".0       *
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444956 15444956 15444956  RECV  8000 080000      9   E0   PLUECHO   00        28
   RECV RESPONSE
       TH   2C0001010001          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   838000   RESPONSE       FM DATA          ONLY IN CHAIN   RESPONSE TYPE=DEF1
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15444956 0096058  11000000  MTRC  0100 083000     88   E0   PLUECHO   00        29
  ITP423I INPUT     IF  0   (PLUECHO  00009) NOT EVALUATED - START OF TEST NOT WITHIN DATA
  ITP423I INPUT     IF  1   (PLUECHO  00010) NOT EVALUATED - START OF TEST NOT WITHIN DATA
------------------------------------------------------------------------------------------------------------------------------------
ECHO           VA1              PLU-1         15445369 15445370 15445370  RECV  8000 080000     39   E0   PLUECHO   00        31
   RECV (DATA) REQUEST
       TH   2C0001010001          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=1
       RH   039020   REQUEST        FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                  INDICATORS=        CHANGE DIRECTION
       RU   7DD1F811 D160E3C8 C9E240C9 E240C140    E2C9D4D7 D3C540D4 C5E2E2C1 C7C5       *'J8.J-THIS IS A SIMPLE MESSAGE  *
------------------------------------------------------------------------------------------------------------------------------------
```

```
ECHO          VA1             PLU-1         15445370 0096058  11000000  MTRC  0100 083000    108   E0   PLUECHO   00       32
  ITP427I INPUT     IF  0   (PLUECHO  00009) MET - THEN ACTION TAKEN: EXECUTED SAVERU   FROM  THE  BEGINNING
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445370 0096058  11000000  INFO  4000 082000     42   E0   SAVERU    00       33
  ITP407I RECALL SAVEAREA 1 CONTAINS NO DATA
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445370 0096058  11000000  MTRC  0100 083000     84   E0   PLUECHO   00       34
  ITP432I INPUT     IF  0   (PLUECHO  00009) EXECUTE ACTION ENDED AT 00004 OF SAVERU
  ITP427I INPUT     IF  1   (PLUECHO  00010) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445371 0096058  11000000  MTRC  0100 083060     53   E0   PLUECHO   00       36
  ITP447I MSG GEN ENTERED: STMT# 00014 OF DECK PLUECHO

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME          NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
  ITP429I IMMEDIATE IF      (PLUECHO  00014) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00015) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00016 OF DECK PLUECHO  TO NOTPF9   AT 00018 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00018) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00019) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00020) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00021) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00022) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00024 OF DECK PLUECHO  TO ECHOLOOP AT 00011 OF DECK PLUECHO
  ITP448I MSG GEN ENDED:   STMT# 00014 OF DECK PLUECHO
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445378 15445378 15444938  XMIT  8000 880000     52   E0   PLUECHO   00       47
   XMIT (DATA) REQUEST
       TH   2C0001010002          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=2
       RH   039020   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                INDICATORS=        CHANGE DIRECTION
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40E2C9D4   *1C.J-..  .+"..  .  THIS IS A SIM*
 00000020   D7D3C540 D4C5E2E2 C1C7C5                                                 *PLE MESSAGE                     *
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445900 15445900 15445900  RECV  8000 080000     65   E0   PLUECHO   00       48
   RECV (DATA) REQUEST
       TH   2C0001010002          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=2
       RH   039020   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                INDICATORS=        CHANGE DIRECTION
       RU   7DD2D211 D160E3C8 C9E240C9 E240C140    F5F040C3 C8C1D9C1 C3E3C5D9 40D4C5E2   *'KK.J-THIS IS A 50 CHARACTER MES*
 00000020   E2C1C7C5 40404040 40404040 40404040    40404040 40404040                     *SAGE                            *
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445900 0096058  11000000  MTRC  0100 083000    108   E0   PLUECHO   00       49
  ITP427I INPUT     IF  0   (PLUECHO  00009) MET - THEN ACTION TAKEN: EXECUTED SAVERU   FROM  THE  BEGINNING
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445900 0096058  11000000  INFO  4000 082000     42   E0   SAVERU    00       50
  ITP407I RECALL SAVEAREA 1 CONTAINS NO DATA
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445900 0096058  11000000  MTRC  0100 083000     84   E0   PLUECHO   00       51
  ITP432I INPUT     IF  0   (PLUECHO  00009) EXECUTE ACTION ENDED AT 00004 OF SAVERU
  ITP427I INPUT     IF  1   (PLUECHO  00010) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15445900 0096058  11000000  MTRC  0100 083060     53   E0   PLUECHO   00       53
  ITP447I MSG GEN ENTERED: STMT# 00014 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00014) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00015) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00016 OF DECK PLUECHO  TO NOTPF9   AT 00018 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00018) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00019) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00020) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00021) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00022) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00024 OF DECK PLUECHO  TO ECHOLOOP AT 00011 OF DECK PLUECHO
  ITP448I MSG GEN ENDED:   STMT# 00014 OF DECK PLUECHO
--------------------------------------------------------------------------------------------------------------------------

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME          NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
ECHO          VA1             PLU-1         15445902 15445907 15445378  XMIT  8000 880000     78   E0   PLUECHO   00       64
   XMIT (DATA) REQUEST
       TH   2C0001010003          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=3
       RH   039020   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                INDICATORS=        CHANGE DIRECTION
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40F5F040   *1C.J-..  .+"..  .  THIS IS A 50 *
 00000020   C3C8C1D9 C1C3E3C5 D940D4C5 E2E2C1C7    C5404040 40404040 40404040 40404040   *CHARACTER MESSAGE               *
 00000040   40404040 40                                                              *                                *
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15450418 15450419 15450419  RECV  8000 080000    215   E0   PLUECHO   00       65
   RECV (DATA) REQUEST
       TH   2C0001010003          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=3
       RH   039020   REQUEST         FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                INDICATORS=        CHANGE DIRECTION
       RU   7DD4E811 D160E3C8 C9E240C9 E240C140    F2F0F040 C3C8C1D9 C1C3E3C5 D940D4C5   *'MY.J-THIS IS A 200 CHARACTER ME*
 00000020   E2E2C1C7 C5404040 40404040 40404040    40404040 40404040 40404040 40404040   *SSAGE                           *
 00000040   40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040   *                                *
          TO NEXT LINE SAME AS ABOVE
 000000C0   40404040 40404040 40404040 4040                                          *                                *
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15450419 0096058  11000000  MTRC  0100 083000    108   E0   PLUECHO   00       66
  ITP427I INPUT     IF  0   (PLUECHO  00009) MET - THEN ACTION TAKEN: EXECUTED SAVERU   FROM  THE  BEGINNING
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15450419 0096058  11000000  INFO  4000 082000     42   E0   SAVERU    00       67
  ITP407I RECALL SAVEAREA 1 CONTAINS NO DATA
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15450419 0096058  11000000  MTRC  0100 083000     84   E0   PLUECHO   00       68
  ITP432I INPUT     IF  0   (PLUECHO  00009) EXECUTE ACTION ENDED AT 00004 OF SAVERU
  ITP427I INPUT     IF  1   (PLUECHO  00010) MET - THEN ACTION TAKEN: CONTINUE (RESET WAIT)
--------------------------------------------------------------------------------------------------------------------------
ECHO          VA1             PLU-1         15450419 0096058  11000000  MTRC  0100 083060     53   E0   PLUECHO   00       70
  ITP447I MSG GEN ENTERED: STMT# 00014 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00014) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00015) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00016 OF DECK PLUECHO  TO NOTPF9   AT 00018 OF DECK PLUECHO
  ITP429I IMMEDIATE IF      (PLUECHO  00018) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00019) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00020) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00021) NOT MET - ELSE ACTION NOT CODED
  ITP429I IMMEDIATE IF      (PLUECHO  00022) NOT MET - ELSE ACTION NOT CODED
  ITP450I BRANCH FROM STMT# 00024 OF DECK PLUECHO  TO ECHOLOOP AT 00011 OF DECK PLUECHO
  ITP448I MSG GEN ENDED:   STMT# 00014 OF DECK PLUECHO
-----------------------------------------------------------------------------------------------------------------------------------
ECHO            VA1              PLU-1          15450420 15450420 15445906  XMIT  8000 880000    228   E0   PLUECHO   00        81
   XMIT (DATA) REQUEST
       TH   2C0001010004          FID=2   WHOLE SEGMENT  NORMAL    FLOW  DAF=01   OAF=01   ODAI=0   SEQUENCE=4
       RH   039020   REQUEST        FM DATA          ONLY IN CHAIN   RESPONSE TYPE=EXCP
                     INDICATORS=        CHANGE DIRECTION
       RU   F1C311D1 60131140 403C4E7F 00114040    124040E3 C8C9E240 C9E240C1 40F2F0F0   *1C.J-..  .+"..  .  THIS IS A 200*
 00000020   40C3C8C1 D9C1C3E3 C5D940D4 C5E2E2C1    C7C54040 40404040 40404040 40404040   * CHARACTER MESSAGE              *
 00000040   40404040 40404040 40404040 40404040    40404040 40404040 40404040 40404040   *                                *
         TO NEXT LINE SAME AS ABOVE
```

CPI-C multiple-instance TP loglist

Figure 41 contains the complete loglist for the CPI-C example with multiple-instance transaction programs, described in “CPI-C example with multiple-instance transaction programs” on page 314.

Figure 41. CPI-C multiple-instance TP loglist output

```
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
                                                12175906 0096141  11000000  CNSL  0800 000000     70
  Workload Simulator (WSim) VERSION 1, RELEASE 1.0
-----------------------------------------------------------------------------------------------------------------------------------
                                                12175910 0096141  11000000  CNSL  0800 000000     52
  ITP029I INITIALIZATION COMPLETE FOR NETWORK CPICSMP2
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-1     12175911 0096141  11000000  CTRC  2000 043000     53   EA           00         0
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPCLIENT-00001
-----------------------------------------------------------------------------------------------------------------------------------
                                                12175911 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00001 - Transaction Program TPCLIENT starting.
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-1     12175912 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00         1
CPI-C VERB ISSUED:               Conversation State = Reset
  CMINIT( conversation_ID ,                                                   ( S1 )                <character output>
          sym_dest_name ,                'SERVER'                             ( S4 )                <character input >
          return_code )                                                       ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMINIT( conversation_ID ,              '00000001'                           ( S1 )                <character output>
          sym_dest_name ,                'SERVER'                             ( S4 )                <character input >
          return_code )                  CM_OK                                ( DC15 = 0 )          <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMSSL ( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          sync_level ,                   CM_CONFIRM                           ( DC23 = 1 )          <numeric   input >
          return_code )                                                       ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMSSL ( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          sync_level ,                   CM_CONFIRM                           ( DC23 = 1 )          <numeric   input >
          return_code )                  CM_OK                                ( DC15 = 0 )          <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMALLC( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          return_code )                                                       ( DC15 )              <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-2     12175913 0096141  11000000  CTRC  2000 043000     53   EA           00         6
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPCLIENT-00002
-----------------------------------------------------------------------------------------------------------------------------------
                                                12175913 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00002 - Transaction Program TPCLIENT starting.
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-2     12175913 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00         7
CPI-C VERB ISSUED:               Conversation State = Reset
  CMINIT( conversation_ID ,                                                   ( S1 )                <character output>
          sym_dest_name ,                'SERVER'                             ( S4 )                <character input >
          return_code )                                                       ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMINIT( conversation_ID ,              '00000001'                           ( S1 )                <character output>
          sym_dest_name ,                'SERVER'                             ( S4 )                <character input >
          return_code )                  CM_OK                                ( DC15 = 0 )          <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMSSL ( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          sync_level ,                   CM_CONFIRM                           ( DC23 = 1 )          <numeric   input >
          return_code )                                                       ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMSSL ( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          sync_level ,                   CM_CONFIRM                           ( DC23 = 1 )          <numeric   input >
          return_code )                  CM_OK                                ( DC15 = 0 )          <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMALLC( conversation_ID ,              '00000001'                           ( S1 )                <character input >
          return_code )                                                       ( DC15 )              <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
                                                12175913 0096141  11000000  CNSL  0800 000000     32
  ITP006I NETWORK CPICSMP2 STARTED
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-3     12175914 0096141  11000000  CTRC  2000 043000     53   EA           00         6
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPCLIENT-00003
-----------------------------------------------------------------------------------------------------------------------------------
                                                12175914 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00003 - Transaction Program TPCLIENT starting.
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-3     12175914 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00         7
```

```
CPI-C VERB ISSUED:               Conversation State = Reset
  CMINIT( conversation_ID ,                                          ( S1 )               <character output>
          sym_dest_name ,                  'SERVER'                  ( S4 )               <character input >
          return_code )                                              ( DC15 )             <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMINIT( conversation_ID ,                '00000001'                ( S1 )               <character output>
          sym_dest_name ,                  'SERVER'                  ( S4 )               <character input >
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMSSL ( conversation_ID ,                '00000001'                ( S1 )               <character input >
          sync_level ,                     CM_CONFIRM                ( DC23 = 1 )         <numeric   input >
          return_code )                                              ( DC15 )             <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Initialize
  CMSSL ( conversation_ID ,                '00000001'                ( S1 )               <character input >
          sync_level ,                     CM_CONFIRM                ( DC23 = 1 )         <numeric   input >
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
CPI-C VERB ISSUED:               Conversation State = Initialize
  CMALLC( conversation_ID ,                '00000001'                ( S1 )               <character input >
          return_code )                                              ( DC15 )             <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-1     12175918 0096141  11000000  CTRC  2000 141000    413   EA   CLIENT    00         6
CPI-C VERB COMPLETED:            Conversation State = Send
  CMALLC( conversation_ID ,                '00000001'                ( S1 )               <character input >
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION           deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                 fill            = CM_FILL_LL
  mode_name               = WSIMLU62                         partner_LU_name = KWSAPC2
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL    receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED        send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                         'E3D7E2C5D9E5C5D9'X
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-1     12175912 12175918 12175912  XMIT  8000 941000     18   EA   CLIENT    00         6
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                    *......J. .TPSERVER            *
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-1     12175918 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00         7
CPI-C VERB ISSUED:               Conversation State = Send
  CMSEND( conversation_ID ,                '00000001'                ( S1 )               <character input >
          send_buffer ,                    *See Data at VERB Completion*      ( S7 )      <character input >
          send_length ,                    63                        ( DC17 = 63 )        <numeric   input >
          request_to_send_received ,                                 ( DC13 )             <numeric   output>
          return_code )                                              ( DC15 )             <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Send
  CMSEND( conversation_ID ,                '00000001'                ( S1 )               <character input >
          send_buffer ,                    *See XMIT Record Data*    ( S7 )               <character input >
          send_length ,                    63                        ( DC17 = 63 )        <numeric   input >
          request_to_send_received ,       CM_REQ_TO_SEND_NOT_RECEIVED        ( DC13 = 0 )         <numeric   output>
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
CPICSMP2       APPCLUC          TPCLIENT-1     12175918 12175918 12175912  XMIT  8000 841000     63   EA   CLIENT    00         8
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F17A40C4   *LU APPCLUC, TP TPCLIENT-00001: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-1     12175918 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00         9
CPI-C VERB ISSUED:               Conversation State = Send
  CMDEAL( conversation_ID ,                '00000001'                ( S1 )               <character input >
          return_code )                                              ( DC15 )             <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-2     12175919 0096141  11000000  CTRC  2000 141000    413   EA   CLIENT    00        12
CPI-C VERB COMPLETED:            Conversation State = Send
  CMALLC( conversation_ID ,                '00000001'                ( S1 )               <character input >
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION           deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                 fill            = CM_FILL_LL
  mode_name               = WSIMLU62                         partner_LU_name = KWSAPC2
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL    receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED        send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                         'E3D7E2C5D9E5C5D9'X
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-2     12175913 12175919 12175913  XMIT  8000 941000     18   EA   CLIENT    00        12
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                    *......J. .TPSERVER            *
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-2     12175919 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        13
CPI-C VERB ISSUED:               Conversation State = Send
  CMSEND( conversation_ID ,                '00000001'                ( S1 )               <character input >
          send_buffer ,                    *See Data at VERB Completion*      ( S7 )      <character input >
          send_length ,                    63                        ( DC17 = 63 )        <numeric   input >
          request_to_send_received ,                                 ( DC13 )             <numeric   output>
          return_code )                                              ( DC15 )             <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Send
  CMSEND( conversation_ID ,                '00000001'                ( S1 )               <character input >
          send_buffer ,                    *See XMIT Record Data*    ( S7 )               <character input >
          send_length ,                    63                        ( DC17 = 63 )        <numeric   input >
          request_to_send_received ,       CM_REQ_TO_SEND_NOT_RECEIVED        ( DC13 = 0 )         <numeric   output>
          return_code )                    CM_OK                     ( DC15 = 0 )         <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-2     12175919 12175919 12175913  XMIT  8000 841000     63   EA   CLIENT    00        14
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F27A40C4   *LU APPCLUC, TP TPCLIENT-00002: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUC          TPCLIENT-2     12175919 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        15
CPI-C VERB ISSUED:               Conversation State = Send
  CMDEAL( conversation_ID ,                '00000001'                ( S1 )               <character input >
          return_code )                                              ( DC15 )             <numeric   output>
-----------------------------------------------------------------------------------------------------------------------------------
CPICSMP2       APPCLUS          TPSERVER-1     12175919 0096141  11000000  CTRC  2000 043000     53   EA             00         0
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPSERVER-00001
-----------------------------------------------------------------------------------------------------------------------------------
                                               12175919 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00001 - Transaction Program TPSERVER starting.
-----------------------------------------------------------------------------------------------------------------------------------
```

```
CPICSMP2        APPCLUS           TPSERVER-1     12175919 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        1
CPI-C VERB ISSUED:                Conversation State = Reset
  CMACCP( conversation_ID ,                                                     ( S1 )                <character output>
          return_code )                                                         ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:             Conversation State = Receive
  CMACCP( conversation_ID ,                  '00000001'                         ( S1 )                <character output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION            deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                  fill            = CM_FILL_LL
  mode_name               = WSIMLU62                          partner_LU_name = KWSAPC1
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL     receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED         send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                          'E3D7E2C5D9E5C5D9'X
------------------------------------------------------------------------------------------------------------------------------------
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
CPICSMP2        APPCLUS           TPSERVER-1     12175919 12175919 12175919  RECV  8000 141000     18   EA   SERVER    00        2
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                 *......J. .TPSERVER              *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-1     12175919 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        3
CPI-C VERB ISSUED:                Conversation State = Receive
  CMRCV ( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          receive_buffer ,                                                      ( S8 )                <character output>
          requested_length ,                 100                                ( DC23 = 100 )        <numeric   input >
          data_received ,                                                       ( DC3 )               <numeric   output>
          received_length ,                                                     ( DC12 )              <numeric   output>
          status_received ,                                                     ( DC19 )              <numeric   output>
          request_to_send_received ,                                            ( DC13 )              <numeric   output>
          return_code )                                                         ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:             Conversation State = Confirm-Deallocate
  CMRCV ( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          receive_buffer ,                   *See RECV Record Data*             ( S8 )                <character output>
          requested_length ,                 100                                ( DC23 = 100 )        <numeric   input >
          data_received ,                    CM_COMPLETE_DATA_RECEIVED          ( DC3 = 2 )           <numeric   output>
          received_length ,                  63                                 ( DC12 = 63 )         <numeric   output>
          status_received ,                  CM_CONFIRM_DEALLOC_RECEIVED        ( DC19 = 4 )          <numeric   output>
          request_to_send_received ,         CM_REQ_TO_SEND_NOT_RECEIVED        ( DC13 = 0 )          <numeric   output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-1     12175919 12175919 12175919  RECV  8000 041000     63   EA   SERVER    00        4
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F17A40C4   *LU APPCLUC, TP TPCLIENT-00001: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-1     12175919 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        5
CPI-C VERB ISSUED:                Conversation State = Confirm-Deallocate
  CMCFMD( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                                                         ( DC15 )              <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-1     12175920 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00       10
CPI-C VERB COMPLETED:             Conversation State = Reset
  CMDEAL( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175920 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00001 - Transaction Program TPCLIENT complete.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-1     12175920 0096141  11000000  CTRC  2000 043000     60   EA             00       11
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPCLIENT-00001
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-3     12175920 0096141  11000000  CTRC  2000 141000    413   EA   CLIENT    00       12
CPI-C VERB COMPLETED:             Conversation State = Send
  CMALLC( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION            deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                  fill            = CM_FILL_LL
  mode_name               = WSIMLU62                          partner_LU_name = KWSAPC2
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL     receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED         send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                          'E3D7E2C5D9E5C5D9'X
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-3     12175914 12175920 12175914  XMIT  8000 941000     18   EA   CLIENT    00       12
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                 *......J. .TPSERVER              *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-3     12175920 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00       13
CPI-C VERB ISSUED:                Conversation State = Send
  CMSEND( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          send_buffer ,                      *See Data at VERB Completion*      ( S7 )                <character input >
          send_length ,                      63                                 ( DC17 = 63 )         <numeric   input >
          request_to_send_received ,                                            ( DC13 )              <numeric   output>
          return_code )                                                         ( DC15 )              <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-1     12175920 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        6
CPI-C VERB COMPLETED:             Conversation State = Reset
  CMCFMD( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175920 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00001 - Transaction Program TPSERVER complete.
------------------------------------------------------------------------------------------------------------------------------------
NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
CPICSMP2        APPCLUS           TPSERVER-1     12175920 0096141  11000000  CTRC  2000 043000     60   EA             00        7
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPSERVER-00001
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175920 0096141  11000000  CTRC  2000 043000     53   EA             00        8
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPSERVER-00002
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175920 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00002 - Transaction Program TPSERVER starting.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175920 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        9
```

```
CPI-C VERB ISSUED:                Conversation State = Reset
  CMACCP( conversation_ID ,                                                     ( S1 )                <character output>
          return_code )                                                         ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:             Conversation State = Receive
  CMACCP( conversation_ID ,                  '00000001'                         ( S1 )                <character output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION            deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                  fill            = CM_FILL_LL
  mode_name               = WSIMLU62                          partner_LU_name = KWSAPC1
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL     receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED         send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                          'E3D7E2C5D9E5C5D9'X
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175919 12175920 12175920  RECV  8000 141000     18   EA   SERVER    00        10
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                      *......J. .TPSERVER             *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175920 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        11
CPI-C VERB ISSUED:                Conversation State = Receive
  CMRCV ( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          receive_buffer ,                                                      ( S8 )                <character output>
          requested_length ,                 100                                ( DC23 = 100 )        <numeric   input >
          data_received ,                                                       ( DC3 )               <numeric   output>
          received_length ,                                                     ( DC12 )              <numeric   output>
          status_received ,                                                     ( DC19 )              <numeric   output>
          request_to_send_received ,                                            ( DC13 )              <numeric   output>
          return_code )                                                         ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:             Conversation State = Confirm-Deallocate
  CMRCV ( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          *seive_buffer ,                    *See RECV Record Data*             ( S8 )                <character output>
          requested_length ,                 100                                ( DC23 = 100 )        <numeric   input >
          data_received ,                    CM_COMPLETE_DATA_RECEIVED          ( DC3 = 2 )           <numeric   output>
          received_length ,                  63                                 ( DC12 = 63 )         <numeric   output>
          status_received ,                  CM_CONFIRM_DEALLOC_RECEIVED        ( DC19 = 4 )          <numeric   output>
          request_to_send_received ,         CM_REQ_TO_SEND_NOT_RECEIVED        ( DC13 = 0 )          <numeric   output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175919 12175920 12175920  RECV  8000 041000     63   EA   SERVER    00        12
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F27A40C4   *LU APPCLUC, TP TPCLIENT-00002: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
------------------------------------------------------------------------------------------------------------------------------------

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME              NAME           TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
CPICSMP2        APPCLUC           TPCLIENT-3     12175920 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        14
CPI-C VERB COMPLETED:             Conversation State = Send
  CMSEND( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          send_buffer ,                      *See XMIT Record Data*             ( S7 )                <character input >
          send_length ,                      63                                 ( DC17 = 63 )         <numeric   input >
          request_to_send_received ,         CM_REQ_TO_SEND_NOT_RECEIVED        ( DC13 = 0 )          <numeric   output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-3     12175920 12175920 12175914  XMIT  8000 841000     63   EA   CLIENT    00        14
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F37A40C4   *LU APPCLUC, TP TPCLIENT-00003: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-3     12175920 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        15
CPI-C VERB ISSUED:                Conversation State = Send
  CMDEAL( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                                                         ( DC15 )              <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175920 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        13
CPI-C VERB ISSUED:                Conversation State = Confirm-Deallocate
  CMCFMD( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                                                         ( DC15 )              <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-2     12175921 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        16
CPI-C VERB COMPLETED:             Conversation State = Reset
  CMDEAL( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00002 - Transaction Program TPCLIENT complete.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC           TPCLIENT-2     12175921 0096141  11000000  CTRC  2000 043000     60   EA             00        17
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPCLIENT-00002
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175921 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        14
CPI-C VERB COMPLETED:             Conversation State = Reset
  CMCFMD( conversation_ID ,                  '00000001'                         ( S1 )                <character input >
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00002 - Transaction Program TPSERVER complete.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-2     12175921 0096141  11000000  CTRC  2000 043000     60   EA             00        15
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPSERVER-00002
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-3     12175921 0096141  11000000  CTRC  2000 043000     53   EA             00         8
  ITP4020I EXECUTION BEGINS FOR CPI-C TP TPSERVER-00003
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00003 - Transaction Program TPSERVER starting.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS           TPSERVER-3     12175921 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00         9
CPI-C VERB ISSUED:                Conversation State = Reset
  CMACCP( conversation_ID ,                                                     ( S1 )                <character output>
          return_code )                                                         ( DC15 )              <numeric   output>
CPI-C VERB COMPLETED:             Conversation State = Receive
  CMACCP( conversation_ID ,                  '00000001'                         ( S1 )                <character output>
          return_code )                      CM_OK                              ( DC15 = 0 )          <numeric   output>
Conversation Characteristics:
  conversation_type       = CM_MAPPED_CONVERSATION            deallocate_type = CM_DEALLOCATE_SYNC_LEVEL
  error_direction         = CM_RECEIVE_ERROR                  fill            = CM_FILL_LL
  mode_name               = WSIMLU62                          partner_LU_name = KWSAPC1
  prepare_to_receive_type = CM_PREP_TO_RECEIVE_SYNC_LEVEL     receive_type    = CM_RECEIVE_AND_WAIT
  return_control          = CM_WHEN_SESSION_ALLOCATED         send_type       = CM_BUFFER_DATA
  sync_level              = CM_CONFIRM
  TP_name                 = TPSERVER                          'E3D7E2C5D9E5C5D9'X
------------------------------------------------------------------------------------------------------------------------------------

NETWORK   APPCLU/TCPIP/VTAMAPPL  DEV/LU/TP       START    STOP     READY  RECORD   HEADER      DATA  TERM  MESSAGE   USER SEQUENCE
 NAME           NAME             NAME            TIME     TIME     TIME    TYPE    FLAGS       LENG  TYPE    DECK    DATA  NUMBER
CPICSMP2        APPCLUS          TPSERVER-3      12175921 12175921 12175921  RECV  8000 141000     18   EA   SERVER    00        10
    FMH-5   120502FF 0003D100 4008E3D7 E2C5D9E5    C5D9                                 *......J. .TPSERVER              *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS          TPSERVER-3      12175921 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        11
CPI-C VERB ISSUED:               Conversation State = Receive
  CMRCV ( conversation_ID ,                 '00000001'                        ( S1 )               <character input >
          receive_buffer ,                                                    ( S8 )               <character output>
          requested_length ,                100                               ( DC23 = 100 )       <numeric   input >
          data_received ,                                                     ( DC3 )              <numeric   output>
          received_length ,                                                   ( DC12 )             <numeric   output>
          status_received ,                                                   ( DC19 )             <numeric   output>
          request_to_send_received ,                                          ( DC13 )             <numeric   output>
          return_code )                                                       ( DC15 )             <numeric   output>
CPI-C VERB COMPLETED:            Conversation State = Confirm-Deallocate
  CMRCV ( conversation_ID ,                 '00000001'                        ( S1 )               <character input >
          receive_buffer ,                *See RECV Record Data*              ( S8 )               <character output>
          requested_length ,                100                               ( DC23 = 100 )       <numeric   input >
          data_received ,                   CM_COMPLETE_DATA_RECEIVED         ( DC3 = 2 )          <numeric   output>
          received_length ,                 63                                ( DC12 = 63 )        <numeric   output>
          status_received ,                 CM_CONFIRM_DEALLOC_RECEIVED       ( DC19 = 4 )         <numeric   output>
          request_to_send_received ,        CM_REQ_TO_SEND_NOT_RECEIVED       ( DC13 = 0 )         <numeric   output>
          return_code )                     CM_OK                             ( DC15 = 0 )         <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS          TPSERVER-3      12175921 12175921 12175921  RECV  8000 041000     63   EA   SERVER    00        12
 00000000   D3E440C1 D7D7C3D3 E4C36B40 E3D740E3    D7C3D3C9 C5D5E360 F0F0F0F0 F37A40C4   *LU APPCLUC, TP TPCLIENT-00003: D*
 00000020   81A38140 A28595A3 40869996 94408393    898595A3 40A39640 A28599A5 85994B     *ata sent from client to server. *
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS          TPSERVER-3      12175921 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        13
CPI-C VERB ISSUED:               Conversation State = Confirm-Deallocate
  CMCFMD( conversation_ID ,                 '00000001'                        ( S1 )               <character input >
          return_code )                                                       ( DC15 )             <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-3      12175921 0096141  11000000  CTRC  2000 141000    253   EA   CLIENT    00        16
CPI-C VERB COMPLETED:            Conversation State = Reset
  CMDEAL( conversation_ID ,                 '00000001'                        ( S1 )               <character input >
          return_code )                     CM_OK                             ( DC15 = 0 )         <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPCLIENT-00003 - Transaction Program TPCLIENT complete.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUC          TPCLIENT-3      12175921 0096141  11000000  CTRC  2000 043000     60   EA             00        17
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPCLIENT-00003
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS          TPSERVER-3      12175921 0096141  11000000  CTRC  2000 141000    253   EA   SERVER    00        14
CPI-C VERB COMPLETED:            Conversation State = Reset
  CMCFMD( conversation_ID ,                 '00000001'                        ( S1 )               <character input >
          return_code )                     CM_OK                             ( DC15 = 0 )         <numeric   output>
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     72
  ITP137I CPICSMP2 TPSERVER-00003 - Transaction Program TPSERVER complete.
------------------------------------------------------------------------------------------------------------------------------------
                                                 12175921 0096141  11000000  CNSL  0800 000000     54
  ITP137I CPICSMP2 TPSERVER-00003 - Simulation complete.
------------------------------------------------------------------------------------------------------------------------------------
CPICSMP2        APPCLUS          TPSERVER-3      12175921 0096141  11000000  CTRC  2000 043000     60   EA             00        15
  ITP4021I EXECUTION HAS COMPLETED FOR CPI-C TP TPSERVER-00003
------------------------------------------------------------------------------------------------------------------------------------
                                                 12180837 0096141  11000000  CNSL  0800 000000      4
```

Chapter 27. Network models

This topic contains copies of the model networks and scripts that are accessible online when you use the WSim/ISPF Interface. You can use these models as prototypes when you code WSim networks and scripts. Both STL procedures and message generation decks are provided. You might need to change some values in these models to operate in your environment. These values are indicated by the "==>" string in the model. A comment following the string indicates the type of change required.

VTAM application simulation

This WSim script simulates two VTAM applications, each having one LU. The name used for this model by the WSim/ISPF Interface is VTAMAPPL.

Network definition

```
/* VTAM application simulation                                              */
@NETWORK
********************************************************************************
* Network Configuration:  VTAM application (VTAMAPPL)                          *
*                                                                              *
* Description:  This WSim script will simulate two VTAM applications (APPL1    *
*               and APPL2), each having one LU.                                *
*                                                                              *
*               Some values may need to be changed in this data set in         *
*               order to operate in your environment.  They are indicated      *
*               by the "==> " string.                                          *
*                                                                              *
* Restrictions/Dependencies:                                                   *
*  1) Start the WSim echo application ITPECHO with the VTAM application name   *
*     ITPECHO.                                                                 *
*  2) The VTAMAPPL names used in this network (APPL1 and APPL2) must be        *
*     defined to VTAM and must be active.                                      *
*                                                                              *
********************************************************************************


*------------------------------------------------------------------------------*
* Network statement operands.                                                  *
*------------------------------------------------------------------------------*
VAPPL    NTWRK    ITIME=1,          * Network interval report every 1 minute  *
                  SCAN=(1,1,0),     * Scan/display/recovery times             *
                  UTI=100,          * User time interval = 1 second           *


*------------------------------------------------------------------------------*
* VTAMAPPL operands coded on the network statement.  These values will         *
* be the default for every VTAMAPPL in the network.                            *
*------------------------------------------------------------------------------*
                  BUFSIZE=3000,     * Buffer size is 3000 bytes               *
                  MLOG=YES,         * Message logging function will be used   *


*------------------------------------------------------------------------------*
* LU operands coded on the network statement.  These values will be            *
* the default for every Logical Unit in the network.                           *
*------------------------------------------------------------------------------*
                  DELAY=F1,         * Use fixed message delay interval        *
                  DLOGMOD=D4A32782, * VTAM logon mode                         *
                  INIT=SEC,          * Secondary LU initiates the session      *
                  LOGDSPLY=BOTH,     * Log the display buffers before and after*
*                                    * message generation                      *
                  LUTYPE=LU2,        * Logical unit type = LU2                 *
                  STLTRACE=YES,      * Write message generation trace          *
*                                    * records to the log                      *
                  PATH=(0),          * Specifies which PATH statement the LU   *
*                                    * will use                                *
                  THKTIME=UNLOCK     * Terminal unlock starts msg delay timer  *


*-----------------------------------------------------------------------------*
0        PATH     LUDECK             * Run the LUDECK msgtxt on this path      *
*-----------------------------------------------------------------------------*


%EJECT
*-----------------------------------------------------------------------------*
* Define the network resources.                                               *
*                                                                             *
* ==> CHANGE the VTAMAPPL names APPL1 and APPL2 as needed to match names      *
*     in your environment.  These names must be defined to VTAM.              *
*-----------------------------------------------------------------------------*
APPL1    VTAMAPPL
LU11     LU
*LU12     LU
*
APPL2    VTAMAPPL
LU21     LU
*LU22     LU
@ENDNETWORK
```

Message generation deck

```
LUDECK   MSGTXT
*******************************************************************************
* The Message Generation deck.                                                *
*******************************************************************************


*******************************************************************************
* Issue an INITSELF to log on to ITPECHO.                                     *
* Wait until the logon is complete.                                           *
* After the logon completes, issue a message to the console.                  *
*******************************************************************************
         CMND COMMAND=INIT,RESOURCE=ITPECHO
0        IF LOC=RU+0,TEXT=(WELCOME),SCAN=YES,THEN=CONT,ELSE=WAIT
         WAIT
         WTO ($LUID$ UP AND RUNNING)

*******************************************************************************
* Send a message to ITPECHO.                                                  *
* Repeat the loop forever until the WSim operator stops the network.          *
*******************************************************************************
LOOP     LABEL
         TEXT (MSG $DSEQ,5$ FROM $LUID$ )
         BRANCH LABEL=LOOP
         ENDTXT
```

STL procedure

```
@PROGRAM=VTAMAPPL
/******************************************************************************
* The variable dictionary.                                                    *
******************************************************************************/
                                     /*****************************************/
bit      unshared  stay              /* Indicates that we haven't gotten the  */
```

```
                                     /* logon screen yet                     */
integer  unshared  counter           /* Counts how many messages have been    */
                                     /* sent between the LUs                  */
                                     /****************************************/


LUDECK:   msgtxt
/********************************************************************************
* The STL deck.                                                                 *
********************************************************************************/


/********************************************************************************
* Issue an INITSELF to log on to ITPECHO.                                       *
* Wait until the logon is complete.                                             *
* After the logon completes, issue a message to the console.                    *
********************************************************************************/
check: onin index(screen,'WELCOME') > 0 then stay = off
stay = on
initself('ITPECHO','D4A32782')
do while stay = on
   wait until onin
end
deact check
say luid() 'up and running'


/********************************************************************************
* Send a message to ITPECHO.                                                    *
* Repeat the loop forever until the WSim operator stops the network.            *
********************************************************************************/
counter = 0
do forever
   counter = counter + 1
   type char(counter) ' FROM ' luid()
   transmit
   end
endtxt
```

Telnet 3270 simulation

This WSim script simulates a TCP/IP connection with 4 simulated Telnet 3270 clients. The name used for this model by the WSim/ISPF Interface is TN3270.

Network definition

```
* Telnet 3270 simulation                                              *
***********************************************************************
* Network Configuration:  Telnet 3270 simulation                      *
*                                                                     *
* Description:  This WSim script will simulate four 3270 devices      *
*               connecting to an application logon screen and logging *
*               back off.  The SERVADDR operand specifies the IP dotted*
*               address of the host to which the devices 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.                       *
*                                                                     *
***********************************************************************


*---------------------------------------------------------------------*
* Network statement operands.                                         *
*---------------------------------------------------------------------*
TN3270   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                     *
*--------------------------------------------------------------------*
SOMEHOST PATH  SOMEHOST              * Execute SOMEHOST msgtxt
*--------------------------------------------------------------------*
* Define the network resources.                                      *
*                                                                    *
* This is a TCP/IP connection with 4 simulated devices.  You may     *
* add additional operands on the devices if desired.  See the WSim   *
* Script Guide and Reference for details on valid operands.          *
*--------------------------------------------------------------------*
TCONN1   TCPIP
DEV11    DEV
DEV12    DEV
DEV13    DEV
DEV14    DEV
```

Message generation deck

```
*--------------------------------------------------------------------*
SOMEHOST MSGTXT
*--------------------------------------------------------------------*
* The Message Generation deck.                                       *
*                                                                    *
* 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 attempted 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, WSim is closed down.                                     *
*                                                                    *
*--------------------------------------------------------------------*
         CALL   NAME=WAITSCRN
         WTO    ($DEVID$ ESTABLISHED TCPIP SESSION, LOGGING ON)
         SET    NC1=NSEQ
         SET    NSEQ=+1
         TEXT   ($UTBL,IDS,NC1$),MORE=YES
         TAB
         TEXT   ($UTBL,PWS,NC1$)
         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
*------------------------------------------------------------------------
*                                    * <== 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

```
/*----------------------------------------------------------------------*
* The Message Generation deck.                                          *
*                                                                       *
* 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, WSim     *
* is closed down.                                                       *
*                                                                       *
*----------------------------------------------------------------------*/
allocate nextnum 'NSEQ'
integer  shared nextid


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


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


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

File Transfer Protocol (FTP) simulation

This WSim script simulates a TCP/IP connection with one simulated TCP/IP FTP client. The name used for this model by the WSim/ISPF Interface is FTP.

Network definition

```
* File Transfer Protocol                                            *
*********************************************************************
* Network Configuration:  File Transfer Protocol simulation         *
*                                                                   *
* Description:  This WSim script will simulate one FTP client user  *
*               connecting to a server, putting a file to that server, *
*               then retrieving the same file.  The SERVADDR operand *
*               specifies the IP dotted address of the host to which *
*               the device 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.                     *
*                                                                   *
*********************************************************************


*-------------------------------------------------------------------*
* Network statement operands.                                       *
*-------------------------------------------------------------------*
FTP      NTWRK HEAD='Model FTP Network', * Set the title line
               CONRATE=YES,        * Print message rates on console
               OPTIONS=(DEBUG,MONCMND), * 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
               TCPNAME=TCPIP,      * <== Default name of the local
*                                  *     TCPIP virtual machine
               SERVADDR=9.67.43.62  * <== Remote Server
*                                  *     to which you will connect
*-------------------------------------------------------------------*
1         UTBL (Hes the invisible man), * First Record Data
               (Catch him if you can)   * Second Record Data
*-------------------------------------------------------------------*
FTPDECK  PATH  FTPDECK
```

```
*--------------------------------------------------------------------*
* Define the file data to be simulated                               *
*--------------------------------------------------------------------*
FILE1    FILE  TYPE=E,             * File Data is EBCDIC
               RECFM=F,            * Fixed Record Format
               RECLEN=80,          * Record Length is 80
               DATA=1              * Get data from UTBL 1
*--------------------------------------------------------------------*
* 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 WSim    *
* Script guide and Reference for details on valid operands.          *
*--------------------------------------------------------------------*
TCONN1   TCPIP  PATH=(FTPDECK)
DEV010   DEV  TYPE=FTP
```

Message generation deck

```
FTPDECK  MSGTXT
*--------------------------------------------------------------------*
* The Message Generation deck.                                       *
*                                                                    *
* Generates FTP commands for file transfer.                          *
* user     - identifies the user to the server.                      *
* pass     - supplies a password to the server.                      *
* ebcdic   - set file transfer type to EBCDIC.                       *
* mode     - specifies file transfer mode.                           *
* sendsite - disables automatic transmission of SITE subcommand.     *
* put      - transfers the FILE data defined by WSim FILE statement.  *
* get      - transfers the data from a file on the server.           *
*                                                                    *
*--------------------------------------------------------------------*
*  Wait for connect
1        IF    WHEN=IN,LOC=D+0,COND=GE,
               TEXT=('00'x),THEN=CONT     /* Wait for next message    */
         WAIT
LOOP     TEXT  (user testuser)            /* <== Enter User ID        */
         TEXT  (pass testpass)            /* <== Enter Password       */
         TEXT  (ebcdic)                   /* EBCDIC Translation       */
         TEXT  (mode B)                   /* Block Mode               */
         TEXT  (sendsite)                 /* Automatic SITE Off       */
         TEXT  (put FILE1 TEST.FILE1)     /* Put File1                */
         TEXT  (get TEST.FILE1)           /* Get File1 back           */
4        IF    WHEN=IN,
               LOCTEXT=($RECALL,B+0,3$),COND=EQ,TEXT=(250),THEN=E-SUCC
         BRANCH LABEL=NOTSU
SUCC     WTO   (Get successful.)
         RETURN
NOTSU    LABEL
5        IF    WHEN=IN,
               LOCTEXT=($RECALL,B+0,3$),COND=EQ,TEXT=(426),THEN=E-FAIL
         BRANCH LABEL=QUIT
FAIL     WTO   (Get failed.)
         RETURN
QUIT     TEXT  (quit),MORE=YES
* Following quiesce will keep automatic reconnect from
* occurring.  Release the DEV to recycle through the
* script.  Remove the QUIESCE to
* automatically recycle after 30 seconds.
         QUIESCE
         BRANCH  LABEL=LOOP
         ENDTXT
```

STL procedure

```
ftpdeck:  msgtxt
/*------------------------------------------------------------------*
* The Message Generation deck.                                       *
*                                                                    *
* Generates FTP commands for file transfer.                          *
* user     - identifies the user to the server.                      *
* pass     - supplies a password to the server.                      *
* ebcdic   - set file transfer type to EBCDIC.                       *
* mode     - specifies file transfer mode.                           *
* sendsite - disables automatic transmission of SITE subcommand.     *
* put      - transfers the FILE data defined by WSim FILE statement.  *
* get      - transfers the data from a file on the server.           *
*                                                                    *
*-------------------------------------------------------------------*/
/*  Wait for connect                                                */
  wait until onin
 Do forever
  type 'user testuser'                   /* <== Enter User ID        */
  transmit
  type 'pass  testpass'                  /* <== Enter Password       */
  transmit
  type 'ebcdic'                          /* EBCDIC Transfer Type     */
  transmit
  type 'mode B'                          /* Block Mode               */
  transmit
  type 'sendsite'                        /* Automatic SITE Off       */
  transmit
  type 'put FILE1 TEST.FILE1'            /* Put File1                */
  transmit
  s: onin substr(buffer,1,3)='250' then   /* Check message number     */
    say 'Get successful.'
  f: onin substr(buffer,1,3)='426' then   /* Check message number     */
    say 'Get failed.'
  type 'get TEST.FILE1'                  /* Get File1 back           */
  transmit
  deact s,f
  type 'quit'                            /* Drop Connection          */
/* Following quiesce will keep automatic reconnect from */
/* occurring.  Release the DEV to recycle through the   */
/* script.  Replace the quiesce with a transmit to      */
/* automatically recycle after 30 seconds.              */
  quiesce
 end
 endtxt
```

Simple TCP client simulation

This WSim script simulates a TCP/IP connection with one simulated TCP/IP Simple TCP Client. The name used for this model by the WSim/ISPF Interface is STCP.

Network definition

```
* Simple TCP Client
*********************************************************************
* Network Configuration:  Simple TCP Client simulation              *
*                                                                   *
* Description:  This WSim 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 a sequence of    *
*                   two carriage return/line feed (CR/LF) sequences;   *
*                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.                        *
*                                                                      *
************************************************************************


*----------------------------------------------------------------------*
* Network statement operands.                                          *
*----------------------------------------------------------------------*
STCP     NTWRK HEAD='Model STCP Network', * Set the title line
               CONRATE=YES,         * Print message rates on console
               OPTIONS=(MONCMND),   * 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
               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 WSim      *
* Script guide and Reference for details on valid operands.            *
*----------------------------------------------------------------------*
TCONN1   TCPIP
DEV010   DEV  TYPE=STCP,           * Simple TCP Client
              PORT=5555,           * Server Port for connection
              SERVADDR=9.67.43.62, * Server IP Address for connection
              PATH=(STCPDECK)      * Path Sequence for this DEV
```

Message generation deck

```
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 ASCII
         DATASAVE AREA=1,
                  TEXT=('000102031A091A7F1A1A1A0B0C0D0E0F')+  * 00-0F
                       ('101112131A1A081A18191A1A1C1D1E1F')+  * 10-1F
                       ('1A1A1C1A1A0A171B1A1A1A1A1A050607')+  * 20-2F
                       ('1A1A161A1A1E1A041A1A1A1A14151A1A')+  * 30-3F
                       ('20A6E180EB909FE2AB8B9B2E3C282B7C')+  * 40-4F
                       ('26A9AA9CDBA599E3A89E21242A293B5E')+  * 50-5F
                       ('2D2FDFDC9ADDDE989DACBA2C255F3E3F')+  * 60-6F
                       ('D78894B0B1B2FCD6FB603A2340273D22')+  * 70-7F
                       ('F861626364656667686996A4F3AFAEC5')+  * 80-8F
                       ('8C6A6B6C6D6E6F7071729787CE93F1FE')+  * 90-9F
                       ('C87E737475767778797AEFC0DA5BF2F9')+  * A0-AF
                       ('B5B6FDB7B8B9E6BBBCBD8DD9BF5DD8C4')+  * B0-BF
                       ('7B414243444546474849CBCABEE8ECED')+  * C0-CF
                       ('7D4A4B4C4D4E4F505152A1ADF5F4A38F')+  * D0-DF
                       ('5CE7535455565758595AA0858EE9E4D1')+  * E0-EF
                       ('30313233343536373839B3F7F0FAA7FF')   * F0-FF
* Set length of input data each time data is received
1        IF       WHEN=IN,STATUS=HOLD,SNASCOPE=ALL,
                  LOC=NC1,COND=GE,TEXT=0,
                  THEN=E-SAVELENG
         BRANCH   LABEL=TRANLOOP
SAVELENG DATASAVE AREA=2,
                  TEXT=($RECALL,B+0$)
         SET      DC2=LENG(2)
         RETURN
* Loop sending transaction
TRANLOOP DATASAVE AREA=2,
                  FUNCTION=TRANSLATE,
                  TEXT=(Sample Transaction from $ID,8$),
                  TABLEO=($RECALL,1$)
         DATASAVE AREA=3,
                  FUNCTION=TRANSLATE,
                  TEXT=(Sample Transaction line 2),
                  TABLEO=($RECALL,1$)
         TEXT     ($RECALL,2$'0D0A'$RECALL,3$'0D0A0D0A')
2        IF       WHEN=IN,STATUS=HOLD,SNASCOPE=ALL,
                  LOC=DC2,TEXT=0,COND=EQ,
                  THEN=CONT
         WAIT
         DEACT    IFS=(2)
         BRANCH   LABEL=TRANLOOP
         ENDTXT
```

STL procedure

```
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 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 */
/* Set length of input data each time data is received */
  onin then ilen=length(buffer)
/* Loop sending transaction                            */
 Do forever
  type translate('Sample Transaction from' id(),ebc2asc)||'0d0a'x||,
       translate('Sample Transaction line 2',ebc2asc)||'0d0a0d0a'x;
  transmit and wait until onin ilen=0     /* Send in the transaction  */
                                          /* and wait for connection  */
                                          /* to close                 */
 end
 endtxt
```

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 WSim scripting.

Simple TCP Client connecting to a server using Telnet Line Mode Network Virtual Terminal

```
@NET
*************************************************************************
* Network Configuration:  Simple TCP Client simulation                  *
*                                                                       *
* Description:  This WSim 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.                         *
*                                                                       *
*************************************************************************

*-----------------------------------------------------------------------*
* 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 WSim      *
* Script Guide and Reference 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
```

CPI-C transaction program simulation

This following WSim script simulates a CPI-C client transaction program communicating with two CPI-C server transaction programs. The name used for this model by the WSim/ISPF Interface is CPIC.

Network definition

```
* CPI-C Transaction Program simulation
***********************************************************************
* Network Configuration:  CPI-C Transaction Program simulation (CPIC)  *
*                                                                     *
* Description:  This WSim script will simulate a CPI-C client         *
*               Transaction Program communicating with two CPI-C      *
*               server Transaction Programs.  The client sends        *
*               data to one server and receives data from the other.  *
*               The sync-level is "none" on the first conversation,   *
*               and "confirm" on the second conversation.             *
*                                                                     *
*               To illustrate that a network-wide Side Information    *
*               Table can be overridden at the APPCLU level, the      *
*               network-wide table contains an entry that points      *
*               to a non-existent TP.  This entry is then overridden  *
*               by the APPCLU statement.                              *
*                                                                     *
*               Some values may need to be changed in this data set in *
*               order to operate in your environment.  They are       *
*               indicated by the "==> " string.                       *
*                                                                     *
* Restrictions/Dependencies:                                          *
*  1) The APPLID names used in this network (APPLID1 and APPLID2)     *
*     must be defined to VTAM and must be active.                     *
*                                                                     *
* Graphical Representation of Network:                                *
*                                                                     *
*   APPCLU: APPCLU1                          APPCLU: APPCLU2          *
*   +----------------------+                 +---------------------+  *
*   |                      |                 |                     |  *
*   |   +--------------+   |  conversation 1 |   +-------------+   |  *
*   |   |              ==========================> TP: TPSERV1 |   |  *
*   |   |              |   |   mode=#inter   |   +-------------+   |  *
*   |   | TP: TPCLIENT |   |                 |                     |  *
*   |   |              |   |  conversation 2 |   +-------------+   |  *
*   |   |              ==========================> TP: TPSERV1 |   |  *
*   |   +--------------+   |   mode=#batch   |   +-------------+   |  *
*   |                      |                 |                     |  *
*   +----------------------+                 +---------------------+  *
*                                                                     *
* Notes:                                                              *
*  1. Conversation 1 uses mode name #INTER.  The conversation         *
*     sync-level is "none".                                           *
*  2. Conversation 2 uses mode name #BATCH.  The conversation         *
*     sync-level is "confirm".                                        *
*  3. The CNOS operand on the APPCLU1 definition is only required     *
*     if you want to control the number of sessions.  If the operand  *
*     is not specified, sessions will be managed by WSim as required  *
*     by the simulation.                                              *
*                                                                     *
***********************************************************************


*---------------------------------------------------------------------*
* Network statement operands.                                         *
*---------------------------------------------------------------------*
CPIC     NTWRK    HEAD='CPI-C NETWORK MODEL',
                  ITIME=1,          * Ntwrk interval rpt every minute *
```

```
*--------------------------------------------------------------------*
* TP operands coded on the network statement.  These values will      *
* be the default for every TP in the network.                         *
*--------------------------------------------------------------------*
                  TPSTATS=YES,       * Keep stats for all TP instances *
                  CPITRACE=VERB      * Trace CPI-C verbs in the log    *

*--------------------------------------------------------------------*
* Define a network-wide Side Information Table.                       *
*--------------------------------------------------------------------*
         SIDEINFO
         SIDEENT  DESTNAME=SERVER1,MODENAME=#INTER,
                  LUNAME=APPLID2,TPNAME=TPSERVER
         SIDEENT  DESTNAME=SERVER2,MODENAME=#BATCH,
                  LUNAME=APPLID2,TPNAME=TPSERV2
         SIDEEND

*--------------------------------------------------------------------*
* Define the Transaction Program paths.                               *
*--------------------------------------------------------------------*
CLIENT   PATH     CLNTDCK            * Define the CLIENT TP path       *
SERVER1  PATH     SERV1DCK           * Define the SERVER1 TP path      *
SERVER2  PATH     SERV2DCK           * Define the SERVER2 TP path      *
*--------------------------------------------------------------------*


*--------------------------------------------------------------------*
* Define the network resources.                                       *
*                                                                     *
* ==> CHANGE the APPLID names APPLID1 and APPLID2 as needed to match   *
*     names in your environment.  If you change APPLID2, also change   *
*     the LUNAME specification in the SIDEINFO and CNOS operands to    *
*     match this name.  These names must be defined to VTAM.           *
*--------------------------------------------------------------------*
APPCLU1  APPCLU APPLID=APPLID1,      * APPC LU; VTAM APPLID is APPLID1 *
         SIDEINFO=((DESTNAME=SERVER1,MODENAME=#INTER,
                    LUNAME=APPLID2,TPNAME=TPSERV1)),
*                                    * Override SERVER1 dest name      *
         CNOS=((LUNAME=APPLID2,MODENAME=#INTER,
                SESSIONS=2,CWL=1,CWP=1))
*                                    * Specify CNOS values             *
TPC      TP TPNAME=TPCLIENT,         * TP name is TPCLIENT             *
            PATH=(CLIENT),           * TP is defined by CLIENT path    *
            TPTYPE=CLIENT,           * TP type is CLIENT               *
            INSTANCE=(1,1)           * 1 initial TP instance           *

APPCLU2  APPCLU APPLID=APPLID2       * APPC LU; VTAM APPLID is APPLID2 *

TPS1     TP TPNAME=TPSERV1,          * TP name is TPSERV1              *
            PATH=(SERVER1),          * TP is defined by SERVER1 path   *
            TPTYPE=SERVER,           * TP type is SERVER               *
            INSTANCE=(0,1)           * No initial TP instances         *

TPS2     TP TPNAME=TPSERV2,          * TP name is TPSERV2              *
            PATH=(SERVER2),          * TP is defined by SERVER2 path   *
            TPTYPE=SERVER,           * TP type is SERVER               *
            INSTANCE=(0,1)           * No initial TP instances         *
```

Message generation decks

```
CLNTDCK  MSGTXT
**********************************************************************
* Message deck defining the TPCLIENT Transaction Program.             *
**********************************************************************
*
* Device save area usage:
```

```
*  1=conversation ID
*  2=destination name
*  3=send buffer
*  4=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=send length
*  dc3=request-to-send received
*  dc4=sync-level
*  dc5=requested length
*  dc6=data received
*  dc7=received length
*  dc8=status received
*
         WTO  (Transaction Program $TPID$ starting.)
*
************************************************************************
*        Initialize and allocate a conversation with TPSERV1.          *
*
*
*        Set the symbolic destination name to "SERVER1".
         DATASAVE AREA=2,TEXT=(SERVER1)
*
*        Initialize the conversation.
         CMINIT(1,2,DC1)
*
*        Allocate the conversation; the sync-level defaults to "none",
*        and the conversation type defaults to "mapped".
*
         CMALLC(1,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=3,TEXT=(LU $APPCLUID$, TP $TPID$$TPINSTNO$:)+
                  ( Data sent from client to server.)    * Send buffer
         SET  DC2=LENG(3)                                * Send length
*
*        Send data to TPSERV1.
         CMSEND(1,3,DC2,DC3,DC1)
*
*        Deallocate the conversation with TPSERV1.
         CMDEAL(1,DC1)
*
************************************************************************
*        Initialize and allocate a conversation with TPSERV2.          *
*
*        Set the symbolic destination name to "SERVER2".
         DATASAVE AREA=2,TEXT=(SERVER2)
*
*        Initialize the conversation.
         CMINIT(1,2,DC1)
*
*        Set the conversation sync-level to "confirm".
         SET  DC4=1                                      * Sync-level
         CMSSL(1,DC4,DC1)
*
*        Allocate the conversation; the conversation type defaults
*        to "mapped".
         CMALLC(1,DC1)
*
*        Set requested length for receive.
         SET  DC5=100
*
*        Receive data from TPSERV2.
         CMRCV(1,4,DC5,DC6,DC7,DC8,DC3,DC1)
*
*        Confirm the data was received.
```

```
         CMCFMD(1,DC1)
*
*        Receive the confirm deallocate status.
         CMRCV(1,4,DC5,DC6,DC7,DC8,DC3,DC1)
*
*        Confirm the deallocate
         CMCFMD(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
         WTO  (Simulation complete.)
*
         ENDTXT

SERV1DCK MSGTXT
************************************************************************
* Message deck defining the TPSERV1 Transaction Program.               *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPCLIENT.
         CMACCP(1,DC1)
*
*        Set requested length for receive.
         SET  DC2=100
*
*        Receive data from TPCLIENT.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT

SERV2DCK MSGTXT
************************************************************************
* Message deck defining the TPSERV2 Transaction Program.               *
************************************************************************
*
* Device save area usage:
*  1=conversation ID
*  2=receive buffer
*  3=send buffer
*
* Device counter usage:
*  dc1=return code
*  dc2=requested length
*  dc3=data received
*  dc4=received length
*  dc5=status received
*  dc6=request-to-send received
*  dc7=send length
*
         WTO  (Transaction Program $TPID$ starting.)
*
*        Accept the conversation with TPCLIENT.
```

```
         CMACCP(1,DC1)
*
*        Set requested length for receive.
         SET  DC2=100
*
*        Receive send status from TPCLIENT.
         CMRCV(1,2,DC2,DC3,DC4,DC5,DC6,DC1)
*
*        Setup the send buffer and length.
         DATASAVE AREA=3,TEXT=(LU $APPCLUID$, TP $TPID$$TPINSTNO$:)+
                  ( Data sent from server to client.)    * Send buffer
*
         SET      DC7=LENG(3)                            * Send length
*
*        Send data to TPCLIENT.
         CMSEND(1,3,DC7,DC6,DC1)
*
*        Request confirmation that the data was received.
         CMCFM(1,DC6,DC1)
*
*        Deallocate the conversation with TPCLIENT.
         CMDEAL(1,DC1)
*
         WTO  (Transaction Program $TPID$ complete.)
*
         ENDTXT
```

STL procedures

```
@PROGRAM=CPIC

/***********************************************************************
* Include the CPI-C variable and constant definition files.           *
*                                                                     *
* ==> The CPICVAR and CPICCON files must be in your STL includes      *
*     dataset.  They are located in the WSim sample dataset.          *
***********************************************************************/
@include cpicvar
@include cpiccon

CLNTDCK: msgtxt
/***********************************************************************
* STL deck defining the TPCLIENT Transaction Program.                 *
***********************************************************************/

say 'Transaction Program 'tpid() 'starting.'

/*********************************************************************/
/* Initialize and allocate a conversation with TPSERV1.              */

/* Set the symbolic destination name to "SERVER1". */
sym_dest_name='SERVER1'

/* Initialize the conversation. */
CMINIT (conversation_ID, sym_dest_name, return_code)
/* Allocate the conversation; the sync-level defaults to "none", */
/* and the conversation type defaults to "mapped".               */
CMALLC (conversation_ID, return_code)

/* Setup the send buffer and length. */
send_buffer = 'LU' appcluid()', TP' tpid()tpinstno()||,
              ': Data sent from client to server.'
send_length = length(send_buffer)

/* Send data to TPSERV1. */
CMSEND (conversation_ID, send_buffer, send_length,,
        request_to_send_received, return_code)
```

```
/* Deallocate the conversation with TPSERV1. */
CMDEAL (conversation_ID, return_code)

/*********************************************************************/
/* Initialize and allocate a conversation with TPSERV2.              */

/* Set the symbolic destination name to "SERVER2". */
sym_dest_name='SERVER2'

/* Initialize the conversation. */
CMINIT (conversation_ID, sym_dest_name, return_code)

/* Set the conversation sync-level to "confirm". */
CMSSL  (conversation_ID, cm_confirm, return_code)

/* Allocate the conversation; the conversation type defaults to  */
/* "mapped".                                                     */
CMALLC (conversation_ID, return_code)

/* Receive data from TPSERV2. */
CMRCV  (conversation_ID, receive_buffer, 100, data_received,,
        received_length, status_received,,
        request_to_send_received, return_code)

/* Confirm the data was received. */
CMCFMD (conversation_ID, return_code)

/* Receive the confirm deallocate status. */
CMRCV  (conversation_ID, receive_buffer, 100, data_received,,
        received_length, status_received,,
        request_to_send_received, return_code)

/* Confirm the deallocate */
CMCFMD (conversation_ID, return_code)

say 'Transaction Program 'tpid() 'complete.'
say 'Simulation complete.'

endtxt

SERV1DCK: msgtxt
/*********************************************************************
* STL deck defining the TPSERV1 Transaction Program.                 *
*********************************************************************/

say 'Transaction Program 'tpid() 'starting.'

/* Accept the conversation with TPCLIENT. */
CMACCP (conversation_ID,,
        return_code)

/* Receive data from TPCLIENT. */
CMRCV  (conversation_ID,,
        receive_buffer,,
        100,,
        data_received,,
        received_length,,
        status_received,,
        request_to_send_received,,
        return_code)

say 'Transaction Program 'tpid() 'complete.'

endtxt

SERV2DCK: msgtxt
```

```
/**********************************************************************
* STL deck defining the TPSERV2 Transaction Program.                  *
**********************************************************************/

say 'Transaction Program 'tpid() 'starting.'

/* Accept the conversation with TPCLIENT. */
CMACCP (conversation_ID,,
        return_code)

/* Set requested length for receive. */
requested_length=100
/* Receive send status from TPCLIENT. */
CMRCV  (conversation_ID,,
        receive_buffer,,
        requested_length,,
        data_received,,
        received_length,,
        status_received,,
        request_to_send_received,,
        return_code)

/* Setup the send buffer and length. */
send_buffer = 'LU' appcluid()', TP' tpid()tpinstno()||,
              ': Data sent from server to client.'
send_length = length(send_buffer)

/* Send data to TPCLIENT. */
CMSEND (conversation_ID,,
        send_buffer,,
        send_length,,
        request_to_send_received,,
        return_code)

/* Request confirmation that the data was received. */
CMCFM  (conversation_ID,,
        request_to_send_received,,
        return_code)

/* Deallocate the conversation with TPCLIENT. */
CMDEAL (conversation_ID,,
        return_code)

say 'Transaction Program 'tpid() 'complete.'

endtxt
```

Part 6. Appendixes

Glossary

This glossary includes terms and definitions from the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699-6. Further definitions are from the following volumes and reports. The symbols follow the definitions to which they refer.

- The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.
- Definitions from draft proposals and working papers under development by the International Standards Organization, Technical Committee 97, Subcommittee 1 are identified by the symbol (TC97).
- Definitions from draft international standards, draft proposals, and working papers in development by the ISO/TC97/SC1 are identified by the symbol (T), indicating final agreement has not yet been reached among participating members.
- Definitions from the *CCITT Sixth Plenary Assembly Orange Book, Terms and Definitions* and working documents published by the International Consultative Committee on Telegraph and Telephone of the International Telecommunication Union, Geneva, 1980 are identified by the symbol (CCITT/ITU).
- Definitions from published sections of the *ISO Vocabulary of Data Processing*, developed by the International Standards Organization, Technical Committee 97, Subcommittee 1 and from published sections of the *ISO Vocabulary of Office Machines*, developed by subcommittees of ISO Technical Committee 95, are indicated by the symbol (ISO).

A

AID Attention identifier.

American National Standard Code for Information Interchange (ASCII)
The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

API Application program interface.

application program interface (API)
(1) The formally defined programming language interface between an IBM system control program or licensed program and its user. (2) The interface through which an application program interacts with an access method. In VTAM, it is the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

ASCII American National Standard Code for Information Interchange.

attention identifier (AID)
A code that the terminal sends in the inbound data stream to identify the operator action or structured field function that caused the data stream to be sent to the application program. An AID is always sent as the first byte of the inbound data stream. Structured fields in the data stream may also contain an AID.

attribute value
A code immediately following the attribute type in the data stream that specifies a particular characteristic from the set defined by the attribute type.

available
In VTAM, pertaining to a logical unit that is active, connected, enabled, and not at its session limit.

Availability Monitor (AVMON)
A sample set of WSim network definition statements and message generation decks provided with WSim which monitors the availability of host application subsystems.

AVMON
Availability Monitor.

B

bind In SNA, a request to activate a session between two logical units (LUs).

C

carriage return (CR)
The operation that prepares for the next character to be printed or displayed at the specified first position on the same line. (A)

CD Change direction.

chain A group of logically linked records, for example, an SNA message.

character set
(1) A defined collection of characters in a loadable or nonloadable set selected by means of a local character set identifier. (2) An attribute type in the extended field and character attributes. (3) An attribute passed between session partners in the Start Field Extended, Modify Field, and Set Attribute orders.

cluster controller
A device that can control the input/output operations of more than one device connected to it. A cluster controller can be controlled by a program stored and executed in the unit; for example, the IBM 3601 Finance Communication Controller. Or it may be controlled entirely by hardware; for example, the IBM 3272 Control Unit.

Common Programming Interface for Communications (CPI-C)
(1) In WSim, an application programming interface (API) used to perform program-to-program communications using LU type 6.2 communication protocols. (2) An evolving application programming interface (API), embracing functions to meet the growing demands from different application environments and to achieve openness as an industry standard for communications programming. CPI-C provides access to interprogram services such as (a) sending and receiving data, (b) synchronizing processing between programs, and (c) notifying a partner of errors in the communication.

CPI-C Common programming interface for communications.

CR Carriage return.

D

DAF Destination address field.

data flow control (DFC)
In SNA, a request/response unit (RU) category used for requests and responses exchanged between the data flow control layer in one half-session and the data flow control layer in the session partner.

data set
The major unit of data storage and retrieval, consisting of a collection of data in one of several prescribed arrangements and described by control information to which the system has access.

DBCS Double-byte character set.

DDNAME
Data definition name.

DE Device-end.

destination address field (DAF)
In SNA, a field in a FID0 or FID1 transmission header that contains the network address of the destination.

destination logical unit (DLU)
The logical unit to which data is to be sent. Contrast with origin logical unit (OLU).

device-end (DE)
In channel operations, a signal from an I/O device that denotes the end of an operation.

DFC Data flow control.

direct access storage device (DASD)
A device in which the access time is effectively independent of the location of the data. For example, a disk.

Display Monitor Facility
A VTAM application program within WSim that displays simulated 3270 screen images on a monitor. It is used to monitor a simulation dynamically, enabling a user to debug scripts and view interactions with host applications.

DLU Destination logical unit.

domain
(1) An access method, its application programs, communication controllers, connecting lines, modems, and attached

terminals. (2) In SNA, a system services control point (SSCP) and the physical units (PUs), logical units (LUs), links, link stations, and all the associated resources that the SSCP has the ability to control by means of activation requests and deactivation requests.

double-byte character set (DBCS)
A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more symbols that can be represented by 256 code points, require double-byte character sets. Because each character requires two bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS.

E

EB End bracket.

EBCDIC
Extended binary-coded decimal interchange code.

end bracket (EB)
In SNA, the value (binary 1) of the end bracket indicator in the request header (RH) of the first request of the last chain of a bracket; the value denotes the end of the bracket.

end-of-transmission-block character (ETB)
(1) A transmission control character used to indicate the end of a transmission block of data when data are divided into such blocks for transmission purposes. (2) In binary synchronous communication, the transmission control character used to end a block of records that began with the start-of-text character.

ETB End-of-transmission-block character.

event (1) An occurrence of significance to a task; typically, the completion of an asynchronous operation, such as an input/output operation. (2) In WSim, a named indicator/flag which can be used for communications among terminal scripts.

extended attribute buffer (EAB)
The buffer in which the extended field attribute for the 3270 kanji display field is stored.

extended binary-coded decimal interchange code (EBCDIC)
A coded character set of 256 8-bit characters.

extended color
(1) A capability that allows color terminals to display or print fields or characters in colors using extended field and character attributes. (2) An attribute type in the extended field attribute and character attribute.

extended field attribute
Additional field definition to the field attribute that controls defining additional properties such as color, highlighting, character set, and field validation. The extended field attribute is altered by information passed in the Start Field Extended and Modify Field orders.

extended highlighting
(1) A function that provides blink, reverse video, and underscore for emphasizing fields or characters on devices supporting extended field attributes and character attributes. (2) An attribute type in the extended field attribute and character attribute. (3) An attribute passed between session partners in the Start Field Extended, Modify Field, and Set Attribute orders.

F

facility
(1) An operational capability, or the means for providing such a capability. (T) (2) A service provided by an operating system for a particular purpose; for example, the checkpoint/restart facility.

FFW Field format word.

FID SNA format identification.

File Transfer Protocol (FTP)
In the Internet suite of protocols, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

FM Function management.

FMD Function management data.

format identification (FID) field
In SNA, a field in each transmission header (TH) that indicates the format of the TH; that is, the presence or absence of

certain fields. TH formats differ in accordance with the types of nodes between which they pass.

formatted system services (FSS)
A portion of VTAM that provides certain system services as a result of receiving a field-formatted command, such as an Initiate or Terminate command.

frame check sequence (FCS)
A field immediately preceding the closing flag sequence of a frame that contains a bit sequence checked by the receiver to detect transmission errors.

FSS Formatted system services.

FTP File transfer protocol.

function control sequence (FCS)
In System/38 (RJEF) MTAM, a control character used to control the flow of individual function streams.

function management data (FMD)
In SNA, a request unit (RU) category used for end-user data exchanged between logical units (LUs) and for requests and responses exchanged between network services components of LUs, physical units (PUs), and system services control points (SSCPs).

H

half duplex
In data communication, pertaining to an alternate, one way at a time, independent transmission (A); Contrast with duplex.

I

I/O Input/output.

ICV Initial chaining value.

ILU Initiating logical unit.

IMS/VS
Information Management System/Virtual Storage.

Information Management System/Virtual Storage (IMS/VS)
A general purpose system that enhances the capabilities of OS/VS for batch processing and telecommunication and allows users to access a computer-maintained data base through remote terminals.

initiating logical unit (ILU)
The logical unit that initiates a session with another logical unit or between two other logical units.

input/output (I/O)
(1) Pertaining to a device whose parts can perform an input process and an output process at the same time. (2) Pertaining to a functional unit or channel involved in an input process, output process, or both, concurrently or not, and to the data involved in such a process. ***Note:*** *The phrase input/output may be used in place of input/output data, input/output signals, and input/output process when such a usage is clear in context.* (3) Pertaining to input, output, or both.

instance
A copy of a transaction program that is executing on a given logical unit. If multiple instances are supported on a logical unit, multiple copies of the same transaction program can execute simultaneously.

Interactive System Productivity Facility (ISPF)
An IBM licensed program that serves as a full-screen editor and dialogue manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

intermessage delay
The elapsed time between receipt of a system response at a terminal and the time when a new transaction is entered. Synonymous with think time.

Inter-User Communications Vehicle (IUCV)
A VM facility for passing data between virtual machines and VM components.

IPM Isolated pacing message.

ISPF Interactive System Productivity Facility.

IUCV Inter-User Communications Vehicle (IUCV).

J

JCL Job control language.

job control language (JCL)
A problem-oriented language designed to express statements in a job that are used

to identify the job or describe its requirements to an operating system. (A)

L

LF Line feed.

line feed (LF)
The incremental relative movement between the paper carrier and the type carrier in a direction perpendicular to the writing line.

Log Compare Utility
A utility that enables WSim to compare 3270 display records from two log data sets and report the differences.

logic test
In WSim, a conditional test on an input or output message, a counter, or other item using theWSim IF statement. The IF actions can be used to control the message generation process.

logical unit (LU)
(1) A port through which a user gains access to the services of a network. (2) In SNA, a port through which an end user accesses the SNA network and the functions provided by system services control points (SSCPs). An LU can support at least two sessions—one with an SSCP and one with another LU—and may be capable of supporting many sessions with other logical units.

Loglist Utility
A utility that enables WSim to produce a formatted report of the log data set.

LU Logical unit.

LUSTAT
Logical unit status.

M

magnetic stripe reader (MSR)
A device that reads precoded information from a magnetic stripe. The device can be hand-held or fixed.

MDT Modified data tag.

message generation
In WSim, the process of executing statements that generate messages from the resources being simulated by WSim.

message generation statements
The collection of statements that define the actions to be performed by WSim, including message generation and logic testing.

MF Modify field.

MLU Multiple logical units.

modified data tag (MDT)
(1) An indicator, associated with each input or output/input field in a displayed record, that is set ON when data are keyed into the field. The modified data tag is maintained by the display device and can be used by the program using the file. (2) In 3270, a bit in each input field that, when set, causes that field to be transferred to the host system.

modify field (MF)
A 3270 data stream order that specifies the field and extended field attributes to be modified without having to respecify all attributes of the field.

module
A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to, or output from, an assembler, compiler, linkage editor, or executive routine. (A)

MSR Magnetic stripe reader.

MTRC
Message generation trace record.

Multiple Virtual Storage (MVS)
An IBM licensed program whose full name is the Operating System/Virtual Storage (OS/VS) with Multiple Virtual Storage/System Product for System/370*. It is a software operating system controlling the execution of programs.

MVS Multiple Virtual Storage.

N

NC Network control.

NetView Performance Monitor (NPM)
An IBM licensed program that collects, monitors, analyzes, and displays data relevant to the performance of a VTAM telecommunication network. It runs as an online VTAM application program.

network control (NC)
In SNA, an RU category used for requests and responses exchanged for such

purposes as activating and deactivating explicit and virtual routes and sending load modules to adjacent peripheral nodes.

network definition statements
A collection of statements that define the network configuration WSim uses when processing the message generation source statements.

network services (NS)
In SNA, the services within network addressable units (NAUs) that control network operation through SSCP-SSCP, SSCP-PU, and SSCP-LU sessions.

node (1) In SNA, an endpoint of a link or junction common to two or more links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. (2) In VTAM, a point in a network defined by a symbolic name.

NRF Network Routing Facility.

NS Network services.

O

OAF Origin address field.

operating system (OS)
Software that controls the execution of programs. An operating system may provide services such as resource allocation, scheduling, input/output control, and data management. ***Note:*** *Although operating systems are predominantly software, partial or complete hardware implementations are possible.* (A)

origin address field (OAF)
In SNA, a field in a FID0 or FID1 transmission header that contains the address of the originating network addressable unit (NAU).

OS Operating system.

P

PA Program attention.

partitioned data set (PDS)
A data set in direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

PDS Partitioned data set.

PEL Picture element.

PF Program function.

physical unit (PU)
In SNA, a type of network addressable unit (NAU). A physical unit (PU) manages and monitors the resources (such as attached links) of a node, as requested by a system services control point (SSCP) through an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links.

physical unit type
In SNA, the classification of a physical unit (PU) according to the type of node in which it resides. The physical unit type is the same as its node type; that is, a type 1 physical unit resides in a type 1 node, and so forth.

picture element (PEL)
(1) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (TC97) (2) The area of the finest detail that can be reproduced effectively on the recording medium.

PLU Primary logical unit.

primary logical unit (PLU)
In SNA, the logical unit (LU) that contains the primary half-session for a particular LU-LU session. Each session must have a PLU and secondary logical unit (SLU). The PLU is the unit responsible for the bind and is the controlling LU for the session. A particular LU may contain both primary and secondary half-sessions for different active LU-LU sessions. Contrast with secondary logical unit (SLU).

programmed symbols (PS)
In the 3270 Information Display System, an optional feature that stores up to six user-definable, program-loadable character sets of 190 characters each in terminal read/write storage for display or printing by the terminal.

PS Programmed symbols.

PU	Physical unit.

R

record (1) A set of data treated as a unit (TC97); for example, in stock control, each invoice could constitute one record. (2) In VTAM, the unit of data transmission for record-mode. A record represents whatever amount of data the transmitting node chooses to send. (3) In Series/1*, a portion of a data set accessed at the logical level (GET/PUT).

request/response header (RH)
In SNA, control information preceding a request/response unit (RU), that specifies the type of RU (request unit or response unit) and contains control information associated with that RU.

request/response unit (RU)
In SNA, a generic term for a request unit or a response unit.

request unit (RU)
(1) In SNA, a message unit that contains control information, such as a request code, or function management (FM) headers, end-user data, or both. (2) In DPCX, the smallest unit of data or control information.

resource
(1) Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs. (2) In the NetView program, any hardware or software that provides function to the network.

Response Time Utility
A utility that enables WSim to analyze response times for activities on the log data set.

response unit (RU)
In SNA, a message unit that acknowledges a request unit; it may contain prefix information received in a request unit. If positive, the response unit may contain additional information (such as session parameters in response to BIND session), or if negative, contains sense data defining the exception condition.

return code
A code used to influence the execution of succeeding instructions. (A)

RH	Request header or response header.
RR	Receive ready.
RU	Request unit or response unit.

S

SA Set attribute.

same-domain
Refers to communication between entities in the same SNA domain. Contrast with cross-domain.

SBI	Stop Bracket Initiation.
SC	Session Control.
script	See WSim script.

Script Generator Utility
A utility that enablesWSim to convert data captured from a system into message generation scripts.

SDT Start data traffic.

secondary logical unit (SLU)
In SNA, the logical unit (LU) that contains the secondary half-session for a particular LU-LU session. An LU may contain secondary and primary half-sessions for different active LU-LU sessions. Contrast with primary logical unit (PLU).

session control (SC)
In SNA, (1) One of the components of transmission control. Session control is used to purge data flowing in a session after an unrecoverable error occurs, to resynchronize the data flow after such an error, and to perform cryptographic verification. (2) A request unit (RU) category used for requests and responses exchanged between the session control components of a session and for session activation and deactivation requests and responses.

SFE	Start field extended.
SI	Shift In. Used with DBCS. This is the X'0F' character that ends DBCS data.

Simple TCP protocol (STCP)
In WSim, a protocol providing a means of transferring data to and from simulated

clients by way of TCP/IP connections without any manipulation of the data other than that provided by the script itself.

single-domain network
In SNA, a network with one system services control point (SSCP). Contrast with multiple-domain network.

SLU Secondary logical unit.

SNA Systems Network Architecture.

SO Shift Out. Used with DBCS. This is the X'0E' character that begins DBCS data.

SOH Start of header.

start of heading character (SOH)
A transmission control character used as the first character of a message heading. (A)

start of text character (STX)
A transmission control character that precedes a text and can be used to terminate the message heading. (A)

STCP Simple TCP Protocol.

STL Structured Translator Language.

STL Translator
In WSim, a utility that acts as the STL translator and translates STL statements into message generation source statements.

Structured Translator Language (STL)
A set of conventions and rules for writing syntactically allowable statements that will create message generation source statements.

STX Start of text.

Systems Network Architecture (SNA)
The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through and controlling the configuration and operation of networks.

T

TCP/IP
Transmission Control Protocol/Internet Protocol.

TH Transmission header.

think time
The elapsed time between receipt of a system response at a terminal and the time when a new transaction is entered. Synonym for intermessage delay.

time sharing option (TSO)
An optional configuration of the operating system that provides conversational time sharing from remote stations in a network using VTAM.

TP Transaction program.

transaction program (TP)
In WSim, a transaction program is any program that uses LU type 6.2 communication protocols to communicate with another program. Transaction programs are implemented in WSim using the CPI-C application program interface.

Transmission Control Protocol/Internet Protocol (TCP/IP)
A suite of protocols designed to allow communication between networks regardless of the technologies implemented in each network.

transmission header (TH)
In SNA, control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network.

TSO Time sharing option.

U

unformatted system services (USS)
In SNA products, a system services control point (SSCP) facility that translates a character-coded request, such as a logon or logoff request into a field-formatted request for processing by formatted system services and translates field-formatted replies and responses into character-coded requests for processing by a logical unit. Contrast with formatted system services.

user table
In WSim, one or more text data entries contained in a table format which may be referenced for logic testing and message generation.

USS Unformatted system services.

UTI User time interval.

V

Virtual Telecommunications Access Method (VTAM)
An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VTAM
Virtual Telecommunications Access Method.

W

ward 42
The portion of DBCS codes that corresponds to those of SBCS. The first byte is X'42'. The second byte is the hexadecimal value of the corresponding single-byte EBCDIC code.

WCC Write control character.

Workload Simulator (WSim)
IBM program product to simulate terminals and networks. It enables the user to test system performance and evaluate network design.

write control character (WCC)
(1) A control character that follows a write command in the 3270 data stream and provides control information for executing display and printer functions. (2) A character used in conjunction with a write-type command to specify that a particular operation, or combination of operations, is to be performed at a display station or printer.

write-to-operator (WTO)
An optional user-coded service that enables the writing of a message to the system console operator that informs the operator of errors and unusual system conditions that may need correcting.

WSF Write structured field.

WSim Workload Simulator.

WSim network
The set of statements defining an entire network, including both the network definition statements and the message generation source statements. Should not be confused with a packet switching network.

WSim script
The set of statements defining an entire network, including both the network definition statements and the message generation source statements.

WTO Write-to-operator.

X

XMIT Transmit.

Bibliography

The following manuals provide additional information about the definition and operation of networks simulated by WSim:

WSim Library

WSim User's Guide, SC31-8948

WSim Messages and Codes, SC31-8951

Creating WSim Scripts, SC31-8945

WSim Script Guide and Reference, SC31-8946

WSim Utilities Guide, SC31-8947

WSim User Exits, SC31-8950

WSim Test Manager User's Guide and Reference, SC31-8949

Related publications

IBM 3270 Information Display System, GA23-0204

IBM 3270 Information Display Unit, GV20-9707

Systems Network Architecture: Formats, GA27-3136

OS/VS2/ MVS System Programming Library: Utilities, GC26-3902

ACF/VTAM Installation and Resource Definition, SC27-0610

ACF/VTAM Network Implementation Guide, SC31-6419

Random Number Generation and Testing, GC20-8011

I/O Installation—Physical Planning for S/360 and S/370, GC22-7064

IBM TCP/IP for MVS User's Guide Version 3 Release 1 SC31-7136

Systems Application Architecture Common Programming Interface Communications Reference, SC26-4399-06. (WSim does not support CPI-C functions that have been added on later releases of this document.)

VTAM Programming for LU 6.2, SC31-6437

OS/VS2 JES2 Logic Manual, SY24-6000

IBM 5250 Information Display System Functions Reference Manual, SA21-9247

Index

Printed in USA

SC31-8945-01